

TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Lehrstuhl für Wirtschaftsinformatik (I 17)
Prof. Dr. Helmut Krcmar

**Model Integration and Traceability
for Product Service Systems Engineering**

Thomas Wolfenstetter

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität
München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Alexander Pretschner
Prüfer der Dissertation:	1. Prof. Dr. Helmut Krcmar
	2. Prof. Dr.-Ing. Birgit Vogel-Heuser
	3. Prof. Dr. Claudia Eckert

Die Dissertation wurde am 08.08.2018 bei der Technischen Universität München eingereicht
und durch die Fakultät für Informatik am 29.01.2019 angenommen.

for Xixi

Abstract

Motivation

The design and development of product service systems (PSS) is a complex enterprise that brings together various domains, such as product, software and service engineering. A fully integrated PSS calls for a deep collaboration among the different engineering domains over the whole PSS life cycle which can pose several challenges to the engineering team. In particular, it should be possible to trace the evolution of engineering artifacts along the complete life cycle starting from early stakeholder requirements to the final solution components. For this purpose, a requirements traceability (RT) model for PSS needs to take into account the special characteristics and complexities that are relevant in the context of PSS.

Research Approach

For the research presented in this thesis we followed the recommendations of Design Science Research as presented by Hevner (2004; 2007). In this process we conducted extensive literature reviews, studied several cases from different industries and iterated through the design cycle using reference modeling and software tool prototyping as research methods. Through this iterative process in which the aspired artifacts are enhanced and evaluated continuously, it was possible to comprehend the issue that was studied in all its details and thus evaluate the solution approach and evolve the artifacts until they solve the issues under consideration.

Results

In this thesis, we developed a traceability and model integration solution for PSS engineering consisting of a model integration ontology and software tool supporting traceability and model integration, which builds on top of that. Along this way, we identified characteristics that make PSS engineering special. On this fundament we determined, whether existing traceability approaches were suited for PSS engineering. Our analysis shows, that none of the existing approaches was recommendable for PSS without restrictions but we concluded that systematic combination and enhancement of those approaches offered great potential. Based on these results we introduce a concept for a cross-disciplinary model integration ontology as well as a conceptual methodology for model transformation. Finally, we present our prototypical software tool TRAILS, which implements the concepts developed in this thesis and offers support to engineers in terms of integrating domain-specific models of a PSS, analyzing them and capturing trace links between the various model artifacts captured.

Contribution

Overall, we believe that the analyses, concepts and solutions presented in this thesis contribute to requirements engineering, model-based systems engineering and product service systems research. Researchers can build upon our analysis of traceability approaches in various domains to develop methods that fit the need of complex cross-disciplinary engineering projects. In this context, our approach facilitates the integration of domain-specific models by abstracting from specific details of the modeling language or data format that are not relevant in the integrated model perspective. In this sense, researchers can use our

model integration ontology as a blueprint approach for linking knowledge in model-based systems engineering. With our approach we also bring a certain degree of formalism into the development of PSS, thus justifying the term “PSS engineering”. Using model-based approaches to specify and document PSS engineering artifacts, it is possible to detect conflicts between different solution components of a PSS much easier and often earlier in the development process.

Study Limitations

Taking an objective look at our research, we have to admit that there are certain limitations regarding our overall research approach, our model integration ontology and our prototypical software tool, TRAILS. By its nature, Design Science Research aims more at building a functional model integration ontology along with a working prototype of our software tool TRAILS, rather than trying to find the optimal structure of the ontology or the implementing the tool with the optimal technology and runtime execution efficiency. Furthermore, we focused on a limited number of domain-specific modeling approaches when developing our model integration ontology and implementing the corresponding model integration features in TRAILS. Also, within the scope of this thesis it was not feasible to conduct extensive empirical evaluations regarding the advantages of our solution in real industry case studies. Instead, we studied its applicability in one detailed case study of developing a bike sharing system.

Future Research

Overall, we see three major starting points for future research to advance from. First to mention is the extension of the integration ontology in order to cover additional artifact types and augment its applicability for additional use cases. Second, we think that there is great potential in the enhancement of TRAILS, our prototypical traceability and model integration software tool by adding additional features and improving the existing ones. Finally, the third point is a detailed empirical evaluation of our results in terms of their performance in real industry cases.

Table of Contents

Abstract	III
Table of Contents	V
List of Figures	X
List of Tables.....	XII
List of Abbreviations.....	XIII
PART A: INTRODUCTION TO THE DISSERTATION'S PUBLICATIONS	1
1 Introduction	2
1.1 Motivation	2
1.2 Problem Statement.....	4
1.2.1 Issue 1: Characteristics of PSS Engineering	5
1.2.2 Issue 2: Common Representation of Artifacts	5
1.2.3 Issue 3: Cross-domain Traceability and Change Management.....	6
1.2.4 Issue 4: Different Engineering Cycles	7
1.3 Research Questions.....	8
1.4 Structure.....	11
2 Conceptual Background	16
2.1 Requirements Traceability.....	16
2.1.1 Imperative for traceability and cross-domain issues	16
2.1.2 Scope of requirements traceability.....	17
2.1.3 Perspectives on traceability	19
2.1.4 Utilization of traceability information.....	21
2.2 Product Service Systems	24
2.2.1 Terminology	25
2.2.2 Types of PSS	26
2.2.3 Development of PSS	27
2.2.4 Need for Integration in PSS engineering.....	28
2.2.5 Advantages of PSS business models.....	29
2.3 Model-based Systems Engineering	30
2.3.1 Traditional engineering vs. model based engineering.....	30
2.3.2 Modeling languages in PSS engineering.....	31
2.3.3 Conceptual Modeling: Reference Models, Meta Models and Ontologies	32
2.3.4 Model Integration and Model Transformation.....	34

2.3.5	Using Semantic Web Technologies in Engineering.....	35
3	Research Approach	38
3.1	Research Strategy	38
3.2	Research Methods.....	40
3.2.1	Literature Review and Expert Interviews.....	40
3.2.2	Ontology Development	41
3.2.3	Tool prototyping and evaluation	43
PART B: PUBLICATIONS		46
Publication 1: Why Product Service Systems Development is Special.....		50
1.	Introduction	50
2.	Methodology	52
3.	Special Characteristics of PSS Development.....	52
3.1.	Integration of Components.....	55
3.2.	Multidisciplinary Development.....	55
3.3.	Customer Integration.....	56
3.4.	Lifecycle Orientation.....	56
3.5.	Variability of Service Delivery	58
3.6.	Individualization.....	58
3.7.	Organizational Challenges	58
3.8.	Value Network	59
3.9.	Sustainability.....	59
4.	Discussion	60
5.	Conclusion.....	63
Publication 2: Analyse der Eignung domänenspezifischer Methoden der Anforderungsverfolgung für Produkt-Service-Systeme.....		64
1.	Ausgangssituation und Problemstellung	65
2.	Grundlagen der Anforderungsverfolgung bei PSS.....	66
3.	Forschungsdesign	67
4.	Methoden der Anforderungsverfolgung.....	68
5.	Bewertungskriterien	71
6.	Bewertung der Anforderungsverfolgungsmethoden	73
7.	Diskussion	76
8.	Zusammenfassung und Ausblick	77
Publication 3: Towards Cycle-Oriented Traceability in Engineering Change Management...		79

1. Introduction	80
2. Research Methodology	80
3. Overview: Requirements Management and Engineering Change Management	81
3.1. Requirements Management and Requirements Engineering	81
3.2. Engineering Change Management	83
3.3. Issues regarding the interface between requirements management and engineering change management	84
4. Implications for the interface between requirements management and engineering change management	85
5. Cycle-oriented traceability for the management of changes	86
5.1. Theoretical background to traceability	86
5.2. A data model for traceability in engineering change management	88
5.3. Academic example	90
6. Conclusion and outlook	90
Publication 4: Traceability von Anforderungen und Tests in agilen Softwareentwicklungsprojekten	92
1. Motivation	93
2. Methodik	94
3. Bestehende Ansätze zu agiler Traceability	95
4. Fallstudie: Agile Softwareentwicklung bei Alpha	97
4.1. Der agile Entwicklungsprozess	97
4.2. Herausforderungen für Traceability im agilen Entwicklungsprozess	99
5. Ein konzeptuelles Datenmodell für Traceability in agilen Projekten	100
6. Diskussion	103
7. Limitationen und Ausblick	105
Publication 5: Concept for an Integration-Framework to enable the crossdisciplinary Development of Product-Service Systems	107
1. Introduction	108
2. State of the Art	108
2.1. Modeling-Approaches	108
2.2. Methods for interdisciplinary system modeling and information exchange	110
3. Integration-Framework	112
4. Application Example	114
5. Conclusions and Outlook	117

Publication 6: Supporting the cross-disciplinary development of product-service systems through model transformations	118
1. Introduction	119
2. State of the Art	119
3. Model Transformations	122
4. Basic Concepts of the PSS-IF	123
5. Transformation Process	126
6. Conclusion and Outlook	126
Publication 7: Towards a Requirements Traceability Reference Model for Product Service Systems.....	128
1. Introduction	129
2. Research Design	130
3. Results	131
3.1. General Model Constructs.....	131
3.2. Development Artifacts	132
3.3. Generic Stakeholders.....	133
3.4. Requirements.....	135
3.5. Specification Artifacts.....	136
3.6. Management Artifacts	137
3.7. Solution Artifacts	138
3.8. Structure Elements	139
4. Exemplary Use Cases.....	140
5. Discussion	143
6. Conclusion and Outlook.....	144
Publication 8: Introducing TRAILS: A Tool supporting Traceability, Integration and Visualisation of Engineering Knowledge for Product Service Systems Development	146
1. Introduction	147
1.1. Motivation	147
1.2. Approach	149
1.3. Structure of Article.....	150
2. Related Work.....	150
2.1. PSS Modelling Methods.....	150
2.2. PSS Computer Aided Modelling Tools.....	151
2.3. Implications for Comprehensive PSS Engineering Tool Support.....	152

3.	Model Integration.....	152
3.1.	Model Transformation.....	153
4.	TRAILS Integration Method.....	155
4.1.	Model Integration Ontology.....	155
4.2.	Model Transformation Process	159
5.	TRAILS Features	161
5.1.	Importing Models.....	161
5.2.	Merging Models	162
5.3.	Adaptable Cross-domain Model Integration Ontology.....	163
5.4.	Editing Models	164
5.5.	Customizable Appearance and Standard Graph Layouts.....	164
5.6.	Customized Filtering and Viewpoint Creation.....	165
5.7.	Matrix View and Spreadsheet Integration.....	165
5.8.	Multi-user Capabilities and Database Server	166
6.	Case Study: Bike Sharing System.....	166
7.	Discussion	174
8.	Conclusion and Future Work	177
PART C: DISCUSSION		180
1	Discussion	181
1.1	Summary of Findings	181
1.2	Implications for Research.....	184
1.3	Implications for Practice.....	187
1.4	Limitations.....	188
1.5	Future Research	191
2	Conclusion.....	196
References		197

List of Figures

Figure 1: Main issues regarding traceability for PSS engineering.....	5
Figure 2: Structure of this dissertation	15
Figure 3: Scope of Traceability	18
Figure 4: Research perspectives on requirements traceability	20
Figure 5: Development of Product Service Systems.....	28
Figure 6: Semantic Web Technology Stack.....	36
Figure 7: Cycles in Design Science Research applied to the dissertation topic.....	39
Figure 8: Overview of the bike sharing PSS case study: PSSycle.....	45
Figure 9: Nine Characteristics of PSS development	61
Figure 10: Dimensionen der Anforderungsverfolgung bei PSS.....	67
Figure 11: Considered activities of RE	82
Figure 12: Generic engineering change process	83
Figure 13: Dependencies between requirement artifacts (RAs) and solution artifacts (SAs)..	84
Figure 14: Change cycles within and between RM and ECM	86
Figure 15: Data model for traceability in engineering change management	89
Figure 16: Schematischer Ablauf des agilen Entwicklungsprozesses bei Alpha.....	97
Figure 17: Herausforderungen für Traceability im agilen Entwicklungsprozess	99
Figure 18: Konzeptuelles Datenmodell für Traceability in agilen Projekten.....	101
Figure 19: Specification of the elements for the integration-framework	113
Figure 20: Excerpt of the mapping concept for the eBike-sharing-system.....	116
Figure 21: General types of transformation possibilities	123
Figure 22: Excerpt of the PSS-IF meta-model	124
Figure 23: Schematic representation of the transformation process	126
Figure 24: General Model Constructs	132
Figure 25: Development Artifacts Submodel.....	133
Figure 26: Generic Stakeholders Submodel.....	134
Figure 27: Requirements Submodel.....	135
Figure 28: Specification Artifacts Submodel	136
Figure 29: Management Artifacts Submodel	137
Figure 30: Solution Artifacts Submodel.....	138
Figure 31: Structure Elements Submodel.....	139
Figure 32: Exemplary Use Case: Traceability of Requirements Refinement	141
Figure 33: Exemplary Use Case: Traceability of Changes	142
Figure 34: TRAILS Model Integration Ontology: Edge Types	156
Figure 35: TRAILS Model Integration Ontology: Node Types.....	157
Figure 36: Generic Transformation Operators	159
Figure 37: SysML block diagram of the stationary bike sharing system.....	167
Figure 38: EPC of the rental process in the stationary bike sharing system	168
Figure 39: Excerpt of requirements document for free floating bike sharing system.....	168
Figure 40: SysML block diagram of the free floating bike sharing system.....	169
Figure 41: EPC of the rental process in the free floating bike sharing system	170
Figure 42: Configuration of comparison algorithms for merging models	171

Figure 43: Merging Results of Requirements with Block Diagram for a Stationary Bike
Sharing System..... 172
Figure 44: Linking Requirements to Solution Components..... 173
Figure 45: Linking activities to system components..... 174

List of Tables

Table 1: Relation between Research Question/Publications, Issues of the Problem Statement and Research Areas at Focus.....	11
Table 2: Definitions of the term Product Service System.....	25
Table 3: Modeling Languages in PSS engineering.....	32
Table 4: Publications included in this dissertation.....	47
Table 5: Further publications by the dissertation's author.....	48
Table 6: Special Characteristics of PSS Development.....	53
Table 7: Zusammenfassung der Methodenbewertung.....	74
Table 8: Classes of traceability links.....	87
Table 9: Summary of existing approaches for interdisciplinary modeling and information exchange.....	111
Table 10: Cross-Domain Modeling Approaches.....	120
Table 11: Characteristics of Discipline-Specific and Cross-Discipline Modeling.....	121
Table 12: General Types of Model Transformations.....	154

List of Abbreviations

BPMN	Business Process Modeling Notation
CAD	Computer-aided Design
DSL	Domain-Specific Language / Discipline-Specific Language
DSR	Design Science Research
EC	Engineering Change
ECM	Engineering Change Management
E/E	Electrics and Electronics
EPC	Event-driven Process Chain
ERM	Entity Relationship Model
FEM	Finite Element Analysis
FMEA	Failure Mode and Effects Analysis
FOAF	Friend of a Friend
HW	Hardware
IL	Intermediary Language
IT	Information Technology
MBSE	Model-based Systems Engineering
MDD	Model-driven Development
OWL	Web Ontology Language
PDM	Product Data Management
PSS	Product Service System
RA	Requirements Artifact
RC	Requirements Change
RDF	Resource Description Framework
SA	Solution Artifact
SW	Software
SysML	Systems Modeling Language

TRAILS	Traceability, Model Integration and Lifecycle Management Support
UML	Unified Modeling Language
URI	Uniform Resource Identifier
WP	Workpiece
XMI	XML Meta-data Interchange
XML	Extensible Markup Language

PART A: INTRODUCTION TO THE PUBLICATIONS

1 Introduction

This doctoral thesis is concerned with the special challenges that the integrated development of product service systems (PSS) imposes on traceability of artifacts in the PSS lifecycle in general but with a special focus on requirements traceability. For this purpose, the thesis first gives a theoretically as well as practically grounded overview of the issues that arise due to the multidisciplinary development process that PSS require. On this basis, the types of artifacts that need to be traced as well as the types of semantic relationships (so-called “trace links”) between these artifacts are identified and specified in the form of a model integration ontology that supports ensuring traceability in PSS engineering. Finally, this thesis introduces a prototypical software tool to support traceability throughout the entire lifecycle of PSS. The following chapter motivates this doctoral thesis and provides detailed problem statements for the research questions addressed in this dissertation.

1.1 Motivation

Due to the ongoing technological evolution that shapes the face of the manufacturing industry, especially high tech products, become increasingly complex (El Maraghy et al. 2012). While in the 1960ies for example, a car consisted of only few electric components, today’s cars have turned into computers on wheels as they have to fulfill more and more customer requirements regarding infotainment, driving assistance (e.g. automatic braking assistance, lane keeping assistance or stability programs), internet connectivity or other comfort features, such as automatic parking or driver recognition. While in the beginning of public aviation airplanes used to be merely mechanical systems, they nowadays have to satisfy various entertainment requirements and some do not even need a pilot as they are able to fly on their own. Also manufacturing systems that used to be simple conveyor belts running at a predefined speed have become automated and intelligent in order to cope with the trend towards mass-customization, just-in-sequence delivery and the increasing use of assembly robots.

All of these examples require a tight integration of hardware and software components as they evolve to so-called cyber physical systems. The technological evolution therefore comes along with complex dependencies between the single system components as they are dependent on each other in order for the systems as a whole to function. Being able to effectively manage these dependencies is a not only an essential capability during development but also during operation, i.e. throughout the whole system lifecycle. However, the more complicated a system becomes, the more complex it is to manage its operation, maintenance and finally, disposal. All of these activities are thus increasingly dependent on experts.

Furthermore, in an increasingly globalized world, companies, especially manufacturing companies, find themselves confronted with fierce competition in which it is hard to differentiate oneself just on the basis of their products. In most of the cases there are a large number of competitors that are capable of delivering a product of comparable quality. This often leads to ruinous price wars causing margins to collapse (Becker and Krcmar 2008).

Therefore, many manufacturing companies begin to offer services in order to differentiate themselves from their competitors (Fritzsche 2007). In the course of this development, more and more companies realize that customers are not interested in products or services per se but rather expect a solution to a problem that they are confronted with or the fulfillment of a demand they have (Leimeister and Glauner 2008; Sawhney 2006). This means that they transform their business models into providing specific solutions for their customers (Davies et al. 2006a; Galbraith 2002). Often these solutions consist of an integrated bundle of hardware, software and services, commonly known as product service system (PSS) (Baines et al. 2007; Boehm and Thomas 2013).

When developing an integrated solution that incorporates components from multiple engineering domains such as mechanical engineering, software engineering and service engineering, it does not make sense to separate the development into domain-specific processes. Instead, the development of those systems is a challenging task that calls for an integrated multi-domain engineering process that comprises mechanical, software and service engineering (Hepperle et al. 2010).

Since PSS are to a large degree customer specific solutions the central point of reference for PSS development are the customer's demands which are to be fulfilled by the solution (Burianek et al. 2007). Due to the fact that customers mostly just have a rough idea of what they really need, it is the developer's task to determine and specify the complete requirements for the intended PSS (Sauerwein et al. 1996; Holtzblatt and Beyer 2013). Moreover, not only the prospective customer's requirements need to be considered but also those of other stakeholders such as experts from the different engineering domains, regulatory bodies or business partners. All these requirements need to be elicited, analyzed, structured, refined and specified in a way that they can be understood by the engineers who develop the various solution components (Cheng and Atlee 2007; Kotonya and Sommerville 1998). Requirements engineering therefore plays a pivotal role in PSS development (Berkovich et al. 2011c; Spath and Demuß 2006).

This situation is aggravated by the fact that engineering artifacts and in particular requirements do not remain consistent during the development process but rather are subject to changes and evolve over time. In fact, it can be noted that the entire lifecycle of a PSS is subject to internal and external cyclic influences that cause changes from time to time. Those changes manifest themselves in changing requirements and consequentially changing designs and specifications, which are to be dealt with. In order to cope with these changes, the evolution of the requirements base and its manifestation in the various PSS components needs to be monitored. This challenge is commonly referred to as requirements traceability (RT) (Gotel and Finkelstein 1994; Ramesh and Jarke 2001). There is a large number of reasons, why traceability is desirable in any engineering project. For example, it is possible to comprehend and reenact which engineering artifacts have been changed at which point during the development and what were the reasons for that. The need for systematically managing traceability is generally accepted in all engineering domains. However, there is no approach that addresses the special challenges that arise in the context of traceability for integrated development of PSS which involves multiple engineering domains, each relying on it's

domain-specific approaches (Berkovich et al. 2011b). In this dissertation, we present a model integration and traceability approach along with a prototypical software tool tackling the issues of PSS engineering.

1.2 Problem Statement

The concept of requirements traceability has been around for decades. In the field of software engineering the term has been around as early as 1975 when researchers were concerned about ensuring software reliability (Williams 1975), especially in the aviation sector and for military purposes. Since then, the need for requirements traceability has been widely acknowledged in many areas of software and systems engineering, especially if the systems under development are safety critical or if their failure would lead to severe adverse effects. The reason for this can be found in the many advantages that proper traceability offers for engineers, especially for tasks such as impact analysis, change management or project management in general. We discuss those advantages in section 2.1.4.

As a consequence, various domain-specific methods and tools that ensure the traceability of requirements fulfillment have been developed and rolled out to industry. Examples for traceability software tools are Agosense¹, IBM Rational DOORS², PTC Integrity³, Orcanos Traceability Management⁴, Tracecloud⁵ or Yakindu⁶. Commonly, these approaches focus on the development of either a physical product or a software system. However, as many businesses aspire offering integrated PSS, challenges that arise in the context of requirements traceability largely differ from traditional product development. In order to provide methods and tools that ensure traceability in PSS engineering it is therefore important to precisely reflect on issues which arise due to the various domains involved in PSS engineering and the dynamic environment in which it takes place.

The research presented in this dissertation was conducted within the collaborative research center “Sonderforschungsbereich 768 – Managing cycles in innovation processes – Integrated development of product-service-systems based on technical products”. Based on experiences made in this collaborative research center that involves researchers and engineers from many scientific disciplines, we identified four major issues that influence traceability in the context of PSS engineering (c.f. Figure 1). We discuss those issues in the following.

¹ <http://www.agosense.com/traceability>

² <https://www.ibm.com/us-en/marketplace/cloud-requirements-management>

³ <http://www.ptc-de.com/application-lifecycle-management/integrity>

⁴ <http://www.orcanos.com/compliance/requirements-traceability-tool/#>

⁵ <https://www.tracecloud.com>

⁶ <https://www.itemis.com/en/yakindu/traceability/>

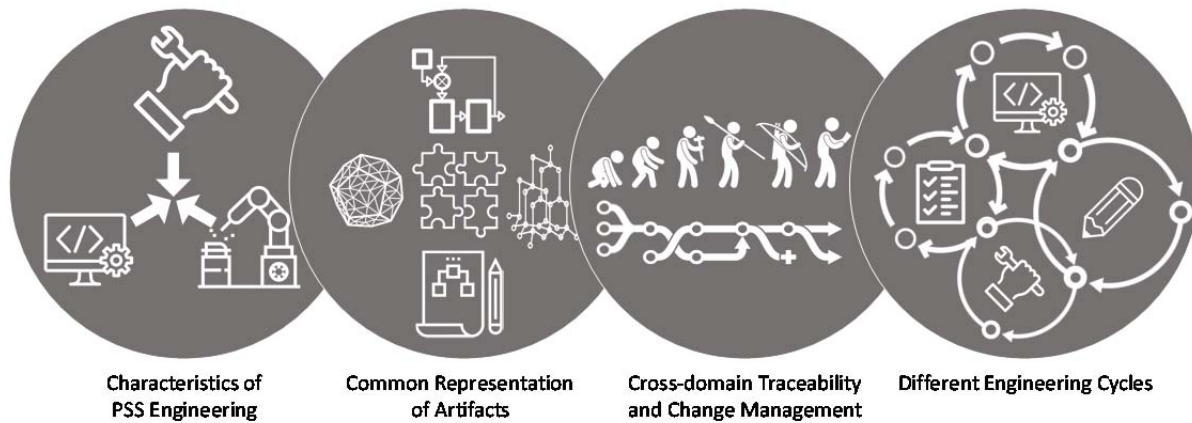


Figure 1: Main issues regarding traceability for PSS engineering

Source: Own illustration

1.2.1 Issue 1: Characteristics of PSS Engineering

When developing a PSS, the starting point is to examine the individual needs of potential customers. For this purpose, PSS providers need to identify and interpret the customers' requirements in detail and translate these into domain-specific engineering instructions (requirements on a more detailed technical level) that can be understood by mechanical engineers, electrical engineers, software developers as well as service designers. Along this way, the initially vague descriptions of needs and the boundary conditions that the solution has to conform to undergo a series of refinement and consolidation steps. This means that throughout the requirements engineering process, requirements are abstracted, detailed, translated, separated, combined, supplemented or discarded until a comprehensive description of the problem domain is found that can be turned into a corresponding solution. Furthermore, particularly in the business-to-business area, PSS providers need to understand the business model and the corresponding business processes of their customers. Otherwise, it is merely impossible to integrate the PSS offering into the value creation processes of their customers (Böhm and Krömer 2007; Tuli et al. 2007). This means that PSS development not only needs to design the components that fulfill a specific function when combined, but also develop the service processes in which those components are being used as well as the surrounding business model that is designed to deliver value in use to the customer.

1.2.2 Issue 2: Common Representation of Artifacts

As mentioned previously, PSS engineering involves various domains, such as mechanical engineering, electrical engineering, software engineering or service engineering. In all these domains engineers have to cope with different challenges. In order to do so they follow different engineering techniques, use different tools and consequentially produce different types of artifacts that are represented in different formats.

For example, while engineering the control software for an automated production line developers might produce C# code that is based on architectural descriptions using UML class diagrams while the mechanical engineers responsible for the physical components, such

as actuators produce 3-dimensional computer-aided design (CAD) models that are simulated using finite element models (FEM) and validated using a failure mode and effects analysis (FMEA). At first glance, the solutions produced by the mechanical engineers and software engineers seem largely independent of each other. However, taking a deeper dive into the solution design one recognizes that the development of such cyber-physical systems requires that the software engineer needs to know what the hardware looks like and vice versa. Again, if we take into account the service and business model boundaries that are involved in offering such a production line as a PSS (e.g. on a pay-per-item manufactured basis), software as well as hardware engineers need to know the overall business specifications (e.g. service blueprint, business process model) to be able to deliver the right functionality for the specific business case.

In conclusion, integrated PSS engineering requires that artifacts from different engineering domains need to be represented in a common format that allows to link related model elements (requirements and solution artifacts that fulfil those requirements) or representations of the same ontological concept in different engineering artifacts (e.g. a certain component in a CAD model with its representation in a UML class diagram). By doing so, it is for example possible to determine whether requirements are being regarded by the solution design or, as discussed in the next section, evaluate how changing one component within the systems impacts other components.

1.2.3 Issue 3: Cross-domain Traceability and Change Management

The prime goal of a PSS is to present an adequate solution to a specific problem or need a customer has. Therefore, the PSS needs to be adapted, since this need, the problem or the customer base evolve over time. As a PSS is primarily a promise for value-in-use, the provider usually guarantees the availability of the PSS functions. As customer needs change over time, laws are revised, technologies emerge and competition steadily varies, the environment in which a PSS operates is highly dynamic.

While traditional products are usually replaced by new generations within the product line in order to adapt to an evolving environment, the PSS business model is usually oriented towards establishing long term customer relationships and therefore needs to be continuously adapted and enhanced. This means that components of the PSS need to be replaced, refurbished, exchanged, updated, added or removed while under operation (Huang and Mak 1999). In order to assure that the PSS is able to still provide service it was designed for, all adaption processes need to be managed and potential effects of change need to be anticipated. The management of these processes of adaption can be summarized in the concept of engineering change management (Huang et al. 2001). Engineering change management (ECM) includes managing, executing and monitoring all change processes to have an impact on the system under consideration (Jarratt and Clarkson 2005).

As argued before, PSS development involves multiple engineering domains. In the final PSS, the solution components produced by these domains need to be integrated seamlessly in order to work together properly. This means that they are highly inter-dependent. Changing, adding or removing a solution component might therefore have an impact not only in the same

domain (e.g. software) but on the other solution domains of the PSS as well. For example, updating a certain software component might lead to the point that service processes that had been considered as independent cannot be executed any longer as they rely on information indirectly provided by this piece of software. In order to avoid any adverse effects that result from engineering changes, in PSS development they need to be managed across domain borders, analyzing direct and indirect impacts of various types within the PSS as a whole.

1.2.4 Issue 4: Different Engineering Cycles

The development of new products is subject to a large variety of influences, originating either in the development process itself or the overall environment, e.g. the target market. These external and internal influences which lead to changes in the innovation process are often of cyclic nature, i.e. they follow a pattern that keeps repeating. Additionally, cycles in innovation processes are not isolated from each other. The same way a PSS is a complex system consisting of various interdependent components, different cycles during the innovation process influence each other as well (Langer and Lindemann 2009). These effects are especially severe in PSS engineering, where many different domains each including various stakeholders have to be coordinated, thus making their management a complex task.

A common external cyclic influence on the development of PSS is the continuous evolution of the stakeholder's requirements forcing PSS providers to adapt the solution components that constitute the PSS. For example, evolving customer needs, the availability of new technologies or the change of legal regulations can lead to new or changing requirements (Berkovich et al. 2011b). These changes lead to iterations of requirements engineering activities as well as solution design and test activities causing the need for further coordination between different stakeholders in various domains. Especially changes of major customer requirements that are recognized relatively late in the development process can cause significant rework and thus being a major cost factor and delay market entry (Berkovich et al. 2011c).

By providing the means of following the dependencies between different engineering artifacts such as requirements or solution components, the impact of a change can be assessed more accurately. Therefore, an important approach to mitigate the undesirable effects of such late and unforeseen change is to ascertain traceability throughout the innovation process (Strens and Sugden 1996). Traceability can support the management of cycles by revealing the dependencies between different kinds of artifacts that are produced throughout the lifecycle of a PSS. Since cycles follow a repeating pattern they can then be managed. However, in order to manage cycles, their impact on the innovation process as well as the various interdependencies in the innovation process and the PSS itself first needs to be understood, modelled and anticipated.

Schenkl et al. (2013) identify the modelling of interdependencies between cycles and their influences on the innovation process as a driving challenge for further research in PSS engineering. Since traceability is concerned with the identification, analysis and documentation of dependencies, integrating the traceability techniques with an approach to

managing cycles in innovation processes can be regarded highly beneficial to complex engineering projects.

1.3 Research Questions

The overall goal of this thesis is to develop a traceability approach that is targeted at solving the issues that arise in the context cross-domain engineering of PSS. In the last section four central issues were discussed, namely (1) the special characteristics of PSS engineering, (2) the common representation of artifacts, the ability to support (3) cross-domain change management and (4) different engineering cycles in the various domains. To solve these issues, in this thesis we answer eight research questions that build on one another and in combination form the path to a comprehensive traceability approach for PSS.

A first important milestone on this path is to clarify the challenges that the development of PSS imposes on requirements traceability. As described by Tan et al. (2010), PSS providers shift their “business strategy from a product-oriented to a service-oriented focus, where instead of the product itself, the activity, its utility and performance associated with the use of the product are considered to be of value to the customer.” This way, value creation in a PSS context cannot be seen from a merely transaction-based perspective, but has to be seen as an interactive and relational process in which the customer acts as a value creation partner (Tuli et al. 2007). This fact of course also impacts the development process of a PSS and therefore the requirements for traceability. To determine how the differences between PSS and traditional products or services influence the engineering process we aim to address the following research question:

RQ1: *What differentiates product service systems engineering from traditional product or service engineering?*

Traceability is a topic that is not only relevant for PSS development. In various engineering domains the importance of tracing the evolution and satisfaction of requirements has already been known for decades. For example, manufacturers of safety critical systems, such as railway signaling and control systems, medical dialysis machines or avionics systems are forced by law to prove that all legally mandatory requirements are satisfied and their products work correctly. Also in many software development projects, especially in the business-to-business or business-to-government sector, customers often demand a proof that all their requirements are met. Having identified what makes the development of PSS so special it is important to evaluate whether existing requirements traceability approaches already resolve the challenges that are prominent in PSS development. We therefore ask the research question of:

RQ2: *What are the merits and shortcomings of existing domain-specific requirements traceability approaches?*

Having analyzed the state-of-the-art in requirements traceability research as well as the challenges related to the characteristics of PSS development our initial insights show that in order to provide a suitable traceability approach for PSS development especially two issues need to be solved: (1) the need for seamless integration of PSS components that are developed

by different engineering domains and (2) the alignment of different development methods (e.g. scrum vs. waterfall development) that are applied in these domains. In order to tackle the first issue, we take the use case of engineering change management across different domains and aim at identifying the necessary information about domain-specific artifacts and the trace links between those artifacts that ensure traceability in engineering change management. To resolve the second issue, we primarily focus on agile development methodologies.

Traditional development methodologies such as a waterfall process or the v-model feature long requirements analysis and concept development phases. Therefore, detailed process documentation and system models which can be used to extract traceability information. Agile methodologies on the other hand are more focused on rapid creation of prototypes and mostly avoid heavy documentation. This way, it is much harder to collect the needed traceability information. However, if traceability for the PSS as a whole is wanted, we have to analyze which kind of information can be collected in agile development and describe the traceable artifacts of agile development in a conceptual traceability model. We thus want to answer the following research questions:

RQ3a: *What is a suitable model for traceability in engineering change management?*

RQ3b: *What is a suitable model for traceability in agile development projects?*

Having identified the different types of artifacts that are relevant for traceability purposes as well as the types of semantic relationships in-between those, the next logical step is to take a detailed look at how those artifacts are composed in terms of their logical structure, namely their meta model. Providing requirements traceability presupposes identifying and documenting the relationships between different kinds of engineering artifacts, such as structural system models, requirements specifications, change requests or use case diagrams. It is therefore inevitable to find a common conceptual model that is able to describe the content of different kinds of domain-specific engineering artifacts and represent the trace links between those. Such a data structure can be either text-based (e.g. simple cross-references in a natural language text document), matrix-based (e.g. traceability matrices) or graph-based (e.g. SysML Requirements Diagram). Each of these forms of representation has its advantages and disadvantages. In order to find the best way of representing the various domain-specific engineering artifacts in an integrated model, we answer the following research question:

RQ4: *What is a suitable approach for integrating different PSS engineering artifacts under a common conceptual model?*

At this stage we have defined which artifacts need to be regarded in order to ensure traceability in PSS engineering as well as how to represent them in an integrated manner. The next step on the road to a comprehensive approach for traceability in PSS engineering is to create a template for the relationship network that is essential to traceability. This means that we have to develop a reference model that defines the generic traceability relationships in PSS engineering, i.e. artifacts and trace links. We thus ask the corresponding research question:

RQ5: *What is a suitable requirements traceability reference model for PSS?*

Speaking figuratively, the PSS Integration Framework which resulted from answering RQ4 equals the “words” that constitute the language in which we need to translate the domain-specific models. The result of answering RQ5 on the other hand can be seen as the grammar of this new language. Since at this point we have a comprehensive language that is able to express the necessary traceability relations in PSS engineering, we can now start to compose the “dictionaries” that are needed for translating from the various domain-specific models (or modeling languages respectively) into our common PSS representation, i.e. the PSS model integration ontology. Hence, we want to answer the following research question:

RQ6: *How can different PSS engineering artifacts be mapped onto the ontology defined in RQ4 and RQ5?*

One way to show the practicability of our approach for realizing requirements traceability in PSS engineering is to develop a software tool that supports our approach and show its implementation in practice using a realistic PSS development process as a use case. This software tool would need to be able to import different types of domain-specific models (behavioral as well as structural models), map these models onto the conceptual structure of our PSS model integration ontology and merge multiple models from different domains into one comprehensive semantic traceability network. Furthermore, it should display this network in a visually appealing fashion to a user and offer functions to interact with this semantic traceability network and analyze it. Therefore, we state our final research question:

RQ7: *What would be a suitable software tool to support requirements traceability in PSS engineering?*

In Table 1 we give an overview over the research questions, the publications that are part of this dissertation, the issues of the problem statement we address as well as the research areas that are at focus in each of the publications.

Table 1: Relation between Research Question/Publications, Issues of the Problem Statement and Research Areas at Focus

RQ	Publication	Issue of the problem statement addressed				Research area at focus		
		Characteristics of PSS Engineering	Common Representation of Artifacts	Cross-domain Traceability & Change Management	Different Engineering Cycles	Requirements Traceability	Model-based Systems Engineering	Product Service Systems
RQ1	Why Product Service Systems Development is Special	X						X
RQ2	Analyse der Eignung domänenspezifischer Methoden der Anforderungsverfolgung für Produkt-Service-Systeme	X		X		X		X
RQ3a	Towards Cycle-Oriented Traceability for Engineering Change Management			X	X	X		
RQ3b	Traceability von Anforderungen und Tests in agilen Softwareentwicklungsprojekten			X	X	X		
RQ4	Concept for an Integration-Framework to enable the cross-disciplinary Development of Product-Service Systems	X	X				X	X
RQ5	Supporting the crossdisciplinary development of product-service systems through model transformations		X				X	X
RQ6	Towards a Requirements Traceability Reference Model for Product Service Systems	X	X	X		X	X	X
RQ7	Introducing TRAILS: A Tool supporting Traceability, Integration and Visualisation of Engineering Knowledge for Product Service Systems Development	X	X	X	X	X	X	X

1.4 Structure

As illustrated in

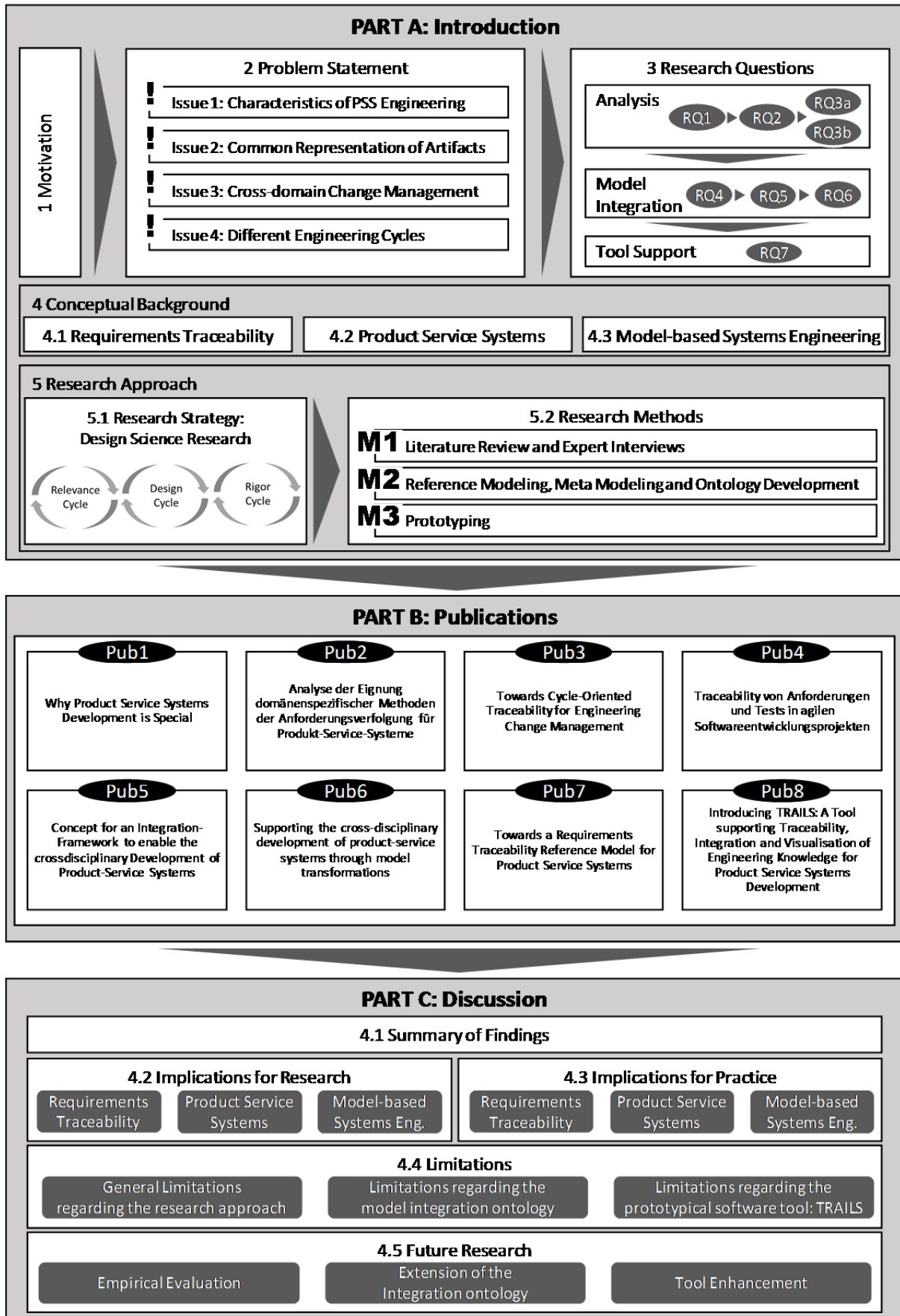


Figure 2, this dissertation is structured in three parts: An introduction to this thesis (Part A), the publications contained in this thesis (Part B) and a discussion of the research results (Part C).

In Part A we first motivate the research topic against the background of current trends and developments that impose challenges on engineering complex technical systems (section 1.1). On this basis, in section 1.2 we illustrate the four major issues that influence PSS engineering today. As a next step, we hereof derive our research questions and explain each of these questions in section 1.3. Next, in section 2, we introduce the conceptual background of this thesis. Here, we focus on requirements traceability (section 2.1), product service systems (section 2.2) and the paradigm of model-based systems engineering (section 2.3). Having laid out the fundamental concepts of the research areas that this thesis is subject to, we explain the research approach in section 3. The overall approach hereby follows the Design Science Research strategy, as presented in section 3.1. In section 3.2 we then illustrate the three major methods that were used in this research, namely literature review, conceptual modeling and prototyping of a software tool.

Part B comprises the eight publications that constitute the results of this research. In the first publication “**Why Product Service Systems Development is Special**”, we determine what differentiates PSS engineering from the development of traditional products or service. Publication 2 “**Analyse der Eignung domänenspezifischer Methoden der Anforderungsverfolgung für Produkt-Service-Systeme**“ analyzes existing requirements traceability approaches and evaluates whether they are applicable for PSS engineering. Subsequently in publication 3 “**Towards Cycle-Oriented Traceability for Engineering Change Management**” and publication 4 “**Traceability von Anforderungen und Tests in agilen Softwareentwicklungsprojekten**” we take a closer look at the issues of cross-domain traceability & change management as well as the different engineering cycles, especially due to agile methods. In publication 5 we then present a “**Concept for an Integration-Framework to enable the crossdisciplinary Development of Product-Service Systems**”. This framework forms the fundament for our model integration ontology and thus the integration of domain-specific artifacts into a comprehensive PSS model. On this basis, publication 6 “**Supporting the cross-disciplinary development of product-service systems through model transformations**” explains the mechanisms behind our model integration approach. Publication 7 “**Towards a Requirements Traceability Reference Model for Product Service Systems**” then summarizes our final model integration ontology, explaining the relevant artifacts of PSS engineering along with the semantic relationships between those. Finally, in publication 8 “**Introducing TRAILS: A Tool supporting Traceability, Integration and Visualisation of Engineering Knowledge for Product Service Systems Development**” we present our prototypical tool that implements our model integration and traceability approach.

As the results of our research are incorporated in the individual publications in part B, we continue with the discussion of our research in Part C. Here, we first present a summary of our findings (section 1.1). Afterward we discuss the implications for research (section 1.2) as well as the implications for practice (section 1.3). Each of those sections hereby explains the

implications from the perspectives: requirements traceability, product service systems and model-based systems engineering. As a next step, in section 1.4 we present the limitations of our research regarding the overall research approach, our model integration ontology as well as our prototypical software tool TRAILS. Finally in the discussion, we lay out a roadmap of potential future research (section 1.5). Here, we focus on three major directions: empirical evaluation, extension of the integration ontology and enhancement of the software tool, before we draw a conclusion in section 2.

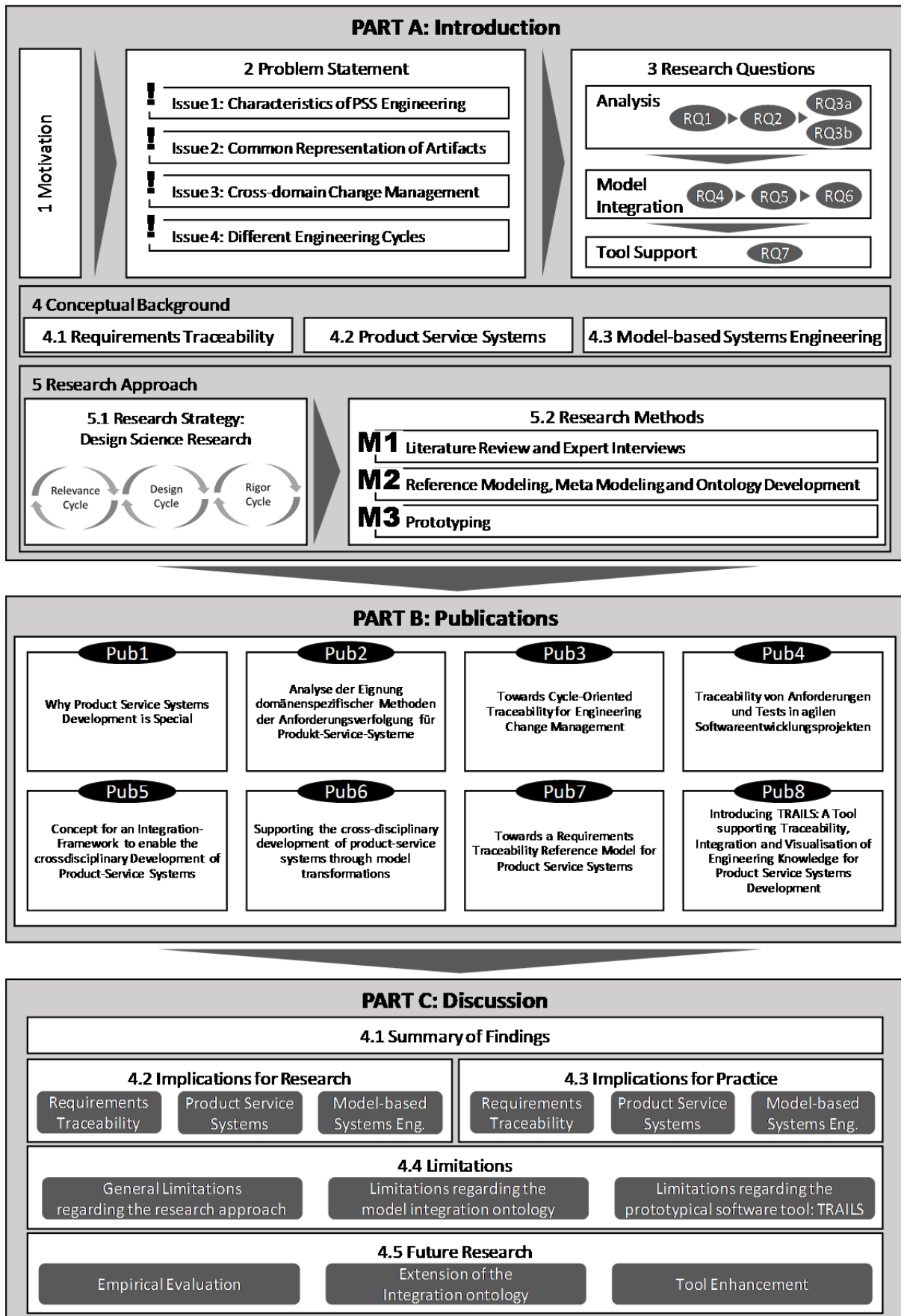


Figure 2: Structure of this dissertation
 Source: Own illustration

2 Conceptual Background

This chapter introduces the theoretical concepts that are fundamental to the topic of this thesis. First, requirements traceability is introduced as a pivotal task in requirements engineering for development projects in general and more specifically in the cross-domain development of PSS. As PSS is the application area that this thesis focuses on, the overall concept of a PSS business model, the different types of PSS that can be found in various industries and the special characteristics that shape their development are introduced. Finally, this chapter presents the concept of model-based systems engineering, a development paradigm which is central to the requirements traceability approach that is proposed in this thesis.

2.1 Requirements Traceability

Tracing the evolution and implementation of requirements is a crucial part of the requirements management process (Kotonya and Sommerville 1998). According to Gotel and Finkelstein (1994) *“requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its further detailing and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases).”* Since all solution approaches and tools presented in this thesis are ultimately targeted at ensuring traceability, this section presents an overview of the topic and general concepts. Furthermore, we illustrate the different dimensions of traceability. Finally, we introduce basic use cases for traceability information.

2.1.1 Imperative for traceability and cross-domain issues

The overall goal of requirements traceability is to document the life cycle of a requirement from its origin through all stages of analysis, detailing, refinement and adaption along the entire development process. Moreover, all dependencies and linkages between the requirements themselves, between requirements and solution components or between any other engineering artifacts need to be identified, documented and maintained (Ramesh and Jarke 2001). Tracing requirements involves the identification and documentation of semantic dependencies, so-called trace links. These trace links specify the relationships between all kinds of artifacts. By navigating along trace links, it is for example possible to comprehend why requirements had been specified in a certain way. Also developers can understand which solution components contribute to fulfilling a certain requirement or which test cases have been established in order to ensure that requirements are satisfied (Spanoudakis and Zisman 2005).

In the development of PSS, dependencies between artifacts that stem from different engineering domains tend to be the rule rather than the exception. Consequently, traceability needs to be guaranteed across domain borders (Berkovich et al. 2011b). In order to recognize which development and solution artifacts should be considered by post-specification traceability one has to examine the characteristics of the different engineering domains. For

example, the service design of a PSS often influences the architecture of software components or the appearance of hardware components that are needed for service delivery.

The development of hardware products does not solely embrace the design of the product itself but reaches out to production planning and beyond (Sharafi et al. 2010b). Among others, issues like manufacturability, logistics, disassembly or environmental considerations play a central role in the development process and should therefore be within the traceability scope. The list of criteria that have to be considered during product development can be easily expanded and is commonly known as “Design for X” criteria (Bauer and Paetzold 2006). Depending on which criteria play a role in a specific engineering process, requirements need to be traced to a wide variety of development artifacts (Dubois et al. 2010) like CAD-models, production plans, assembly line concepts or supply chain designs.

Since the great majority of traceability approaches that are described in literature were explicitly developed for software engineering purposes, those approaches already cover many of the factors that are important for this engineering domain (Hildenbrand 2008). However, in the case of PSS development it should also be noted that software often acts as the glue between hardware and service components (Berkovich et al. 2011c). Therefore, besides traceability links between requirements and software components also further relationships between software artifacts and artifacts in other engineering domains need to be considered in post-specification traceability.

Service engineering is probably the one domain that is most difficult to approach with regard to requirements traceability. By definition services are intangible, usually time-dependent and unique, but can have long-lasting effects. A service provider can only make the required factors and potentials available and define the process of service provision but services themselves can't be stored. Instead, they are created instantly in the interaction with the customer. Because services are produced and consumed at the same time a-priori trust in the service provider is an essential factor. Consequentially, this causes a latent problem for quality checks (Eversheim et al. 2006; Frese et al. 1998). For requirements traceability this means that services can only be designed with regard to fulfill the specified requirements but the actual fulfillment of each requirement cannot be checked until the service is provided. Additionally, the requirements for a service can be very volatile since they depend largely on the situation in which the service is provided. Therefore, a suitable traceability approach needs to define some kind of service proxies like service blueprints or other service process templates which can instead be traced during the phase of service provision.

2.1.2 Scope of requirements traceability

As shown in Figure 3, the scope of requirements traceability can be differentiated according to the direction in which requirements are traced. Forward traceability covers trace links to subsequent artifacts while backward traceability refers to links to preceding artifacts (Kotonya and Sommerville 1998; Jarke 1998). Accordingly, traceability is often differentiated according to the types of artifacts it links within the development process. This way, **pre-specification traceability** refers to linking initial stakeholder needs to the requirements specification while **post-specification traceability** follows the requirements from this stage

to various development artifacts and the final PSS components. Relationships between individual requirements are subject to **inter-requirements traceability** (Pohl and Rupp 2010).

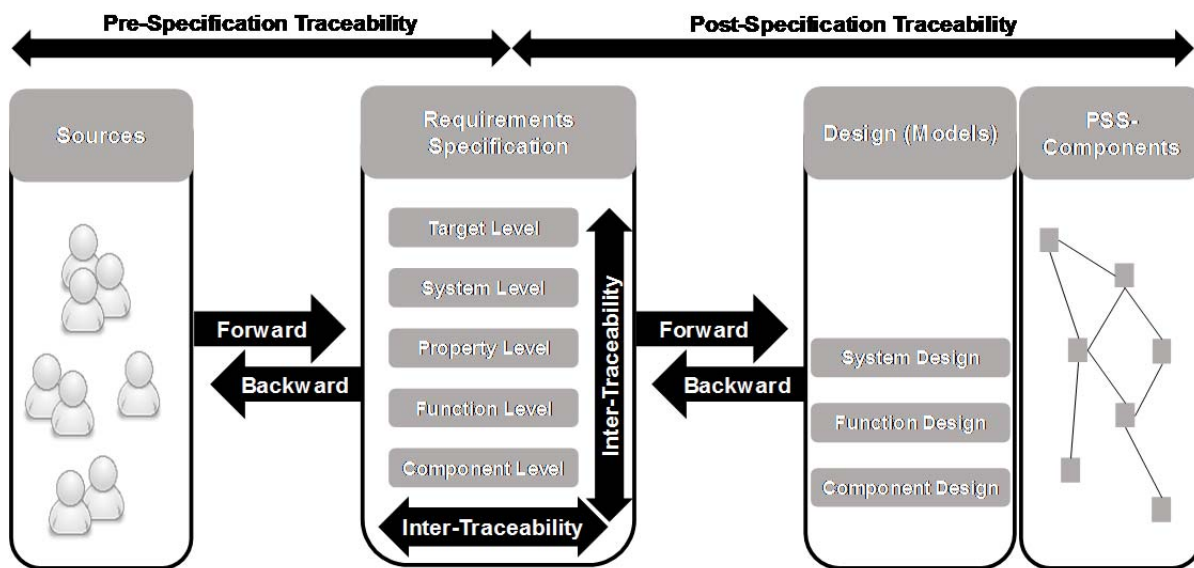


Figure 3: Scope of Traceability

Source: Adapted from Wolfenstetter et al. (2015a)

The development of a PSS starts with the appearance of a need in the market. Recognizing this need or demand respectively marks the beginning of each innovation process. In order to understand what the demand carrier really wants the demand has to be translated into requirements which summarize the interests that each stakeholder has in the desired solution. Being able to trace which source triggered the specification of a requirement is essential for the innovation process as it allows understanding why the requirement was specified in the first place. It is also reproducible, why a requirement was originally specified in a certain way and who to refer to if adaptations are needed (Gotel and Finkelstein 1995). Especially in the context of PSS development this becomes a real challenge since the spectrum of stakeholders that are involved in the process and who have largely varying intentions is very broad and heterogeneous. In fact, many studies (Ramesh and Jarke 2001; Pohl 1996a; Ramesh 1998) that are concerned with the industrial practice in requirements traceability argue that the so-called **pre-specification traceability** is the most apparent problem in practice and simultaneously the least understood. The reason for this is that early in the development process the needs of stakeholders are usually very vague and high-level and, if even, they are often documented in a free and unstructured manner. This makes it difficult to explicitly link them to a certain requirement within the specification.

By nature, the number of requirements rises with the increasing technical complexity of a product or the organizational complexity of a service. A complex product like a car or a complex service like surgery for example has to comply with a long list of legal regulations and standards, where each item on this list can lead to multiple requirements for each system component. Consequentially, the integration of complex products and services in a PSS

increases the volume and complexity of the requirements base tremendously. As a result, a complex and multi-hierarchical network of interrelated requirements evolves during the development of a PSS. This fact gives rise to the need for tracing the relationships between requirements on multiple levels of abstraction (we refer to it as **inter-requirements traceability**) in order to structure the requirements base (Lin et al. 2006).

A proven approach to accomplish requirements structuring for PSS is the artifact model which breaks the requirements down into five levels of abstraction. As illustrated in Figure 3 each level (target, system, property, function and component) covers multiple types of requirements. On the target level one defines generic requirements, not yet specifically linked to the development process of a PSS (i.e. overall targets of the service provider and the customer). On the system level neutral and initial requirements imposed on the PSS are considered. On the property level PSS engineers specify the results of service processes, the features of software components as well as the functions of hardware components. Requirements specifications on the function level are concerned with the way these service results, software features and hardware functions are realized, i.e. the behavioral structures of the components, their functionality and the requirements related to them. Finally, the domain-specific requirements on the component level describe the components in the language of the particular engineering domain (Berkovich et al. 2012).

The requirement artifacts influence each other, for example by limiting or concretizing each other. Therefore, several relationships between the artifacts can be distinguished (Pohl 2010; Geisberger 2005). By the differentiation into five levels, the artifact model supports the interdisciplinary elicitation and concretization of the requirements in accordance with the development process. An adequate traceability approach for PSS should therefore be able to trace relationships within such a hierarchically structured requirements base (Berkovich et al. 2011b).

Probably the subarea of requirements traceability that has gained the most attention in research as well as in practice is **post-specification traceability**, i.e. tracing how requirements are implemented. In a narrow sense, this means linking specified requirements to domain-specific solution components (e.g. software code) and test cases (Cleland-Huang and David Schmelzer 2003). However, especially in the case of PSS development, post-specification traceability should have a wider scope that among other things takes into account how product components are produced and maintained as well as how services are delivered.

2.1.3 Perspectives on traceability

Reviewing the literature, we found that various publications focus on different aspects of requirements traceability. As shown in Figure 4, these aspects can be broadly classified into three perspectives on requirements traceability, namely (1) the conceptual perspective, (2) the methodology perspective and the (3) process and management perspective.

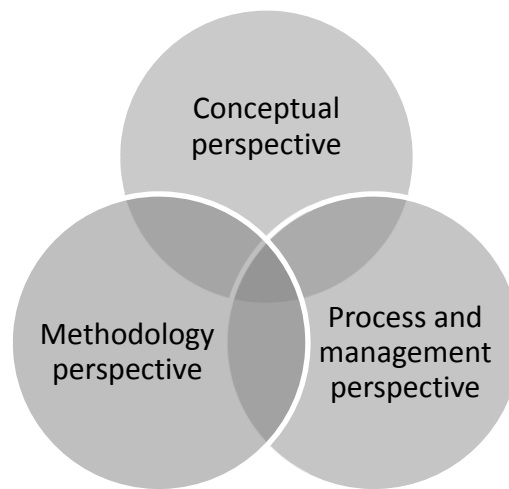


Figure 4: Research perspectives on requirements traceability

Source: Own illustration

The **conceptual perspective** mainly deals with the types of artifacts that are in the scope of requirements traceability as well as their structure. Publications that address traceability from this perspective often deal with the question, which development or solution artifacts need to be regarded in order to ensure traceability as well as the types of relationships existing between those artifacts. For example, Pohl (2010) or Spanoudakis and Zisman (2005) define several types of semantic trace links, namely dependency, generalization/refinement, evolution, satisfiability, overlap, conflict, rationalization and contribution. Moreover, Ramesh and Jarke (2001) as well as Hildenbrand (2008) present empirically backed reference models for traceability with a focus on software engineering. The potential of effectiveness and efficiency gains that can be realized in complex development processes due to proper traceability depends on various factors, e.g. number of requirements, size of the team composition, degree of requirements fluctuation, complexity of the product, heterogeneity of the customer structure. The stronger these factors are, the more important is an accurate and thorough management of traceability information (Schienmann 2001, p. 104), which comes in hand with having a conceptual traceability model that is align with the organization's needs.

A second category of publications approaches the traceability issue from a **methodology perspective**. Research from this perspective is concerned with methods and tools that can be applied in the area of requirements traceability. Here, the overall focus lies on “how” the traceability information should be captured, maintained and used (Ramesh 1998; Ramesh and Edwards 1993). In this context, methods for capturing and maintaining traceability information can be divided into three types, namely manual, semi-automatic and fully automatic (Spanoudakis and Zisman 2005; Aizenbud-Reshef et al. 2006). Additionally, research from this perspective is concerned with finding appropriate ways to visualize traceability information, suggesting various use cases for text-based, matrix-based or graph based visualization (Li and Maalej 2012).

The third view on requirements traceability is from a **process and management perspective**. Here, the focus is on the integration of requirements traceability activities into the innovation process as a whole. When implementing means to assure requirements traceability, factors

that influence the capturing, maintaining and utilizing of traceability information, such as organizational environment or development methodology need to be considered (Ramesh 1998). From a strategical point of view, the implementation of traceability requires finding the right level of information granularity, determining responsibilities within the organization and establishing routines within the development process for capturing and maintaining traceability information. Furthermore, engineers need to have a detailed understanding of potential use cases for traceability information in order for the company to profit from the laborious capturing and maintenance activities (Arkley and Riddle 2006; Ramesh et al. 1997).

The proposed categories often overlap with each other. For instance, requirements traceability processes depend on tools and methods that are applicable for each step in the process. This was also observed in our literature review, where we found that some publications address issues from more than one category.

In the research for this dissertation we focus on the conceptual perspective, since it is the most basic perspective of the three. By providing a conceptual approach to requirements traceability for the cross-domain engineering of PSS it is possible to focus on the methodology or the process and management perspective in future research.

2.1.4 Utilization of traceability information

Comprehensive requirements traceability within an engineering project is in our eyes implemented, if all traceability information that is needed for the use cases defined in the traceability strategy is captured and thoroughly documented. In reality however, capturing, documentation and maintenance of traceability information is often seen as an additional burden to the developer and thus neglected. Another obstacle in achieving comprehensive traceability is that the documentation of traceability information can take the form of very heterogeneous artifacts, such as software code, graph-based models, natural language documents and others. In practice, transparency suffers from these different complex contents (Egyed and Grünbacher 2005). For many organizations, the high costs of realizing comprehensive requirements traceability deter them from an investment in this area (Kannenbergh and Saiedian 2010). It is thus a primary task of research into requirements traceability to analyze and communicate the potential use cases of traceability information.

In a widely noticed study Mäder and Egyed (2012, p. 171) found that in development projects where traceability is provided, engineers generate better solutions in 60% of the tasks and were about 21% faster. The advantages of the requirements traceability are manifold. Especially systems validation, identification of inconsistencies, change management, impact analysis, knowledge management, stakeholder identification, artifacts reuse, accountability and project management are facilitated significantly.

Perhaps the most prominent use case for traceability information is **systems validation**. In quality management validation means testing if a system, product or software is suitable for its intentional use. Especially in industries where solutions are built to customer order, customers commonly demand proof that all their requirements have been met. If requirements engineering standards have been followed properly during system development, validation

can be accomplished by checking whether the specified requirements are satisfied. By doing so, it is possible to **demonstrate fulfillment of requirements** to the customer and at the same time enhance the overall quality of the system (Pohl 2010; Kirova et al. 2008; Ebert 2014). With adequate traceability information at hand, quality engineers can perform these checks by following the links from requirements to solution artifacts and further to test artifacts. In this context is also possible to assess which of the requirements are verified by which of the test cases and to assure that each requirement is checked in a corresponding test case. Furthermore, it becomes evident, which components of the solutions can be traced back to which requirements (Ramesh and Jarke 2001). This makes it possible to identify features that are incorporated in the system although they are not based on customer's demands. From the provider's perspective, such features need to be avoided, as their implementation usually causes a waste of resources and lowers future margins, since the customer is not willing to pay a premium for them (Ramesh 1998; Pohl 1996b). In this regard, traceability not only ensures implementation of requirements, it also helps to **identify unrequired system features** by checking if every feature contributes to satisfying the requirements. This development issue, also referred to as gold-plating often comes along with another related threat to systems development that is commonly labeled as requirements creep. Requirements creep means that additional requirements are added to the requirements specification by the developers after it has been considered complete (Robertson and Robertson 2006). Those requirements creep into the specification without being based on any customer, market or legislation demands. A systematic traceability approach can help to **avoid requirements creep** by capturing the source of and reason for each requirement.

Proper traceability information may further assist with the **identification of inconsistencies**. If, for example, trace links indicate several requirements that refer to the same solution component it needs to be ensured that these requirements do not contradict each other (Qusef 2013). Moreover, on the basis of traceability information it is possible to identify violations of physical laws, conversion errors between different units of measurement, mismatches between the different domain-specific engineering artifacts regarding the same solution component or even problems regarding the manufacturability of a physical component.

As mentioned in section 1.2.3 and section 1.2.4, the emergence of new requirements or the identification of inconsistencies within the requirements base as well as within or between other development or solution artifacts are a primal reason for changes and adaptations. The reasons for such changes are manifold and vary from an evolving competition in the target market to organizational changes or changes in the price structure of a PSS offering (Boehm 2000). This way, it is not surprising, that on average requirements have a monthly rate of change of about one to five percent, taking the project scope as a basis (Ebert 2012, p. 363). So, in an engineering project that spans over several years, a large fraction of the originally specified requirements has been changed at least once. To ensure that such changes are implemented in a controlled manner, active change management needs to be established (Doppler and Lauterburg 2008). Traceability information assists engineering **change management** (as well as organizational change management) in many different ways. To name one example, trace links show which artifacts are related to the artifact under

consideration which is subject to change and might therefore need to be changed as well (Kotonya and Sommerville 1998). Hence, comprehensive traceability information empowers change managers to act out the effects of change propagation within the network of interrelated development and solution artifacts and perform a change **impact analysis**. This fact is important since the efficiency of change management depends largely on the ability to evaluate how changing a requirement affects the system as a whole (Ebert 2008).

Not only can traceability information be used for the development of PSS, it can also be capitalized during service provision. In many PSS for example, **maintenance and support** services are an integral part of the business model. Comprehensive traceability information can support system maintenance by examining causes and effects of system failures (e.g. during FMEA), identifying affected system parts and predict the effort that is necessary to eliminate the failure (Pohl and Rupp 2010). This way, traceability information encourages product understanding and supports stakeholders in handling critical tasks in the development and maintenance of the product (Egyed and Grünbacher 2005; Maeder and Egyed 2012). Furthermore, traceability facilitates an understanding of how and why the system satisfies the needs of stakeholders by illustrating the connections between requirements and the system design. Support staff can also use traceability information to explain to the customer why the system behaves in a certain way and customer will comprehend the advantages of the current system design.

Due to the cross-disciplinary development of PSS, the various stakeholders involved in the innovation process are often familiar with only those parts of the system that they work on. Whenever these stakeholders leave the development project or the organization for whatever reason their knowledge that is not documented is usually lost and hard to rebuild. By diminishing this issue, traceability information can play a pivotal role in organizational **knowledge management**. Since traceability information explains the overall relations between all development and solution artifacts as well as their evolution, new team members can easily build up an understanding of the current state of development, facilitating the integration of new member into the development team (Ghazarian 2008). Especially organizations with a high staff turnover should rely on traceability of engineering information to guarantee that knowledge is preserved (Gotel and Finkelstein 1994).

If the relationships between artifacts and the people, departments and roles within and organization are captured, it is easily possible to **identify stakeholders** that are impacted by a change or that could be consulted if expert knowledge is needed for engineering tasks (Ramesh and Edwards 1993). This way, it is possible to bring together the right people in charge in order to implement necessary changes quickly or resolving mutual misunderstandings. This is especially important since from a realistic point of view, it is impossible to codify all tacit knowledge within the organization into documents.

Additionally, traceability encourages the **reuse of artifacts** as it is possible to identify artifacts that have already been developed for or used in a similar situation but in past engineering projects. Especially in software engineering the reuse of fragments of software code is a proven way to reduce development time (Krueger 1992). This strategy can also be

applied in the other domains of PSS engineering, namely service engineering or mechanical engineering. For example, it is possible to reuse generic business process specifications, such as payment processes, in other contexts or to adapt hardware products for other purposes than the originally intended (c.f. part C, section 1.5). By comparing requirements in past and new development projects one can identify traceability relationships and development artifacts that can be adapted and reused, thus saving resources (Pohl and Rupp 2010). However, this is not only true for solution artifacts that fulfill similar requirements specifications but also for requirements that have been specified according to similar customer needs.

For managers of PSS engineering projects traceability information is a valuable knowledge base for various **project management** tasks. Particularly, quantitative analyses of traceability information regarding the status of the development project or the resources spent for development are of interest for these tasks (Ramesh and Edwards 1993). For example, by mapping development effort from system features to requirements it can be reconstructed how much effort was needed to fulfill a specific requirement. Furthermore, based on traceability information, project managers can assess to which degree development is completed, forecast whether milestones will be met, determine the performance of the development team or establishing a pricing structure of the future business model based on development efforts for certain features. In summary, requirements traceability can play a key role in monitoring project progress, allocating resources and controlling costs (van Lamsweerde 2007).

While the advantage that the utilization of traceability information offers for the use cases just described makes the provision of traceability and its proper management recommendable for every single engineering domain, the availability of traceability information unfolds its entire potential in the context of PSS engineering, because in this context cross-domain engineering and customer orientation are dominating factors.

2.2 Product Service Systems

In various industries and markets companies feel the pressure of increasing competition in combination with more and more complex customer requirements (Boehm and Thomas 2013; Leimeister 2012). Since even developing countries rapidly catch up in terms of product quality and technology, many companies that are based in countries with a higher cost of labor have realized competitive advantages elsewhere than in technology or quality leadership. Today, in developed markets, the prime source of competitive advantage for local companies is the proximity to the customer, which comes in hand with understanding the customer's problems and being able to deliver individual solutions to these problems. Moreover, customers increasingly demand individually customized solutions to their problems (Leimeister and Glauner 2008; Sawhney 2006) rather than separate standardized products or services that were developed for a mass market (Knackstedt et al. 2008; Nordin and Kowalkowski 2010).

2.2.1 Terminology

Many companies, product manufacturers as well as service providers, are changing their traditional business model to become so-called solution providers (Davies et al. 2006a; Galbraith 2002). The solutions they offer are integrated bundles of software, hardware and services that aim at resolving a specific customer need (Sawhney 2006; Tuli et al. 2007). Scientific literature refers to such bundles mainly as Product Service Systems (PSS) (Baines et al. 2007; Boehm and Thomas 2013) as they integrate physical and/or software products with service components into one offering whose single components are not distinguishable as such for the customer.

However, the terminology in the context of integrated product and service bundles varies from one author to another. Consequentially different terms are used synonymously (Knackstedt et al. 2008; Velamuri et al. 2011). For example, we often find terms like hybrid product (Leimeister and Glauner 2008), solution (Johansson et al. 2003), customer solution (Foote et al. 2001) or covalent product (Weber et al. 2002) when referring to PSS. In order to clarify, what we understand by the term PSS, we present a selection of definitions and derive what we understand when referring to product service systems in this dissertation.

Table 2: Definitions of the term Product Service System

Source	Definition
(Goedkoop et al. 1999)	“A Product Service system (PS system, or product service combination) is a marketable set of products and services, jointly capable of fulfilling a client’s need. [...] PS system knowledge enables companies to find strategic options for business growth, renewal, innovation and diversification.”
(Mont 2002)	“[A] system of products, services, supporting networks and infrastructure that is designed to be: competitive, satisfy customer needs and have a lower environmental impact than traditional business models.”
(Manzini and Vezzoli 2003).	“[A]n innovation strategy, shifting the business focus from designing (and selling) physical products only, to designing (and selling) a system of products and services which are jointly capable of fulfilling specific client demands.”
(Tukker 2004)	“A product-service system (PSS) can be defined as consisting of tangible products and intangible services designed and combined so that they jointly are capable of fulfilling specific customer needs.”
(Baines et al. 2007)	“A PSS is an integrated product and service offering that delivers value in use to the customer.”
(Böhm and Krcmar 2007)	Translated from German: Hybrid products are combinations of products and services that are offered as integrated bundles to a market. The goal of combining products and services is to offer

	solutions to specific customer problems.
(Tan et al. 2007)	“PSS is a shift in business strategy from a product-oriented to a service-oriented focus, where instead of the product itself, the activity, its utility and performance associated with the use of the product are considered to be of more value to the customer.”

Following the definitions presented, a PSS is a combination of products and services that are tailored towards solving a specific customer problem. In this regard, the tight integration of its components is a key characteristic (Tuli et al. 2007; Burianek et al. 2009). This means that the individual components of a PSS cannot be distinguished easily (Leimeister and Glauner 2008). In the context of this work we thus define the term PSS in the following way:

“A PSS comprises a technical product with integrated services that provides a complete solution to users. Further, the product itself could be made up of mechanical, electronic and/ or software components.”

2.2.2 Types of PSS

PSS can be found in a variety of industries, such as mechanical and industrial engineering, transportation, public infrastructure or information technology (Becker and Krcmar 2008; Tuli et al. 2007). In the IT sector for example, so-called “software as a service” is an area that has been subject to tremendous growth over the past years. In this business model, the functionality of standard software is offered over the internet on a rental basis and can be accessed via a web browser while the application itself is hosted in a remote data center (Böhmman and Krcmar 2007; Berkovich et al. 2010a). However, also complete solutions at the customer’s site are quite common. In such business models the provider is responsible for setup, operation, maintenance and updates of a software solution as well as the necessary hardware and end user training and support (Floerecke et al. 2012).

According to Tukker (2004) one can differentiate between three generic types of PSS, namely **(1) product-oriented, (2) use-oriented and (3) result-oriented PSS.**

Product-oriented PSS augment a traditionally sold product by adding product-related services to customers. In this type of business model, the product is owned by the customer, while the service is provided by the firms. In such business models during the operational phase of the product the PSS provider offers specific product-related services such as maintenance or repair. When reaching the end-of-use or end-of-life phase, take-back and reprocessing services are offered. In **use-oriented PSS**, customers usually pay for the usage of products, while the ownership of the product remains with the PSS provider. Consequentially the provider takes over the responsibility of providing a certain product with a promised functionality in operable condition (Tukker 2004; Bartolomeo et al. 2003). **Result-oriented PSS** concentrate on delivering a certain result to the customer by keeping the full responsibility at the provider’s side (Tukker 2004). This last type of PSS business model is totally independent of any product instance. The provider just obligates himself to deliver a certain result.

In opposition to traditional businesses in which value creation is commonly seen from a transactional perspective the creation of value in PSS business models is a relational process in which the provider and the customer interact (Tuli et al. 2007). This way, the customer is no longer just a mere receiver of the value created, but acts as a value creation partner (Becker et al. 2009; Schmitz 2008). This way, PSS business models tend towards more intensive and long-lasting customer relationships than traditional businesses. In such relationships the provider usually takes over responsibilities along the entire lifecycle, from requirements elicitation to the stage of service provision and finally to the replacement and disposal of PSS components (Baines et al. 2007; Herzfeldt et al. 2012).

2.2.3 Development of PSS

Comparing the different domains that are commonly involved in PSS development, namely mechanical engineering, software engineering and service engineering, we find that in each domain solution artifacts can be described in three dimensions.

In mechanical engineering we usually distinguish between the function, the behavior and the structure of a system (Qian and Gero 1996). Software engineers on the other hand often view system architecture from a feature (or function) dimension, a workflow (or control) dimension and a data dimension (Scheer 1992). Also the domain of service engineering mostly resorts to a tri-partition of service dimensions⁷. First, the service results describe the desired outcome of a service. The process dimension describes which activities need to be performed in order to generate this outcome and the resource dimension defines what is required in order to perform the activities that generate the service result. These resources can for example be human factors, information and knowledge as well as hardware or software artifacts (Bullinger and Schreiner 2006). This means that the hardware and software components of a PSS constitute service resources. As such they are primarily means to an end in order to provide the service to the customer. **Fehler! Verweisquelle konnte nicht gefunden werden.** clarifies the aforementioned similarities between hardware, software and service engineering and illustrates the development of PSS that is generically speaking composed of (requirements) analysis (dashed arrows) and component integration (solid arrows).

⁷ Some authors add the market dimension as a fourth dimension to describe services.

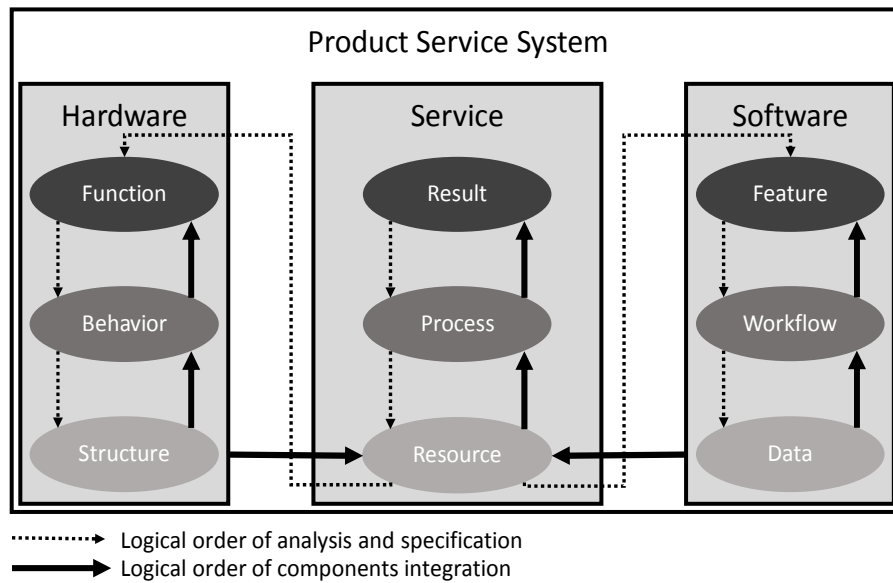


Figure 5: Development of Product Service Systems

Source: Own illustration

One fundamental characteristic of PSS that becomes evident during their development is the special focus on satisfying customer needs (Aurich et al. 2006). The starting point of a PSS development project is usually the identification of the customer's (unconscious) problems and desires. During requirements analysis, the PSS provider defines that essential service results that are needed in order to fulfill the basic customer requirements. As a next step it is necessary to determine the processes needed in order to achieve the service results and break them down into single activities. For each activity one can now specify the stakeholders involved in performing the activity and evaluate which other service resources are required. Apart from human actors these resources are mainly information (which can be the result of another service or s delivered by software) or physical items, i.e. hardware.

2.2.4 Need for Integration in PSS engineering

Customer-oriented business models, such as PSS aim at providing a solution to the customer's problems rather than the means to solve the problems himself. Thus they are usually based on various solution components, such as hardware, software and human labor (in the PSS development process represented in terms of service guidelines and instructions, e.g. service blueprints or process models). To deliver the desired outcome all these components need to seamlessly work together (Cook et al. 2006; Tukker and Tischner 2006). The need for integration is therefore a defining factor in PSS engineering.

When it comes to functional harmonization the various solution components, experts speak of technical integration. Here, the goal is to generate a value-added to the customer when comparing the integrated PSS to the sum of its components (Johansson et al. 2003). In general, PSS derive their reason to exist from the fact that they are more than just products and services sold as a bundle (Reichwald et al. 2009). In this regard, Burianek et al. (Burianek et al. 2009) speak of the „1+1=3“-effect.

In addition to the technical integration of solution components, PSS require organizational integration. This means that all internal business processes of the provider that are related to the PSS need to be aligned (Beverungen et al. 2009). Furthermore, the PSS often needs to be integrated with existing business processes or the IT-landscape of the customer. This is often the case when it is not economically reasonable or desired by the customer to replace all existing processes or IT-systems with the solutions offered by the PSS. In this case the PSS provider often needs to take care of the continuation of such legacy systems (Böhm and Krcmar 2007). The need for integration that is inherent to any PSS also presents itself in terms of marketing. PSS providers need to bundle formerly independent products and service together in a way that delivers the highest value in use to the customer (Sawhney 2006; Sturm and Bading 2008).

Since in most of the cases product-oriented PSS business models are basically products with built around services, components can be developed independently of each other. However, the situation with use- or result-oriented PSS is different. Deep integration between hardware, software and services requires high levels of collaboration between the different engineering domains continuously during all phases of the PSS life cycle (Johansson et al. 2003). Usually, each of the stages of delivery, after-sales, and reprocessing services also different stakeholders within the value creation network are involved (Bonnemeier et al. 2007).

2.2.5 Advantages of PSS business models

From the provider's perspective the introduction of a PSS usually allows firms to generate **continuous revenues** rather than one time sales and realize other benefits (Mont 2000) such as entering new markets or increasing customer satisfaction and loyalty (Tukker 2004) by entering long-term in depth customer relationships. Due to this, providers can profit from **faster feedback loops** with the customer and as a result of the closer contact to the customer there is also the possibility of **add-on sales** that increase the provider's business volume as well as margins.

Seen from the perspective of a customer, PSS offerings promise an **easy scale up** of the service provided. As with, for example, cloud computing services or even automated production systems that are obtained as a service, customer often just have to extend their service contract and the provider will care for the rest. As a consequence, the **total cost of ownership (TCO)** is also more transparent to the customer, since the cost of "one unit of service" is broken down in the service contract with the provider. Another advantage for the customer is the **reduction of risk**, because most of the risk regarding for example underused capacities or fast recovery after system failures is born by the PSS provider.

Besides economic advantages for both the customer as well as the PSS provider, PSS also feature advantages for society in general since they tend to have a smaller adverse **impact on the environment** than traditional business models (Baines et al. 2007; Beuren et al. 2013). For example, Mont (2002) argues that due to the special usage patterns of PSS fewer units of a product need to be manufactured and maintained. Moreover, PSS business models often imply so-called closed-loop material flows, fostering refurbishment rather than disposal.

Furthermore, due to the adaptation to special customer requirements PSS usually feature a more **economical factor input** (Tukker and Tischner 2006).

2.3 Model-based Systems Engineering

In general, Systems Engineering can be defined as *“the intellectual, academic and professional discipline the principal concern of which is the responsibility to ensure that all requirements for a bioware/hardware/software system are satisfied throughout the life cycle of the system”* (Wymore 1993, p. 5).

As especially technical systems are getting smarter, increasingly miniaturized and interconnected, their architecture and design becomes ever more complex. This development forces engineers to describe them in a way that facilitates the handling of their inherent complexity. One well suited and at the same time popular way of dealing with complexity is specifying the system from multiple viewpoints using adequate modeling approaches.

Following Stachowiak (1973) a model is a representation of an original that has three basic characteristics. First, models are always models of something, i.e. representations of originals, which can be models themselves (mapping characteristic). Second, models don't represent all features of an original but only such that are selected as relevant by the modeler (reduction characteristic). And third, a model is not a mere representation of an original but has a certain purpose to a certain subject within a certain timeframe (pragmatic characteristic).

By anchoring the usage of models as the focal point of all engineering activities the model-based engineering approach aims at solving engineering issues through abstraction and reuse. At the same time, complexity is manageable and the formal or at least semi-formal specifications avoid ambiguity and make the system structure easier to comprehend. Model-based systems engineering (MBSE) can be summarized as a paradigm for development projects in which models are seen as the central development artifacts. This means, that models are not seen as a tool for mere documentation purposes. Instead, models deliver additional benefits for the engineering process, e.g. through semi-automated generation of deliverables such as software code also referred to as model-driven approaches (France and Rumpe 2007). Overall, models can play a vital role along the entire lifecycle of a system ranging from concept development to deployment, operation, maintenance and even update, refurbishment or disposal (Feiler and Gluch 2012).

For PSS providers a model-based approach to engineering comprises enormous benefits and yet obstacles at the same time. Probably the most common challenges are related to communication and integration of engineering information as well as the management of complexity. By its very definition, PSS engineering involves several domains, each acquainted with its own modeling languages and software tools.

2.3.1 Traditional engineering vs. model based engineering

When characterizing the model-based approach in systems engineering we find that it rather focuses on abstraction of a specific problem and the stakeholder's requirements rather than generating a quick fix for the current problem at hand and altering this solution draft until it

sufficiently satisfies all requirements as we find it in traditional engineering approaches. By doing so, model-based systems engineering elaborates more on the problem domain in order to capture the problem from various perspectives. This process involves different types of modeling languages, each decomposing the problem setting according to a specific dimension. While three dimensional CAD models show the spatial structure of physical components, e.g. SysML block diagrams show a more logical decomposition of the systems architecture, while at the same time abstracting from the physical dimensions.

In current industry practice model-based systems engineering is mostly synonymous to the use of graph-based modeling languages such as UML, SysML, BPMN or ERMs. By illustrating the system under consideration as a graph consisting of interrelated elements, these models support engineers in understanding the overall structure of the system. It is often argued that the major advantages of model-based systems engineering are the improvements in the design quality due to a more complete and formal form of specification that is beneficiary to the likelihood of inconsistencies and design errors getting uncovered. This is especially true, since some test artifacts can be generated automatically, corresponding test and simulations can be executed continuously and their results can be evaluated instantly. Furthermore, in many cases the development time can be reduced significantly as already existent designs and models can be adapted and reused (Bergenthal 2011). Another advantage of using generally understandable models in the engineering process is that it improves the communication and knowledge sharing between the various, often heterogeneous groups of stakeholders. Due to the more intuitive way of communicating the structure and behavior of a system, the semi-formal model representation is easier accessible, also for someone from outside the subject area.

However, the model-based approach has several disadvantages when compared to traditional development methodologies. First to mention, there is redundancy. The artifacts that are created during the model-based engineering process represent the system from multiple viewpoints. Therefore, engineering information about a certain part of the system, e.g. its dimensions, exists in duplicated form, spread over multiple models. As a consequence, there is a duplication of work because the system models are to a large part created manually (Hailpern and Tarr 2006). Second, there is the problem of ensuring consistency between the models. This means that if there is a change in one of the models an impact analysis must be performed and each model affected by the change must be updated. Otherwise, different engineers might work on inconsistent information resulting in problems when it comes to the integration of components or even malfunctions of the final system.

2.3.2 Modeling languages in PSS engineering

As argued before, modeling a complex product service system in all its details is likely to turn out as an extensive task involving various engineering domains, layers of abstraction and perspectives on the system. Therefore, instead of representing all aspects of the system as a whole in one large model, the model-based engineering approach rather resorts to constructing a wider landscape of sub-models with limited focus, each of which just captures a specific part of the system from a certain perspective. For this purpose, in each engineering

domain, in each abstraction layer and for each perspective on the system specialized modeling languages and technologies have established themselves (Becker et al. 2010). Furthermore, each domain-specific model only contains those aspects of the overall systems, which are relevant for the respective engineering domain. Table 3 summarizes some of the most popular modeling languages in PSS engineering (See also: Durugbo 2013) and classifies them into four categories according to the perspective on the system they represent.

Table 3: Modeling Languages in PSS engineering

	Modeling languages or diagram types
System Environment	e3 Value, Molecular Models (Shostack 1977), Business Model Canvas (Osterwalder and Pigneur 2010)
System Result	Requirements Diagram (SysML), I*, IDEF, SADT, Use Case Diagram (UML, SysML), Function Tree
System Behavior	BPMN, EPC, Activity Diagram (UML, SysML), Sequence Diagram (UML, SysML), Service Blueprint (Shostack 1982), Petri Nets
System Structure	Block Diagram (SysML), Class Diagram (UML), Design Structure Matrix,

While it can be argued, that with UML or SysML there are already two candidates for a common modeling language, despite their growing popularity in industry, both have downsides when it comes to model integration. A natural phenomenon with modeling languages is that higher expressive power increases the inherent complexity of a language. Since both languages lie their focus on being able to model all kinds of systems in as much detail as possible, they are becoming increasingly complex as new features are getting added. In fact, the expressive power of these modeling languages has become so extensive that one can observe increasing endeavors to customize e.g. SysML for specific purposes (e.g. modeling mechatronic systems Barbieri et al. 2014).

2.3.3 Conceptual Modeling: Reference Models, Meta Models and Ontologies

Traceability requires a comprehensive conceptual model that defines the artifacts (i.e. the general concepts) of each engineering domain and their relationships with each other, desirably throughout the entire life cycle of a PSS. Conceptual modeling formally describes various aspects of real world entities, thus supporting humans as well as machines to understand or interpret the purpose, structure and behavior of these entities and communicate about them. As such, conceptual modeling plays an important part in the process of model integration or model mapping, respectively. When it comes to conceptual modeling, the terms reference model, meta-model and, lately, ontology are often used (and they are used synonymously). In this context, we subsequently explain these terms from the perspective of model integration and traceability for PSS engineering.

First, reference models can be seen as best-practice examples for common issues in a certain domain that have been generalized in order to be applicable universally. Reference models contain domain knowledge so that companies can use those models in order to create solutions for specific issues (Becker et al. 2010). As such, they have a recommending

character in innovation processes facilitating the transfer of knowledge into companies (Schütte 2013). Especially in engineering, reference models are needed at various levels of abstraction and in various domains, whereby each describes generic solution concepts in a generalized terminology. From a systems engineering perspective, reference model can best be created through iterative refinement in order to advance to more detailed levels of understanding (Wieringa 2008). In pursuit of the target of this thesis, the integration of knowledge from the different engineering domains is necessary. For this purpose, it is advisable to advance from existing reference models within the engineering domains and join those in order to come up with an integrated reference model (Rosemann 2013).

The primary goal of reference models is to provide generalized domain knowledge. In contrast to this, modeling languages (such as the ones discussed in the chapter before) together with the meta models that define those languages formally specify the structure of conceptual models (Becker et al. 2010). Meta models can thus be seen as orthogonal to reference models. An example of a meta model is the Meta Object Facility (MOF) which defines the elements used in UML diagrams. And again, UML itself consists of meta models that define for example what a UML class diagram that models an actual software system is composed of. In this sense, there can be multiple levels of meta modeling, each defining the concepts of the level below in an abstracted manner. In the perspective of this thesis, a meta model defines the structure of model while a reference model summarizes the domain knowledge needed to develop meaningful conceptual models.

Looking at PSS engineering specifically, we see a lack of modeling approaches that are capable of delivering a comprehensive picture of all aspects that are relevant to the PSS life cycle. For this purpose, a demand-driven integration of existing modeling languages as well as the integration of reference models from the different engineering domains seems to be necessary (Becker et al. 2010). This integration can be done in several ways, one of which is based on identifying common concepts that are specified by different domain-specific modeling approaches and the defining a unified representation of those.

Of course, it is in the context not enough to simply define a common vocabulary, i.e. a list with defined concepts that are “allowed” in the unified representation that all domain-specific modeling approaches can be translated to. In addition to just defining the “words” of this language, one would also have to define its internal structure, i.e. the semantic relationships between the “words” that are used. Only if both things are available, the definition of concepts as well as the definition of the semantic relationships between those concepts, the unified representation has an internal logic that can be understood by machines (Noy 2004).

A well suited approach for this purpose that is for good reasons extremely popular in the area of artificial intelligence but has also recently become common on the world wide web, is ontologies. In practice, ontologies are among other things used for categorizing websites or products in online shopping applications (Noy et al. 2001). The term “ontology” can be defined as “a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological

commitment. An ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models” (Guarino 1998).

From a philosophical perspective, the term “ontology” is used in a much broader sense, referring to a system of categories that describe the world from a certain point of view. In the context of artificial intelligence, an ontology is an engineering artifact that defines a specific vocabulary along with assumptions regarding the meaning of the words, i.e. the semantic relationships between those words. Therefore, ontologies are a valuable tool when dealing with the semantic heterogeneity in structured data (Noy 2004). In this context, every natural language, modeling language or every other type of specification technique imaginable can be viewed as an ontology. So, in order to translate from one ontology to the other, i.e. transform one type of model artifact into another type of model artifact, we require a dictionary that contains the transformation rules between the languages (Guarino 1998).

A fundamental goal of using ontologies is to create artifacts that are compatible to different applications. If these ontologies refer to the same top-level ontology, integration between application is relatively simple (Noy 2004). Perhaps the most interesting aspect about using ontologies for making sense of structured data is the perspective of ontology-driven information systems (Guarino 1998). Today, in the semantic web, many researchers agree that the challenge of semantic integration is one of the hardest and that ontologies are the most promising solution approach for this (Noy 2004).

2.3.4 Model Integration and Model Transformation

With models playing a pivotal role in all areas of engineering it is of utmost importance that the various sub-models that together form a comprehensive specification of the PSS are consistent. Consistency in the context comprises syntactic consistency as well as semantic consistency. While syntactic consistency means that a certain model must conform to its meta model, semantic consistency refers to the claim that various models that refer to the same ontological entity may not contradict each other (Lucas et al. 2009).

In general, consistency among the various models of the PSS engineering process can be reached by either regularly checking the models by hand or performing automatic and continuous consistency checking. However, in order to do the latter, models need to be integrated. Correspondingly, model integration refers to merging several source models into one comprehensive target model. By doing so, it is possible to document and display the various inherent relationships that are hidden within the different sub-models. By combining the different perspectives on the PSS into one common structure engineers can better understand the various dependencies within the system as a whole (Patel and Nagl 2010).

This means that models from different engineering domains that are written in different modeling languages using different data formats need to be transformed into one common form of representation (Tratt 2005). These so-called model transformations usually map element of a source (meta) model onto corresponding elements of a target (meta) model in accordance with certain transformation characteristics and requirements. In order to do so, one needs to analyze the different types of entities a modeling language differentiates between

as well as the types of semantic relationships these entities may engage in. Furthermore, one needs to identify all possible attributes of entities and relationships. On this basis it is then possible to specify the transformation rules between one modeling language and another.

In the context of PSS engineering, model integration and transformation comes with a number of requirements. First, the landscape of models that are used in PSS engineering is rather heterogeneous. As already mentioned, software engineers, mechanical engineers or service engineers all use various domain-specific modeling languages for various purposes. However, even one and the same modeling language (e.g. UML) can be represented using different data formats, for example XMI or proprietary formats. Furthermore, models can differ regarding the perspective they take on the object that is modeled (e.g. behavioral vs. structural perspective) or the level of abstraction at which they capture the PSS. A comprehensive approach for model integration in PSS engineering should thus support a larger variety of models or in the best case operate independently from the concrete model instances.

Another major requirement in this context is that the model transformation tool produces dependable results on a reliable basis. In the best case, a smart algorithm is capable of performing fully automated model transformations and consequentially reduce the manual efforts for the development engineers to zero. As a matter of principle, automated model mapping is in general less error prone than manual efforts as long as the transformation process itself is deterministic and does not rely on a semantic interpretation of the source model. In this sense, an automated model transformation process needs to be verifiable. However, sometimes the transformation also requires a context sensitive interpretation of the source model in order to produce the correct result. Therefore, a suitable transformation algorithm needs to be flexible enough to produce acceptable transformation results even when the source artifacts are not fully conformant to the meta models of the languages they are specified in. Consequentially, another requirement for model transformation in PSS engineering is that the resulting integrated PSS model preserves all attributes of the source models involved. If modifications to original attributes of an entity within a source model are necessary (e.g. unit conversions to homogenize the use of metrics in the integrated model), the original attribute needs to be reproducible.

2.3.5 Using Semantic Web Technologies in Engineering

The term “semantic technologies” is mainly used to summarize algorithms, data structures and software solutions that bring structure and meaning to information. The basic idea behind these semantic technologies is to specify information objects and their relationships in a way that makes them interpretable by computers (Davies et al. 2006b). In order to do so, information objects are annotated using meta information to explain the meaning of the original information captured in the respective information object. As an example of how semantic technologies impact the way information is being processed and interpreted we can refer to large parts of the internet as we know it today. Currently, the vast majority of information within the internet is stored as natural language text that needs to be manually interpreted by the human user. By adding context and structure to the data it can be interpreted automatically allowing more accurate search results and logical reasoning

performed by machines (Hitzler et al. 2007). Another issue with the internet today is that the data is rather heterogeneous and unstructured. By adding meta information to information object, semantic technologies are able to homogenize and structure which allows for a better integration of heterogeneous data sources. In short, previously unrelated information objects can be linked within a semantic network.

As argued before, the integration of information from different (domain-specific) sources, the provision of traceability and the contextual analysis of engineering data are also a challenge that is common in systems engineering. Similar to the vision of a semantic internet, a “semantic engineering web” containing all knowledge about the system under development would open up tremendous possibilities regarding the automated analysis of engineering artifacts and the anticipation of change effects and other dynamic influences. For this vision to become reality, two prerequisites need to be fulfilled: data structuring and data integration. While the issue of representing engineering information in a structured manner is already being addressed through the model-based systems engineering approach, integration of the various data formats and model artifacts used by the different engineering domains still remains an open issue. Here, the goal is to transform heterogeneous the model artifacts into an integrated representation in order to identify and analyze the inherent semantic relationships between model entities.

To support the development of the semantic web, the world wide web consortium proposes a set of specific technologies that build upon each other and form the so-called semantic web technology stack. As shown in Figure 6: Semantic Web Technology Stack, this technology stack is composed of several layers, each of them using functions that are offered by the layer below.

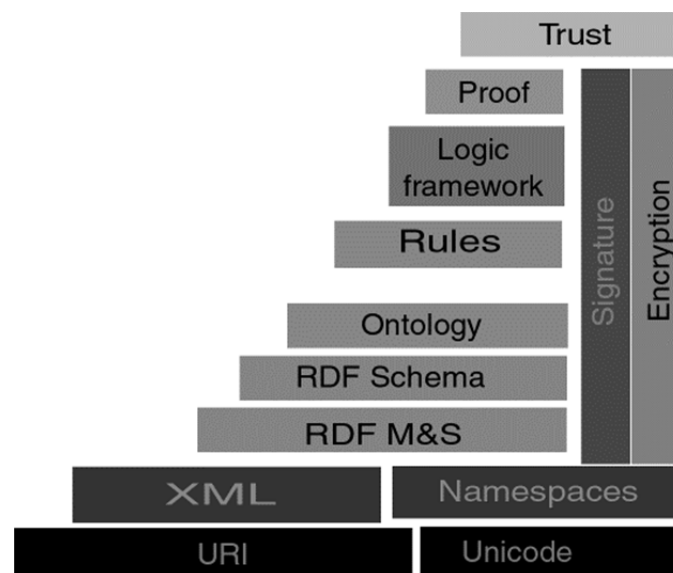


Figure 6: Semantic Web Technology Stack
 Source: adapted from (Hazaël-Massieux 2003)

On the fundamental level, the Unicode standard defines a repertoire of characters to allow consistent coding and representation of text. Furthermore, the semantic web technology stack defines on this level Uniform Resource Identifiers (URI) that are used to uniquely identify

any resource. One layer above, the extensible markup language (XML) specifies a set of rules for encoding documents defining a syntax for the semantic web. In this sense, XML serves as a basic format for various higher level semantic web technologies.

Probably one of the most important technologies in the semantic web field is the Resource Description Framework (RDF). RDF is mainly used for interchanging data in a way that preserves its meaning. The general idea behind RDF is to make statements about resources. From a linguistic perspective, each RDF term is a triple composed of subject, predicate and object. While subject and predicate always take the form of an URI the object can also be a rare data value, a so-called literal. For the serialization of RDF models different notation formats can be used, the most popular being RDF/XML and the compact and more human-friendly Turtle format. Each RDF model intrinsically represents an annotated directed multi-graph with subjects and objects being nodes and predicates being edges that reflect the semantic relationship between two nodes (Hitzler et al. 2007).

On the technology stack of the semantic web, we find one level above RDF some technologies that can be used to logically interpret the semantic RDF data and define complex rules and a vocabulary of concepts, so-called ontologies. In general philosophy an ontology refers to the science of existing things (Chandrasekaran et al. 1999). In the information science domain, ontologies are used to describe all types of entities, their properties as well as their relationships with each other. This way, ontologies help to limit complexity and organize the information within a specific domain. As such, ontologies are being used to represent the domain knowledge in a self-consistent manner in order to create a consistent knowledge base. As means for defining ontologies in the semantic web context the world wide web consortium has established the Web Ontology Language (OWL).

3 Research Approach

As explained in the last chapter, our research combines three different fields of research, namely requirements traceability, product service systems and model-based systems engineering. Because of this, we looked for a research approach that allowed us to explore these fields of research, gradually improving our solutions in several iterations. Eventually, we found an adequate research strategy by following the recommendations of Design Science Research as presented by Hevner et al. (2004, 2007). In this process, extensive literature reviews were conducted in the fields of requirements traceability, PSS development and model-based systems engineering in order to capture the scientific theory on the subject matter. Throughout this process we studied literature from various engineering domains, such as service engineering, software engineering and mechanical engineering. Moreover, we also studied several cases from different industries in order to consider the practitioners' perspective. Furthermore, the design cycle involved conceptual modeling (especially reference modeling) and software tool prototyping. The fundamentals of these methodologies are explained in the subsequent sections.

3.1 Research Strategy

Design Science Research (DSR) is a methodology that focuses on the development and evaluation of IT artifacts that are targeted at solving a specific management problem (Hevner et al. 2004). This way, Design Science Research is rather concerned with identifying design rules, principles and methodologies which manifest themselves as artifacts, rather than finding descriptions or explanations for certain phenomena like other branches of science do (Wieringa 2008). The artifacts can be for example algorithms, interfaces, processes, models, languages or software tools. In the process of developing such IT artifacts the researcher is required to build up domain-specific knowledge in order to be able to design a solution for the management issue (van Aken 2005). Through an iterative process in which the artifact is enhanced and evaluated continuously, the researcher is getting a more and more detailed picture of the issue studied and can therefore evaluate the solution approach and evolve the artifact until it solves the issue satisfactorily (Hevner et al. 2004). In its core, the process of Design Science Research builds upon three cycles: **(1) the relevance cycle, (2) the design cycle and (3) the rigor cycle** (cf. Figure 7).

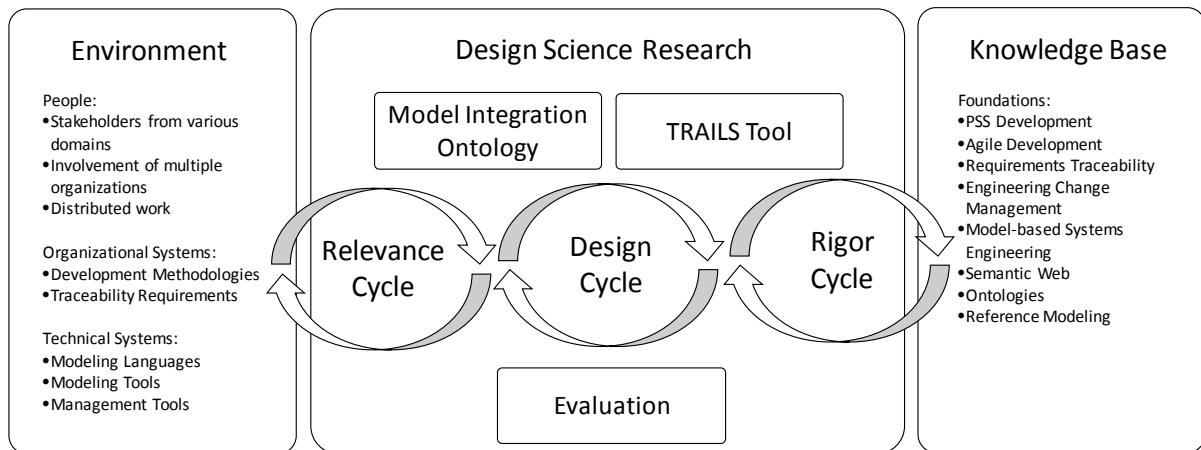


Figure 7: Cycles in Design Science Research applied to the dissertation topic

Source: Adapted from (Hevner 2007)

The research activities of the **(1) relevance cycle** unfold from the necessity to understand the environment in which the aspired artifacts need to be used in. This application area especially involves people, organizational systems and technical systems. To retrieve a clear picture of the issues to be solved, Design Science Research strives to analyze these issues within their practical setting and also identify the opportunities that can be realized. The relevance cycle therefore forms a bridge between the activities related to development and evaluation of the design science artifacts and their practical application domain.

In the context of this dissertation, the people related issues evolve due to the fact that stakeholders from various engineering domains need to be involved in the development of PSS. Furthermore, PSS engineering often requires the collaboration of multiple organizations in order to be able to provide the desired service. Both of these characteristics of PSS development also promote the distribution of engineering activities over multiple locations forcing the development team to work together remotely. Organizational systems that need to be regarded are amongst others the use of different engineering methodologies leading to synchronization issues within the PSS engineering problems as a whole. For example, in PSS development software development teams often pursue agile methodologies while hardware development teams are having a harder time with rapidly generating prototypes on a weekly basis and therefore often rely on more sequential development methods. Furthermore, from a technical perspective the application domain in which the artifacts developed during this dissertation operate in involve various modeling languages being used by the multiple engineering domains and organizations as well as different software tools that are used for model-based systems engineering and project management. After each iteration of the design cycle, its results are being mirrored to the real world issues.

In addition to the application environment Design Science Research also recommends to take a detailed look into scientific theories and engineering methods. In order to do so, the Design Science Research methodology demands a **(2) rigor cycle** in which the IT artifact under development is checked against the general knowledge base within the topic area. Only by checking against what has already been there, it is possible to develop truly innovative IT artifacts. The degree of innovativeness of an IT artifact is to a large fraction determined by

whether it is a valuable contribution to research within the area or just common problem solving. Hence, in this rigor cycle the researcher has to accomplish two things. First, important contributions to the various adjacent research areas need to be analyzed and their results need to be adapted to the problem under consideration and second, the experiences made during the development of IT artifacts need to be mirrored against and contribute to the research within those fields. While research contributions to the knowledge base are primarily targeted at the academic public, contributions that arise from the relevance cycle are focused more on industry practice as an audience (Hevner 2007). The research fields that are most relevant for this dissertation are of course requirements traceability, product service systems as well as model-based systems engineering all of which have been introduced in the preceding sections. However, one also needs to look into more distantly related topic areas. For example, development methodologies, such as agile development have an influence on which artifacts are being generated as a result of engineering activities and the research on engineering change management shows which traceability information for efficiently performing change management. In order to build on a strong fundament of academic knowledge, this dissertation contains multiple literature reviews in various areas of research.

The centerpiece of Design Science Research is the **(3) design cycle**. Within this cycle the researcher alternates continuously between further development and the evaluation of the IT artifact under consideration. These activities alternate in multiple iterations so that the development of the IT artifact receives regular feedback. As described earlier, the relevance cycle yields requirements from the application area and allows for practical evaluation whereas the rigor cycle delivers input from related research and checks the results against scientific methods and theories. It is important to always view the design cycle in the context of the relevance and rigor cycles and that there needs to be a balance between both of them in order to develop an innovative IT artifact and to establish this balance the cycles need to be traversed multiple times (Hevner 2007).

3.2 Research Methods

3.2.1 Literature Review and Expert Interviews

As just described Design Science Research recommends to explore the theoretical knowledge base iterating through the rigor cycle and to get practical insights while exploring the application environment in practice in the relevance cycle. During the research for this thesis, we passed through both of these cycles several times, conducting expert interviews in order to explore traceability from a practitioners perspective as well as literature reviews within the relevant scientific fields of study.

Conducting a literature review helps the researcher in identifying, evaluating and explaining the existing publications within the area of interest. This includes for example methods, theories, experiences concerning certain aspects of the topic area and research gaps in both, theory and practice (Vom Brocke et al. 2009; Cooper 1988; Fettke 2006). In order to analyze the current state of the art in requirements traceability research and evaluate whether existing approaches in this field are suitable for use in the development of product service systems we conducted literature reviews that draw from several research fields. Particularly we were

interested in requirements engineering research within the domains of software engineering, mechanical engineering and service engineering as well as publications on product service systems in general and model-based systems engineering.

In each case, we started our analysis by performing keyword searches in the literature databases Google Scholar, IEEE Xplore, ACM Digital Library, Springer Link and Emerald Insight. From the results we eliminated all duplicates as well as all publications that were concerned with traceability in domains not related to engineering products, services or systems (e.g. traceability within supply chains or in particular food traceability). We then reviewed the abstracts of the publications in order to decide whether a publication should be incorporated in the detailed analysis and selected those with the best fit to the topic area to undergo a detailed review. As recommended by Webster and Watson (2002), we further examined publications that were cited by the ones that we already selected (backward search) and conducted a forward search to identify publications that are citing the key publications (in terms of citation count) that were identified during keyword search. Again, each of the publications that was identified in this step was evaluated regarding its fit with the topic of research. The final set of publications were then analyzed in detail in order to identify, group and structure the underlying concepts (Fettke 2006). Overall, our analysis focused on models, methods and tools that support the PSS engineer in ensuring traceability throughout the lifecycle of PSS, especially focusing on traceability during the development stage and cross-domain challenges.

As a main outcome of performing those literature reviews, we were able to identify the different types of engineering artifacts as well as the basic entities and concepts which are used in these artifacts. Again, this collection of entities and concepts formed the basis for our model integration ontology.

In addition to these literature reviews, we explored how traceability was implemented in practice and which challenges arise in this regard by conducting 31 expert interviews in various industry sectors (e.g. public infrastructure, automotive, software and finance). The interviews followed semi-structured interview guidelines, each lasting between 40 minutes and one hour. All interviews were recorded, transcribed and finally evaluated using a software tool for qualitative data analysis (MaxQDA v12). Based on the insights from these interviews, we were able to get practical insights of whether and how different companies ensured traceability in requirements engineering in their development projects and which kind of difficulties they were facing.

3.2.2 Ontology Development

As explained in section 2.3.3 ontologies can be seen as formal specifications that define the terms or general concepts within a domain of interest as well as the semantic relations that exist between those terms. In general, ontologies can be used for various different purposes. First, ontologies are predestinated to share a common understanding of the structure of information among stakeholders or even software systems. As such, they facilitate the reuse of domain knowledge and can be used to analyze domain knowledge and make the

assumptions within a domain explicit. Furthermore, ontologies may also be used to separate domain knowledge from operational knowledge (Noy et al. 2001).

In literature various approaches for developing ontologies can be found, ranging from rather stage-based processes (e.g. TOVE Gruninger and Fox 1994) to continuously evolving prototype models (e.g. METHONTOLOGY Fernández-López et al. 1997). Although these approaches differ regarding the exact sequence, scope and repetition rate of the steps they contain, a comparative study (Jones et al. 1998) showed, that all more or less require the same three fundamental activities. First, one needs to create an informal description of the relevant concepts as well as their relationships with each other. Second, this informal model needs to be formalized. This means, it needs to be formally defined using an adequate ontology language. Third, the ontology needs to be implemented in a software system that allows to create instantiations of the concepts defined in the ontology and allow fo logical reasoning (Jones et al. 1998).

Accordingly, the best way to start ontology development is to select a specific (business or engineering) tasks that needs to be supported. In our case, this is the task of ensuring the traceability of requirements along the development of PSS. On this basis, it is possible to specify the domain that is addressed by the ontology as well as the scope it needs to cover. As a next step, one can consider reusing and enhancing other ontologies that already exist in that area (Noy et al. 2001). For this purpose, we analyzed various ontologies with different characteristics, ranging from universal ontologies to rather specialized ones that are focused on describing a particular field of knowledge.

Cyc for example, a more universal ontology defines more than 500,000 concepts from multiple areas, with a type hierarchy ranging from e.g. “type of US work visa” to “TV show format” (Cycorp 2016). Despite the impressive number of concepts defined in this ontology, we found particularly few concepts that are relevant to PSS engineering were defined while other areas of knowledge that are irrelevant for our purpose are overpopulated extensively. In summary, we found that such universal ontologies are not recommendable for the specialized purpose of PSS engineering.

The second type of ontologies that we looked into are specialized ontologies that focus on a particular problem but on a generic level. For example, Dublin Core is an ontology that defines a standard for describing the metadata and semantics of documents (DCMI 2017). As such, it is being used in the world wide web to specify the semantics of hypertext documents. While most of the concepts it specifies can be used for characterizing PSS engineering artifacts (especially traditional engineering documents such as technical drawings), it solely is not sufficient to describe the semantic relations that are needed for traceability in PSS development. Another ontology that focuses on capturing the semantic relationships within a network is FOAF (friend of a friend) which can be used to describe the semantics of social networks (Brickley and Miller 2014). As such, it only defines characteristics and relationships between people. This ontology delivers valuable input for our PSS engineering ontology as we also need to capture human stakeholders in the engineering process and some of their

social characteristics or relationships (e.g. organizational structure within a company or areas of knowledge that can be attributed to a developer).

Last, we looked into specialized ontologies that were designed for a certain domain. Good Relations, for example, is an ontology designed for improving the e-commerce sector through semantic descriptions. For this purpose it defines concepts like product and service, but only as a black box. This means, that the internal composition of a product or service or their development are not captured by the Good Relations ontology, but only the concept of a product or service itself (Hepp 2011). While some of the concepts defined in this ontology are suitable for describing PSS service provision and market-oriented aspects of a PSS, the ontology does not specify anything related to the engineering process and the internal structure of a PSS.

An ontology that addresses the engineering sector is for example the Engineering Ontology by Sevckenko and Mann (2002). Although its described area of application is engineering in general, this ontology primarily defines the physical and mathematical concepts that are needed for simulation models or different types of electrical or mechanical components. It is thus to be seen on a rather low technical level of engineering. What it does not capture is for example the various types of development artifacts and other important concepts that are needed for managing a PSS development projects.

Having defined the desired scope of the target PSS engineering ontology, we had analyzed the potential benefits of existing ontologies and concluded that none of them would be sufficiently expressive to capture the entire domain of interest. As a next step, ontology development methodologies suggest to conceptualize the domain knowledge (Fernández-López et al. 1997), i.e. identify and specify important terms within the knowledge domain (Noy et al. 2001). As explained in the section before, we accomplished this through literature reviews as well as the analysis of domain-specific modeling approaches.

Having done so, one can then start to formalize the conceptual model (Fernández-López et al. 1997). This means that first, the classes as well as the class hierarchy needs to be defined (c.f. Publication 5). Second, the properties (also known as slots) which define the attributes of a specific class or it's potential relationships with other classes need to be determined (c.f. Publication 7). Third, for each of the properties one can then define so-called facets, i.e. cardinality or value types of the properties (Noy et al. 2001). At this stage, the ontology can then be integrated with other existing ontologies (Fernández-López et al. 1997) and it should finally be implemented in order to make it computable (c.f. Publication 8). This way, it can serve it's intended purpose to bridge the gap between executable systems and the real world that it models (Jones et al. 1998).

3.2.3 Tool prototyping and evaluation

With regard to the overall research goal in the context of design science research we can differentiate between practical problems and knowledge problems. While solving a practical problem relates to finding out the difference between the real world and how stakeholders would like the world to be, solving knowledge problems demands the researcher to fill the

gap between what a stakeholder knows and what he needs to know for his task (Wieringa 2009). In this sense, this thesis focuses on solving a knowledge problem by providing a software tool that supports engineers in overviewing the system under development and reenacting its evolution.

Two essential activities in the DSR process are demonstration and evaluation. In the demonstration step, the research needs demonstrate the use of the artifact (in this case the software tool) by solving an instance of the problem (Peppers et al. 2007). We do this by applying our tool to the case study of developing a bike sharing system. In this context we modeled the bike sharing use case including the requirements specification, architectural description as well as the service processes. All of these models were generated using commercially available software tools that have a high degree of dissemination throughout various industries. During the following evaluation activity, one needs to observe and measure how well the artifact under consideration performs in solving the problem (Peppers et al. 2007). Due to the nature of the problem we are concerned with, we decided to perform tool evaluation also in workshops with researchers in the area of model-based systems engineering asking for direct feedback on the tool.

In order to gradually improve our software tool (TRAILS), we kept iterating between implementing new tool features or re-designing existing ones and evaluating how these features would find application in realistic case studies, always searching for potentials of improvement. While traditional methodologies often view iteration as an inevitable evil that needs to be avoided or at least minimized through better proactive planning, evolutionary or iterative development processes (such as DSR) embrace iteration as a mechanism that finally delivers better outcomes (Berente and Lyytinen 2007). In this respect, the design science approach literally suggests an endless iteration over implementation and evaluation. Practically however, it is too expensive or simply not valid to iterate over successive versions of a solution design. This is especially true, when the environment in which the tool is planned to find application is a complex social system, such as a PSS engineering team. As an alternative, researchers can resort to a technical action research approach, evaluating the tool by using it under conditions of practice (Baskerville 1997).

In context of the model integration approach that we propose, we followed the recommendations of Wieringa (2009) performing laboratory experiments in the form of a modeling case study of limited complexity and conducted expert modeling workshops to receive professional feedback. However, in order to ensure controllability of the setting and due to the absence of a suitable industry case study, we decided to refer to an academic case study of developing a free-floating bike sharing system. This case study was conducted within the collaborative research center SFB768⁸: *“Managing Cycles in Innovation Processes – Integrated Development of Product-Service Systems Based on Technical Products”* of which the research conducted for this thesis was a subproject.

⁸ See also: <https://www.sfb768.tum.de/en/home/>

In our case study PSSycle we guided a team of students from informatics and mechanical engineering in developing the prototype of a an electrically powered bicycle than can be rented using a smartphone application (see Figure 8). In this project, which had a total duration of 2 years, we developed mechanical, electrical as well as software components and the PSS business model. This way, it was possible to follow the entire development process from ideation and need analysis to prototyping of hardware, software and service processes.

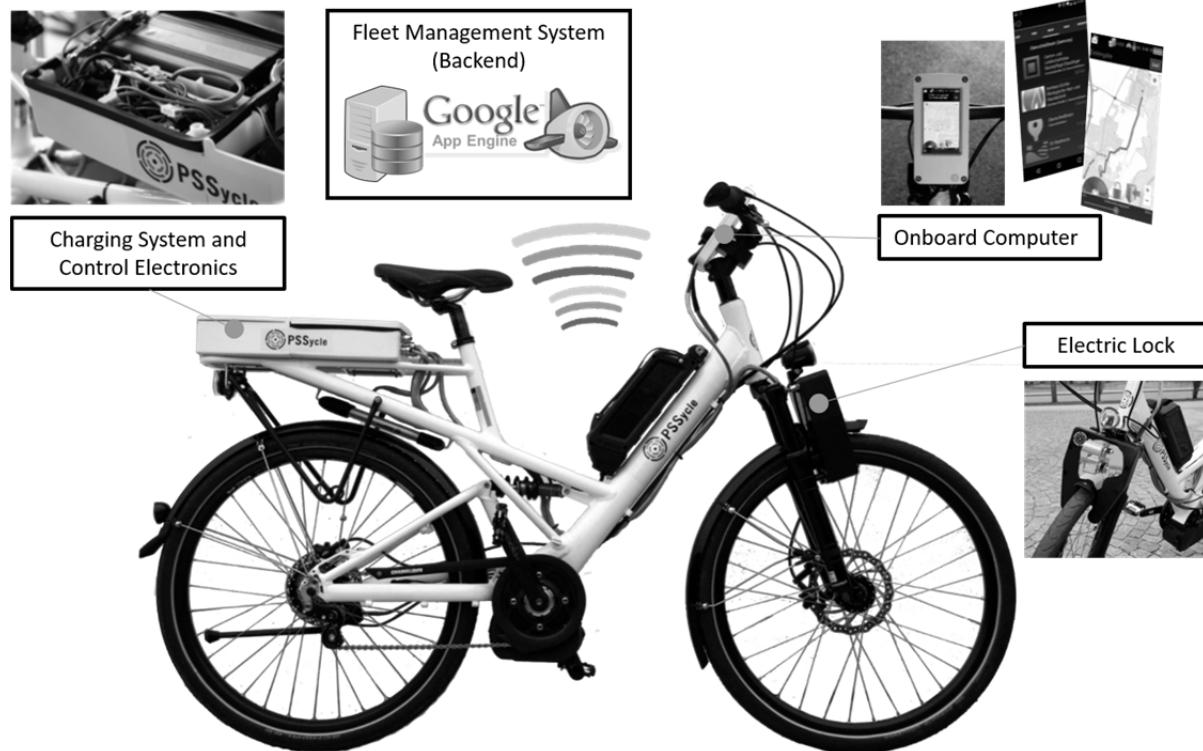


Figure 8: Overview of the bike sharing PSS case study: PSSycle
Source: own illustration

Researchers often view information systems as technical systems with their development being viewed as a socio-technical process that involves communication, participation and power. However, agreeing with Berente and Lyytinen (2007) we rather see information systems as organizational and social communication systems. Consequently, information systems are undergoing constant evolution and they behave different in different social contexts. Hence, there is no single entity that can be considered as the system but rather the system is some vague idea and it can be approximated through representations (Berente and Lyytinen 2007).

PART B: PUBLICATIONS

Overview of Publications included in this Dissertation

Table 4: Publications included in this dissertation

No. [Type]	Autors	Title	Outlet
P1 [C]	Thomas Wolfenstetter , Simon Bründl, Markus Böhm, Helmut Krcmar	Why Product Service Systems Development is Special DOI: 10.1109/IESM.2015.7380308	IESM 2015, Seville, Spain ISBN: 978-2-9600532-6-5
P2 [C]	Thomas Wolfenstetter , Sebastian Floerecke, Markus Böhm, Helmut Krcmar	Analyse der Eignung domänenspezifischer Methoden der Anforderungsverfolgung für Produkt-Service-Systeme	WI 2015, Osnabrück, Germany ISBN: 978-3-00-049184-9 VHB-Jourqual ⁹ : C
P3 [C]	Nepomuk Chucholowski, Thomas Wolfenstetter , Martina Wickel, Helmut Krcmar, Udo Lindemann	Towards Cycle-Oriented Traceability in Engineering Change Management	DESIGN 2014, Dubrovnik, Croatia ISSN: 1847-9073
P4 [C]	Thomas Wolfenstetter , Jonas Zitzelsberger, Markus Böhm, Helmut Krcmar	Traceability von Anforderungen und Tests in agilen Softwareentwicklungsprojekten	SE&M 2015, Dresden, Germany ISBN: 978-3-88579-633-6
P5 [C]	Konstantin Kernschmidt, Thomas Wolfenstetter , Christopher Münzberg, Daniel Kammerl, Suparna Goswami, Udo Lindemann, Helmut Krcmar, Birgit Vogel-Heuser	Concept for an Integration-Framework to enable the crossdisciplinary Development of Product- Service Systems DOI: 10.1109/IEEM.2013.6962430	IEEM 2013, Bangkok, Thailand ISBN: 978-1-4799-0986-5
P6 [C]	Thomas Wolfenstetter , Konstantin Kernschmidt, Christopher Münzberg, Daniel Kammerl, Suparna Goswami, Udo Lindemann, Birgit Vogel-Heuser, Helmut Krcmar	Supporting the cross-disciplinary development of product-service systems through model transformations DOI: 10.1109/IEEM.2014.7058623	IEEM 2014, Bandar Sunway, Malaysia ISBN: 978-1-4799-6410-9
P7 [C]	Thomas Wolfenstetter , Simon Bründl, Kathrin Füller, Markus Böhm, Helmut Krcmar	Towards a Requirements Traceability Reference Model for Product Service Systems DOI: 10.1109/IESM.2015.7380307	IESM 2015, Seville, Spain ISBN: 978-2-9600-5326-5
P8 [J]	Thomas Wolfenstetter , Mohammad R. Basirati, Markus Böhm, Helmut Krcmar	Introducing TRAILS: A Tool supporting Traceability, Integration and Visualisation of Engineering Knowledge for Product Service Systems Development DOI: 10.1016/j.jss.2018.06.079	Journal of Systems and Software ISSN: 0164-1212 h5-index ¹⁰ : 51

Notes: [C]: Conference; DESIGN: Proceedings of the 13th International Design Conference; IEEM: IEEE International Conference on Industrial Engineering and Engineering Management; IESM: International Conference on Industrial Engineering and Systems Management; [J]: Journal; P: Publication; SE&M: Multikonferenz Software Engineering & Management 2015; WI: 12th International Conference on Wirtschaftsinformatik

⁹ <http://vhbonline.org/vhb4you/jourqual/vhb-jourqual-3/teiltrating-wi/>

¹⁰ https://scholar.google.com/citations?view_op=top_venues&hl=en&vq=eng_softwareystems

Table 5: Further publications by the dissertation's author

Year [Type]	Autors	Title	Outlet
2018 [J]	Alexander Herzfeldt, Thomas Wolfenstetter , Christoph Ertl, Helmut Krcmar	The role of individualization and project learning for cloud service profitability DOI: 10.4018/JECO.2018040104	JECO Vol. 16 • No.2 ISSN: 1539-2937
2017 [J]	Alexander Herzfeldt, Thomas Wolfenstetter , Markus Böhm, Helmut Krcmar	From Products and Service to IT solutions: Nine Principles for Managing IT solutions DOI: 10.2979/eservicej.10.2.01	e-Service Journal, Vol. 10, No. 2 ISSN: 1528-8226
2017 [C]	Toias Engel, Thomas Wolfenstetter , Nikolaos Sakiotis	Optimizing distribution logistics within cities through time-slot deliveries DOI: 10.15480/882.1479	HICL 2017, Hamburg, Germany ISBN: 978-3-7450-4332-7
2015 [C]	Stefan Feldmann, Sebastian J. I. Herzig, Konstantin Kernschmidt, Thomas Wolfenstetter , Daniel Kammerl, Ahsan Qamar, Udo Lindemann, Helmut Krcmar, Christiaan J. J. Paredis, Birgit Vogel-Heuser	Towards effective management of inconsistencies in model-based engineering of automated production systems DOI: 10.1016/j.ifacol.2015.06.200	IFAC 2015 ISSN: 2405-8963
2015 [J]	Sebastian Floercke, Thomas Wolfenstetter , Helmut Krcmar	Hybride Produkte–Stand der Literatur und Umsetzung in der Praxis	IM+ io Vol. 30, No. 2 ISSN: 1616-1017
2015 [C]	Stefan Feldmann, Sebastian J. I. Herzig, Konstantin Kernschmidt, Thomas Wolfenstetter , Daniel Kammerl, Ahsan Qamar, Udo Lindemann, Helmut Krcmar, Christiaan J. J. Paredis, Birgit Vogel-Heuser	A comparison of inconsistency management approaches using a mechatronic manufacturing system design case study DOI: 10.1109/CoASE.2015.7294055	CASE 2015, Gothenburg, Sweden, ISBN: 978-1-4673-8183-3
2014 [C]	Olga Roht, Tobias Engel, Thomas Wolfenstetter , Suparna Goswami, Helmut Krcmar	An Analysis of Synergy Effects between Closed-Loop Supply Chains and Product-Service Systems	POMS 2014, Atlanta, USA
2013 [C]	Andreas Kohn, Julia Reif, Thomas Wolfenstetter , Konstantin Kernschmidt, Suparna Goswami, Helmut Krcmar, Felix Brodbeck, Birgit Vogel-Heuser, Udo Lindemann, Maik Maurer	Improving common model understanding within collaborative engineering design research projects DOI: 10.1007/978-81-322-1050-4_51	ICoRD 2013 ISBN: 978-81-322-1050-4

Year [Type]	Autors	Title	Outlet
2011 [C]	Armin Sharafi, Thomas Wolfenstetter , Petra Wolf, Helmut Krcmar	Analysis of Current IT Support for Product Development Processes	ICoRD 2011
2010 [C]	Armin Sharafi, Thomas Wolfenstetter , Petra Wolf, Helmut Krcmar	Comparing product development models to identify process coverage and current gaps: A literature review	IEEM 2010

Notes CASE: IEEE International Conference on Automation Science and Engineering; HICL: Proceedings of the Hamburg International Conference of Logistics; ICoRD: International Conference on Research into Design; IEEM: IEEE International Conference on Industrial Engineering and Engineering Management; IFAC: International Federation of Automatic Control; IM+ io: Magazin für Innovation, Organisation und Management; JECO: Journal of Electronic Commerce in Organizations; POMS: International Meeting of the Production and Operations Management Society

Publication 1

Why Product Service Systems Development is Special

Thomas Wolfenstetter^a, Simon Bründl^b, Markus Böhm^a, Helmut Krcmar^a

<p>^a Chair for Information Systems Technische Universität München Munich, Germany {thomas.wolfenstetter, markus.boehm, krcmar}@in.tum.de</p>	<p>^b Institute for Information Systems and New Media, Ludwig-Maximilians-Universität München Munich, Germany bruendl@bwl.lmu.de</p>
---	--

Abstract

The design and development of product service systems (PSS) is a complex process that brings together product, software and service engineering. A fully integrated PSS calls for significant collaboration among the different engineering disciplines along the entire design and development process which can pose several challenges to the development team. Therefore, when developing a PSS, companies should take into account the special characteristics and complexities that are relevant in the context of PSS. However, in practice companies often follow their traditional development processes and simply design a service around their existing products. This way, many PSS offerings do not live up to their full potential. In order to overcome this issue practitioners need to know what makes PSS development special. By reviewing existing literature on PSS development, this paper identifies and describes the special characteristics of PSS development in contrast to traditional products or services.

Keywords— Product Service System; Development; Characteristics; Literature Review

1. Introduction

In an increasingly globalized world, companies, especially manufacturing companies, find themselves confronted with fierce competition in which it is hard to differentiate oneself just on the basis of their products. In most of the cases there are a large number of competitors that are capable of delivering a product of comparable quality. This often leads to ruinous price wars causing margins to collapse (Becker and Krcmar 2008). Therefore, manufacturing companies across various industry sectors increasingly realize that they can realize additional profits and tighten their customer relationships by offering services that complement and

enhance their original product portfolio. In doing so, they create bundles of products and services that form a specialized solution for a certain problem or need of the customer. These bundles are often described as so called product service systems (PSS), i.e. integrated customer solutions consisting of product and service components that generate value in use (Baines et al. 2007).

In general, mainly three types of PSS can be differentiated: (1) product-oriented PSS, (2) use-oriented PSS and (3) result-oriented PSS (Tukker 2004). In product-oriented PSS, the main focus is still on the selling and vending of products and only few services such as maintenance and repair are added. Product-oriented PSS can be further classified into product-related services as well as advice and consultancy. Product-related services refer to additional services which are needed during the use phase of the product (Tukker 2004) e.g. maintenance or financing schemes. In contrast, advice and consultancy are services that guide the efficient and effective use of the product sold. For instance, training and demonstrations are examples of advice and consultancy services.

Use-oriented PSS refer to the use or consumption of a particular product, and the services associated with such use-oriented solutions. Use-oriented PSS can be differentiated into product lease, product renting/sharing and product pooling. Here the product itself stays in the ownership of the provider, but is made available to and shared by numerous users (Tukker 2004). While the user has unlimited access to the leased product e.g. a car, this is not the case for product renting/sharing where the product is used by a number of different customers in succession. In contrast, the PSS business model of product pooling enables simultaneous use of the product by a certain number of users, e.g. sharing of a car for a ride to work so that several consumers use the car at once.

Result-oriented PSS focus on the service component of the PSS. The provider and the customer agree on a pre-specified result, which is later delivered by the provider (Tukker 2004). Examples for result-oriented services are the delivery or outsourcing of activities such as cleaning, catering or hosting, support and maintenance of IT infrastructure.

While PSS provide manufacturing companies with an opportunity to gain competitive and strategic benefit, they also pose significant challenges and complexities in terms of designing such systems. Companies are faced with the need to shift from a product orientation to a service orientation, and manage the inherent interdependencies between the product and service components, more so since each of these components are likely to be designed and developed by different business organizations. Developing a PSS incorporates the integration of components from multiple engineering disciplines such as mechanical engineering, software engineering and service engineering. In this regard it does not make sense to separate the development into domain-specific processes. Instead the development of those systems is a challenging task and calls for an integrated multi-domain engineering process that comprises mechanical, software and service engineering (Hepperle et al. 2010). Since PSS are customer specific solutions the central point of reference for PSS development are the customer's demands which are to be fulfilled by the solution (Burianek et al. 2007).

Given these challenges, there is a need to move beyond a purely product-centric orientation, or a purely service dominant logic, in order to develop successful PSS. However, in practice companies often follow their traditional development processes and simply design a service around their product. This way, many PSS offerings do not live up to their full potential. Based on a review of existing literature this paper aims at identifying and categorizing the special characteristics that companies have to face in the course of PSS development

2. Methodology

The research is guided by the following research question: What are the special characteristics of PSS that differ their development from traditional products and services? To answer this research question, we conducted a literature review as recommended by Webster and Watson (Webster and Watson 2002), which includes a keyword search in selected relevant sources, a forward and backward search, and a subsequent content analysis of the as relevant evaluated publications. The initial search included the scientific databases ACM Digital Library, AIS Electronic Library, EBSCOhost, IEEE Xplore Digital Library, Science Direct, and the scientific search engine Google Scholar. The literature search was based on a structured keyword search. We used the keywords “PSS”, “product-service system”, “servitization” and “hybrid product”. All keywords have been expanded into synonymous German terms. In each database we then looked at the abstracts of the 500 most cited publications in order to decide whether a publication should be incorporated in the detailed analysis.

After an initial review and compression of redundant duplicates a total of 106 papers remained for secondary analysis. In this secondary analysis, relevant articles were reviewed and analyzed on a full text basis. This process resulted in 60 articles, which provide a significant contribution to the research question of this study. As part of the forward and backward search, twelve additional relevant articles were identified. The identified papers have been analyzed using a concept centric approach as recommended by Webster and Watson (2002).

3. Special Characteristics of PSS Development

Based on our literature study we were able to identify nine characteristics that distinguish the development of PSS from traditional products and services. We categorized these characteristics into (1) Integration and Multidisciplinarity, (2) Provision of solutions as well as (3) Environment and Organization (see Table 1).

et al. 2011)									
(Leimeister and Glauner 2008)									
(Li and Liu 2010; Roy et al. 2009a)									
(Liu et al. 2010)									
(Luiten et al. 2001)									
(Manzini and Vezzoli 2003)									
(Maussang et al. 2009)									
(Mont 2002, 2000)									
(Mont and Tukker 2006)									
(Baxter et al. 2009; Morcos and Henshaw 2009; Vezzoli et al. 2012)									
(Morelli 2002, 2006; Phumbua and Tjahjono 2012)									
(Wilkinson et al. 2009)									
(Roht et al. 2014)									
(Schweitzer 2010)									
(Spath and Demuß 2006)									
(Sturm and Bading 2008)									
(Sun and Zhang 2012; van Ostaeyen et al. 2013)									
(Tukker 2004; Wu and Gao 2010)									
(Mont and Tukker 2006)									
(Wang et al. 2011)									
(Wolf et al. 2010)									
(Yang et al. 2009; Yang et al. 2010, 2011)									
Sum	15	9	25	32	20	10	12	21	17

The first category, Integration and Multidisciplinarity, summarizes features that are related to or result from the integration of the product and service components that form a PSS. Furthermore, these components are subject to different life cycles and other cycles that influence development and service provision. As a consequence, PSS development and service provision require an intense collaboration of different engineering disciplines, such as mechanical engineering and service engineering.

The orientation towards the Provision of Solutions to individual customer needs is accomplished through modular system architectures. The fact that system modules are individually combined to satisfy a specific need further results in a high variability of service provision. In order to be able to cater to customer needs individually further requires a high degree of customer integration in both, development and service provision.

In addition to that, PSS further features that are related to Environment and Organization can be distinguished. They comprise organizational challenges, the value network integration as well as the sustainability goal that is often associated with PSS.

3.1. Integration of Components

PSS combine products and services to deliver an additional value to the customer (Becker and Krcmar 2008). This often requires a technical integration of the components resulting in interdependencies (Mont 2002). Some experts even view this integration as the central feature of PSS development (Burianek et al. 2007; Böhmman and Krcmar 2007). According to Aurich, Fuchs and Wagenknecht (2006), simply adding a service to an existing product is not productive. In fact, a holistic approach is necessary in order to generate the desired outcome (Baxter et al. 2009). Based on stakeholder requirements the various components need to be aligned and integrated. Thus, an isolated analysis of the required solution components is insufficient (Burianek et al. 2007). As a consequence PSS development is by far more complex than traditional engineering (Creusen 2011).

Changes on the physical product eventually lead to modifications to the package of services. Vice versa do modifications of services have an impact on the product components (Becker and Klingner 2013). These interdependencies lead to an increased complexity, which is much higher compared to a delimited consideration of goods and services (Mont 2002). Therefore, product and service components of a PSS need to be developed and aligned in an integrative PSS development process (Wang et al. 2011). Only then interdependencies between product and service are adequately taken into account. On the other hand, Yip, Phaal and Probert (2014) represent the opinion that a PSS development process is either product or service driven. However, no matter what discipline drives the development, a high degree of integration of physical and non-physical components requires a systematic co-dependent process for product and service development (Aurich et al. 2006). Spath and Demuß (2006) describe a solution approach for PSS as the consistent and customer integrated product and process development, which is extended by a design method of service engineering.

3.2. Multidisciplinary Development

The combination of products and services and therefore the integration in PSS involves different domains in the development process. In order to take cross-disciplinary aspects into account, both the development and production of the physical product as well as the non-physical service must be considered systematically (Aurich et al. 2006). Mont and Tukker (2006) state that the multidisciplinary approach influences a wide range of disciplines such as economics, environmental science, sociology, psychology, product design and engineering. In general three disciplines are mainly involved in the development of PSS: product, software and service development (Herzfeldt et al. 2011). A stakeholder in the PSS development process has typically specific expertise in one of the disciplines. A deeper knowledge about the other disciplines is desirable but rare in practice. It is therefore necessary to derive a common sense for cross-disciplinary problems (Gürtler et al. 2013). The consideration of all engineering disciplines has to be done early in the requirements analysis (Berkovich et al.

2011a). A successful implementation requires the introduction of interfaces to coordinate and integrate the participating developers in the development process. The general objective is an integration of different engineering disciplines to a systematic development approach of powerful tangible and intangible components (Spath and Demuß 2006).

3.3. Customer Integration

In order for the PSS provider to compose an individual service offering the customer has to get actively involved in the service development (Langer 2013; Tuli et al. 2007). According to Böhmman and Krcmar (Böhmman and Krcmar 2007), this integration comprises the technical and organizational embedding of the solution into the business processes of the customer. Especially when developing use-oriented and result-oriented PSS for industrial customers, the technical integration of software and service components into the customers system landscape has to be considered (Burianek et al. 2007; Berkovich et al. 2011a). This requires the PSS provider to have a broad understanding of the customer-specific business processes (Langer 2013; Sturm and Bading 2008). In addition to that PSS require a high degree of customer interaction during development which exceeds traditional customer-orientation (Isaksson et al. 2009; Laperche and Picard 2013). In the scope of hybrid value creation the relationships between the PSS provider and the customer has to go beyond the integration of the customer in the development process. In PSS the customer becomes a vital part of the PSS business model. From the customers' point of view, the acceptance of a PSS is based on the relative evaluation of the novel PSS-concept versus already existing products (Lee and Park 2010). Customers value the performance, availability and shifting of risks to the provider (Baines et al. 2007). The PSS business model aims at offering the customer increased value at lower costs while at the same time being more sustainable than traditional business models (Mont and Tukker 2006). To support their business strategy PSS providers try to build up and keep a strong relationship to their client base (Phumbua and Tjahjono 2012). This enables the PSS provider to collect information about the customers' needs (Wu and Gao 2010). The results are long lasting commercial partnerships between the PSS provider and the customer (Alonso-Rasgado et al. 2004; Wolf et al. 2010). In contrast to traditional business models the responsibility of a PSS provider does not end with the sale, the interaction between provider and customer in the utilization phase leads to dynamic changes as well. For the PSS provider it is important to understand and to be able to cope with the risks caused by this dynamic to guarantee compliance of the contract (Phumbua and Tjahjono 2012). Changes in the market lead to changed requirements of the business model (Phumbua and Tjahjono 2012). Therefore the continuous innovation of the business model is crucial for the long-term success of the PSS provider (Hao 2012). According to Böhmman and Krcmar (Böhmman and Krcmar 2007), the goal-oriented transformation enables an orientation to changed customer needs even later in the process of service provision. As a consequence the companies therefore have to continuously modify or even redefine their existing PSS business model (Hao 2012).

3.4. Lifecycle Orientation

In contrast to traditional business models the lifecycle management of PSS focuses on development and realization of the necessary user functionalities along the entire product

lifecycle (Laperche and Picard 2013). According to Mont (2002) as well as Li and Liu (2010), the successful development of a PSS requires the providers of hybrid bundles of services to broaden their participation and responsibility onto the phases of the lifecycle which are not in their responsibility scope if you take a look from the traditional point of view. This implies for PSS providers implementation and management of ‘closed-loop supply chains’ which cover the forward- as well as the backward flow of consumable supplies (Roht et al. 2014). The ownership for use-oriented and result-oriented PSS remains in most cases with the provider (Baines et al. 2007; Geum and Park 2011; Mien et al. 2005; Roy et al. 2009a; Liu et al. 2010; Maussang et al. 2009; Wilkinson et al. 2009). For the PSS provider this results in increased responsibility over the product lifecycle. The customer profits from the reduced responsibility, splitting of risks and decreased support departments (Isaksson et al. 2009). From the view of the provider the PSS lifecycle begins with the planning and development, succeeded by the manufacturing, the according service delivery and the disposition (Roy et al. 2009b; Schweitzer 2010). This broad perspective containing the whole lifecycle is indispensable for reaching a good performance of the PSS (Wang et al. 2011). Through the offering of integrated, hybrid bundles of services the motivation of the firm to change the product design increases. Products where the ownership is not transferred to the customer are less price-sensitive. The minimization of costs over the entire lifecycle thus is a strong driver for product development. Simplification of maintenance and remanufacturing are recognized as valuable by the PSS provider, too (Roy et al. 2009a). In addition to the necessity of a closed circuit for the consumable supplies, there need to be bi-directional information flows between the provider and the customer (Mien et al. 2005). The PSS provider continuously gains information about the utilization behavior and potential trends through the cooperation with the customer (Roht et al. 2014; Schweitzer 2010). The balanced planning and development of PSS has to be conducted with the goal of guaranteeing requirements suitable extension of products and services over the entire lifecycle. To gain advantages the relevant products and services have to be integrated over all phases of the product lifecycle (Liu et al. 2010). The majority of influencing factors for the PSS development have a temporal, often repetitive character. Some examples for externally triggered cycles in the context of PSS development include: The availability and maturity of technologies, competitive trends, different lifecycles of hard- and software, changed customer requirements, financial changes, development changes or legislative changes (Berkovich et al. 2011c). Moreover, PSS are not only subject to external but also internal cycles (Schenkl et al. 2013). This means development- and production processes have to be adjusted, staff and organizational structures change and the flow of information has to be defined and coordinated. Internal and external cycles depend on and influence each other. Every component has independent characteristics like storage life, maximal useful life or value to the market. Additionally they contain properties like option to repair, reuse and recyclability affecting the lifecycle (Komoto et al. 2005). The different items of work are therefore influenced by their own lifecycle (Böhmman and Kremer 2007). Especially the heterogeneous lifecycles of PSS components are a challenge for the development of a PSS. Due to the heterogeneity of the components’ lifecycles high coordination and iterative adjustments of the PSS between the different domains is required (Berkovich et al. 2011c). The involved disciplines have to ensure tight coordination regarding the temporal dependencies (Schenkl et al. 2013). For the efficient

development of PSS the potential cyclic influences and temporal dependencies during the entire lifespan of the PSS have to be considered (Langer et al. 2009).

3.5. Variability of Service Delivery

The basic idea of PSS is that customers do not ask for specific products or services but rather to solve a problem or to fulfill a demand (Leimeister and Glauner 2008; Sun and Zhang 2012; Yang et al. 2009). The customer has no interest in owning the product, but on its use (Isaksson et al. 2009; Maussang et al. 2009). Becker et al. (2009) state that the altered customer expectations, results in significantly increased requirements which means higher complexity for the provider. A possible solution is to provide alternative variations of goods and services. Services may adopt functions of the physical product and vice versa. The increased degree of freedom leads to a higher complexity in the PSS development process, but also offers new business opportunities if the variation complexity is managed well and supported systematically (Gürtler et al. 2013; Wolf et al. 2010).

3.6. Individualization

Variations in PSS offer specific service deliveries for individual customers. This solution-oriented and individual adaption to rising customer requirements is one of the main challenges for providers of PSS (Langer 2013). Field experience shows that PSS, compared to pure physically products, need a profound adaption of customer-specific requirements (Wolf et al. 2010). PSS is based on parts, which are categorized into components and modules (Böhmman et al. 2008). Based on customer requirements modules are built on the level of solution components. Already existing modules rated regarding their reusability. This categorization enables the standardization of modules (Berkovich et al. 2011a).

For targeted individual design must the variations meet differentiated tangible and intangible components, in order to provide a customized products and services as a systematically combination of PSS modules (Wolf et al. 2010). The real configuration of the PSS is thus an integrated system of standardized and customer-specific modules and components (Böhmman and Krcmar 2007).

3.7. Organizational Challenges

The integration of products and services requires transformations of the provider's organizational structure and internal business processes (Baines et al. 2007; Mont 2002; Wilkinson et al. 2009). Organizations which conduct this transformation not only have to take a look up on their business model but also on their processes and procedures, relations to suppliers as well as their employees' mindsets (Baxter et al. 2009). The transformation to a service oriented approach leads to changes regarding company culture, competences, knowledge and commercial partnerships while at the same time increasing complexity (Mont 2002; Laperche and Picard 2013). Intra-organizational infrastructures like for example acquisition, production, store and distribution but also the company's strategy or the quality management decide about competitive advantages against competing provider. Therefore, the

PSS development requires a focus on internal infrastructure not recognizable by the customer. The increased company interaction with other organizations results in inter-organizational changes which require the use of new performance indicators and employees. These internal transformations lead to a modification of the relationships between the company functions within the organization (Mont 2002). Companies transforming to PSS providers have to face challenges which are bigger than for organizations that are either product- or service oriented. The different orientations of the customer-centered front-end and the product-centered back-end lead to divergent hierarchies, processes, incentives and objectives (Böhm and Krcmar 2007; Roy et al. 2009a). Furthermore, there is the necessity of organizational changes when transforming to a PSS provider but also mention that more knowledge about how these changes have to look like and how the change process in the company has to be handled is required (Roy et al. 2009a).

3.8. Value Network

In a PSS the solution is provided by an integrated value network in which multiple actors interact with each other (Luiten et al. 2001). Therefore, the development does not only involve different disciplines and components, but also various stakeholders from different organizations (Berkovich et al. 2011a).

This leads to substantial changes regarding the way relationships with external stakeholders are being managed. For example, the implementation of closed-loop supply chains requires tight collaboration with the suppliers as well as the customer (Mont 2002). PSS often require competencies, resources and capabilities that are new to the organization thus requiring integrative partnerships with business partners (Wilkinson et al. 2009). It is therefore essential for PSS providers to actively manage the composition of the network of partners and identify how each partner can contribute to the value network. Especially for complex PSS this calls for a high level of trust and information sharing between partners. In contrast to traditional product development these partnerships are not limited to the development phase but along the entire lifecycle as the provision of the desired solution can often only be realized in partnerships (Morelli 2006). However, in these partnerships each stakeholder might have his own, often conflicting goals (Wilkinson et al. 2009). As PSS are the result of a collaborative value-added process among partners the traditional value chains become value networks (Baines et al. 2007; Isaksson et al. 2009; Schweitzer 2010; Wolf et al. 2010). Durugbo (2011) emphasizes the need for PSS providers to implement a network-thinking mindset in order to master the interorganizational challenges related to PSS. In total, the effective collaboration of heterogeneous partners in multiple organizations is a key premise for developing successful PSS.

3.9. Sustainability

In many cases the basic principle behind PSS is the shift from volume-driven production to value-driven business models (Mien et al. 2005). The goal of this paradigm change is dematerialization leading to more sustainable ways of creating value. Sustainability in this regard refers to economic as well as environmental aspects. Mont and Tukker (Mont and

Tukker 2006) describe PSS as business models that deliver more value to the customer at lower costs while at the same time being less harmful to the environment than traditional business models. The major advantage of a PSS is the renunciation of conventional product concepts shifting the focus onto the basic customer need. This allows for a higher degree of freedom for developers (Tukker and Tischner 2006). Because in most cases the ownership of PSS components remains with the provider, there is a strong incentive to minimize the cost and amount of material used (Laperche and Picard 2013). PSS providers therefore implement closed-loop supply chains as they are mostly responsible for retraction, upgrading and refurbishing of components in the loop (Mont 2002; Aurich et al. 2006; Roht et al. 2014).

The environmental efficiency of PSS thus results from a lifecycle wide optimization of resources resulting from a convergence of the stakeholder's interests (Manzini and Vezzoli 2003). Because of the holistic perspective of the development process focusing on optimization for long-term use, the development of sustainable PSS is affected by higher complexity (Luiten et al. 2001). Tukker and Tischner (2006) highlight that the different types of PSS vary in how sustainable they usually are. Product-oriented PSS often only add services to existing products and therefore mostly add little to the sustainability of the solution. Use-oriented PSS on the other hand tend to intensify the use of the product which is often more sustainable if they are being used to capacity. Result-oriented PSS have to potential to limit the resources required to the fulfilment of the original customer need therefore being more sustainable.

In summary, the development of more sustainable PSS is big challenge but at the same time an even bigger opportunity for companies (Luiten et al. 2001). Laperche and Picard (2013) highlight that the awareness for sustainability needs to be present in the entire organization. In order to systematically profit from positive effects while avoiding obstacles, there is need for flexible approaches for PSS engineering and service provision (Aurich et al. 2006). In this context it is most promising, if the design for sustainability is already carefully considered during the development stage (Mont and Tukker 2006).

4. Discussion

By analyzing the state of the art in literature on PSS we were able to identify nine characteristics that differentiate the development of PSS from traditional product or service engineering (c.f. Fig 1). The integration of products and service leads to mutual dependencies of the components of a PSS since modifications on one side may affect other parts of the solution as well. Moreover, the different hardware, software and service components of a PSS have different lifecycles, so that changing specific components at later lifecycle stages needs to be already considered during development. The need for consistency of development artifacts requires intensive coordination of the various disciplines involved, thus making the development of PSS more complex than traditional development. As each engineering discipline has specific characteristics, various perspectives and different engineering, cycles PSS development requires a cross-disciplinary approach for managing the interfaces and interdependencies. As the multiple disciplines have to be integrated in a holistic engineering process, the development of PSS also requires a broader sphere of competence than traditional

approaches. Since the focus of a PSS does not lie on a product or a service itself but on satisfying a customer need their development requires a value-based design approach in which different service or product components may be freely combined in order to form the solution. This results in a higher variability of service provision. However, the customer individual service provision also produces a bigger range of variants that have to be managed.

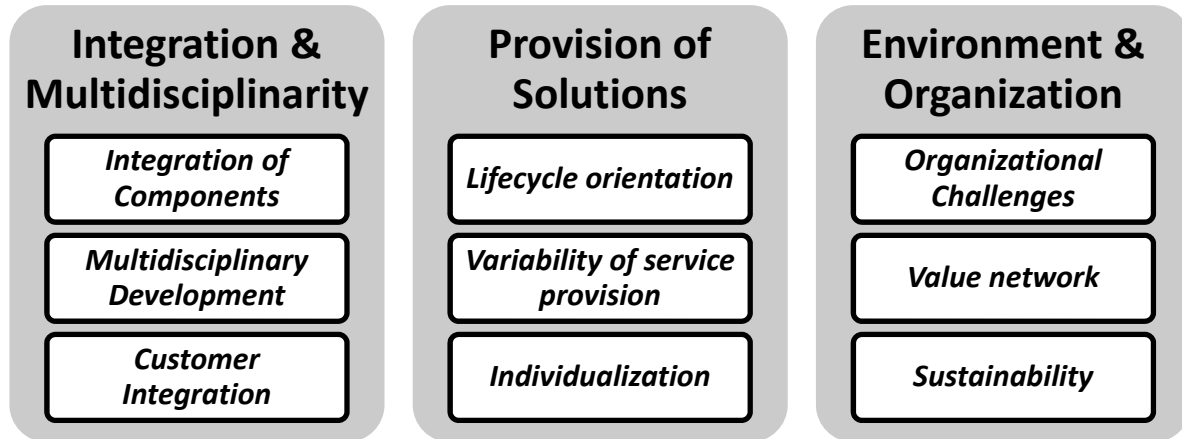


Figure 9: Nine Characteristics of PSS development

The variability of the customer individual composition of a PSS favors a design approach that allows for individualization through modules. If the promised solution can be broken down into partial solution components PSS can be adapted to individual customer needs by combining standardized and customer individual components. The fact that PSS need to be individually adapted to customer needs demands for intensive customer integration in the development process of PSS that goes beyond traditional levels of customer orientation. However, the integration of customers is not limited to the development of a PSS. With PSS the customer is also part of the business model as the value is co-created together with the customer. The interaction of the provider and the customer during the lifecycle of a PSS is subject to dynamic changes. The effects of those dynamic changes need to be anticipated and controlled. Therefore, PSS therefore require the consideration of the entire lifecycle including the design of closed-loop supply chains for service provision already during the development phase. With the perspective shifting to full lifecycle consideration the provider should not only keep development and production costs in mind but all costs that arise during the lifecycle of a PSS.

The development of a PSS and the subsequent phase of service provision also feature a number of organizational challenges that force the PSS provider to align two sometimes contradicting goals. Customer orientation requires the company to react flexibly in order to satisfy dynamically changing customer needs. At the same time the company needs to optimize internal structures of the organization to remain competitive. The difference between the internal and external focus manifests itself in diverging hierarchies, processes and goals. Furthermore, the goals of the PSS provider need to be aligned with those of its partners as development and service provision is done in value networks. In this context changes in the market or in PSS may also force partners to adapt. Inter-organizational dependencies therefore need to be considered already during the development stage. The development of a PSS thus

requires a high level of trust and collaboration among internal and external stakeholders. Due to the increasing importance of value networks and ecosystems PSS are often viewed as business models focusing on economic sustainability. One of the main goals of PSS is the shift from a volume-driven to a value-driven economy. This paradigm change is often believed to result in de-materialization of value-chains and therefore be more environmentally sustainable. However, PSS are not per se more sustainable than traditional products or services. In order to realize the potentials of less resource consumption and higher efficiency, a PSS provider needs to take a lifecycle sustainability perspective already during the development of a PSS.

For the PSS provider, the research findings can be translated into nine design recommendations:

1. Implement a traceability strategy to actively manage the interdependencies between the discipline specific development artifacts.
2. Enhance the collaboration between stakeholders of the various disciplines.
3. Actively integrate the customer along the entire lifecycle of a PSS and engage in tight customer relationships.
4. Pro-actively manage and adapt to the various cycles that affect the development of a PSS as well as service provision.
5. Start the development of a PSS from solution independent requirements and evaluate multiple alternatives of service provision.
6. Individualize the service offering by using a modular PSS architecture.
7. Be aware that changing the business model might require changing the organizational structure of the company.
8. Concentrate on the core competencies and collaborate with external partners in development as well as during service provision.
9. Create economically and environmentally sustainable solutions by focusing on the essential needs of the customer.

5. Conclusion

Based on a systematic review of existing literature, we identified the most important challenges that are likely to be faced by multidisciplinary PSS development teams. The publications that were reviewed address a broad variety of aspects and propose solutions for the basic challenges that are relevant in this context. These can be translated into design recommendations for PSS providers.

Acknowledgment

We thank the German Research Foundation (DFG) for funding this work as part of the collaborative research center ‘Sonderforschungsbereich 768 – Managing cycles in innovation processes – Integrated development of product-service-systems based on technical products’ (SFB768).

Publication 2

Analyse der Eignung domänenspezifischer Methoden der Anforderungsverfolgung für Produkt-Service-Systeme

Thomas Wolfenstetter^a, Sebastian Floerecke^b, Markus Böhm^a, Helmut Krcmar^a

^a Chair for Information Systems Technische Universität München Munich, Germany {thomas.wolfenstetter, markus.boehm, krcmar}@in.tum.de	^b Universität Passau, Lehrstuhl für Wirtschaftsinformatik II, Passau, Deutschland sebastian.floerecke@uni-passau.de
--	---

Abstract

Die Anforderungsverfolgung bringt bei der Entwicklung von Produkt-Service-Systemen (PSS) zahlreiche Herausforderungen mit sich. Gründe hierfür sind komplexe Schnittstellen zwischen den Domänen Produkt-, Software- und Dienstleistungsentwicklung, unterschiedliche Lebenszyklen und gegenseitige Beeinflussung einzelner Komponenten, ein hoher Grad an technischer Integration sowie eine kundenindividuelle Leistungserstellung. Verstärkt wird diese Komplexität dadurch, dass sich Anforderungen an das PSS entlang des gesamten Lebenszyklus ändern können. Während in der Literatur domänenspezifische Anforderungsverfolgungsmethoden zu finden sind, existiert bislang kein PSS-spezifischer Ansatz. Ziel dieses Beitrags ist daher, zu untersuchen, inwieweit sich diese Methoden für PSS eignen. Die Grundlage dafür bilden zehn, aus den Eigenschaften von PSS abgeleitete Kriterien. Die Analyse zeigt, dass keine der Methoden alle Kriterien vollständig erfüllt. Dennoch bieten einige bei der Verfolgung der Anforderungsherkunft, der Beziehung zwischen Anforderungen, der Anforderungsumsetzung sowie dem Versionsmanagement vielversprechende Ansätze. Diese Bewertung dient als Ausgangspunkt für eine gezielte Kombination und Erweiterung der Methoden, um eine adäquate Anforderungsverfolgung bei PSS zu ermöglichen.

Keywords: Produkt-Service-System, Anforderungsverfolgung, Anforderungsmanagement, Analyse, Literaturstudie

1. Ausgangssituation und Problemstellung

Um dem gestiegenen Wettbewerbsdruck entgegenzuwirken und sich von der Konkurrenz zu differenzieren, bieten Unternehmen unterschiedlichster Branchen vermehrt Komplettlösungen für individuelle Kundenprobleme an (Leimeister and Glauner 2008). Ein Beispiel hierfür sind Carsharing-Angebote zur Erfüllung eines Mobilitätsbedürfnisses. Bei derartigen Lösungen handelt es sich um integrierte Leistungsbündel, bestehend aus Hardware-, Software- und Dienstleistungskomponenten, die als Produkt-Service-Systeme (PSS) bezeichnet werden (Baines et al. 2007). Der Wertschöpfungsprozess im Kontext von PSS ist dabei auf eine dauerhafte und intensive Geschäftsbeziehung ausgerichtet, in welcher der Kunde nicht mehr länger nur als Wertschöpfungsempfänger, sondern als Wertschöpfungspartner auftritt (Tuli et al. 2007). Hierbei trägt der Anbieter häufig die Verantwortung für den gesamten Lebenszyklus eines PSS (Baines et al. 2007; Herzfeldt et al. 2012). Die Entwicklung eines PSS erfordert eine integrative Zusammenarbeit von Produkt-, Software- und Dienstleistungsentwicklung. Jede dieser Disziplinen entwickelt einzelne Komponenten, die sich gegenseitig beeinflussen, nahtlos zusammenwirken sollen aber verschiedenen Lebenszyklen unterworfen sind (Berkovich et al. 2011b; Krcmar 2010). Oberstes Ziel eines PSS ist die möglichst optimale Erfüllung eines individuellen Kundenbedürfnisses. Daneben müssen Anbieter, zumindest im Business-to-Business-Bereich, die Geschäftsprozesse ihrer Kunden verstehen, um das PSS in deren Wertschöpfung integrieren zu können (Böhmman and Krcmar 2007; Tuli et al. 2007). Aus Anbietersicht ist es daher entscheidend, sämtliche Anforderungen an die Lösung genau zu erheben, zu spezifizieren und deren Abhängigkeiten zu kennen (Berkovich et al. 2011c).

Entlang des gesamten PSS-Lebenszyklus kommt es jedoch fortlaufend zu Änderungen (Berkovich et al. 2011a; Herzfeldt et al. 2010), da dieser zyklischen Einflüssen unterworfen ist (Langer and Lindemann 2009). Beispiele für Zyklen sind die Änderung von Kundenwünschen und Gesetzen oder die Verfügbarkeit neuer Technologien (Berkovich et al. 2011c). Im Carsharing-Bereich etwa besteht eine zentrale Anforderung an den Dienstleistungsprozess darin, dass der Kunde vor Übernahme des Fahrzeugs dieses auf Vorschäden prüft und nicht erfasste Mängel meldet. Durch Einführung einer neuen Sensortechnologie könnten dagegen Schäden bereits bei Entstehung erfasst und weitergeleitet werden. Dies würde die ursprüngliche Anforderung überflüssig machen und die Nutzung vereinfachen, brächte aber Änderungen am Fahrzeug und der Software mit sich, welche wiederum weitere Änderungen anstoßen könnten. Zudem könnte diese Anpassung gegen gesetzliche Datenschutzregeln verstoßen und somit rechtliche Folgen haben. Im Rahmen des Anforderungsmanagements müssen PSS-Anbieter daher mögliche Änderungen antizipieren, die Auswirkungen auf weitere Anforderungen und Komponenten erkennen und Maßnahmen zum Umgang mit Änderungen anstoßen. Voraussetzung dafür ist, den Lebenszyklus einer Anforderung zu verfolgen und Abhängigkeiten zwischen Anforderungen und PSS-Komponenten zu identifizieren, diese abzubilden und zu pflegen (Berkovich et al. 2011b). Dieser Bereich des Anforderungsmanagements, dem besonders bei der PSS-Entwicklung eine

große Bedeutung zukommt, wird als Anforderungsverfolgung bezeichnet (Gotel and Finkelstein 1994; Ramesh and Jarke 2001).

Obwohl in der Literatur verschiedene, domänenspezifische Anforderungsverfolgungsmethoden, aus der Software- und Hardwareentwicklung, zu finden sind, existiert bislang kein domänenübergreifender Ansatz, der den Herausforderungen von PSS genügt (Berkovich et al. 2011b; Wolfenstetter et al. 2013). Dieser Beitrag untersucht daher, inwieweit sich diese domänen-spezifischen Methoden der Anforderungsverfolgung für die PSS-Entwicklung eignen. Der Analyse liegen zehn Kriterien zugrunde, die aus den speziellen Eigenschaften von PSS und den Aufgaben der Anforderungsverfolgung bei PSS hergeleitet wurden.

2. Grundlagen der Anforderungsverfolgung bei PSS

Die Verfolgung von Anforderungen stellt eine Querschnittsfunktion des Anforderungsmanagements dar, die während des kompletten Lebenszyklus eines Produkts, einer Dienstleistung oder eines PSS zu gewährleisten ist (Cheng and Atlee 2007; Ebert 2012). Der Lebenszyklus einer Anforderung wird dabei von ihrem Ursprung über alle Phasen der Entwicklung verfolgt und die Abhängigkeiten zwischen verschiedenen Artefakten, wie etwa Anforderungen, PSS-Komponenten oder Testfälle, dokumentiert und gepflegt (Gotel and Finkelstein 1994; Ramesh and Jarke 2001). Da, wie das Carsharing-Beispiel zeigt, beispielsweise Dienstleistungen das Design der Software oder der Hardware beeinflussen können, muss die Anforderungsverfolgung bei PSS domänenübergreifend durchgeführt werden (Berkovich et al. 2011c). Zur Verfolgung von Anforderungen werden semantische Abhängigkeiten, sogenannte „Trace Links“, dokumentiert. Diese Trace Links geben an, wie Artefakte zueinander in Beziehung stehen (Spanoudakis and Zisman 2005). Durch Navigieren entlang der Trace Links lässt sich unter anderem nachvollziehen, warum eine Anforderung spezifiziert wurde, durch welche PSS-Komponenten eine Anforderung erfüllt werden soll oder welche Testfälle ihrer Absicherung dienen.

Die Anforderungsverfolgung lässt sich in drei Dimensionen unterteilen: Pre-Traceability erfasst Trace Links von der Anforderungsquelle, beispielsweise Kunde oder Gesetzgeber (Pinheiro 2004), bis zu ihrer Spezifikation (Gotel and Finkelstein 1994). Post-Traceability dagegen umfasst Trace Links einer Anforderung von ihrer Spezifikation bis zu ihrer Umsetzung, etwa als Softwarecode oder physische Komponente (Gotel and Finkelstein 1994). Drittens können Beziehungen zwischen Anforderungen auf derselben oder unterschiedlichen Abstraktionsebenen verfolgt werden. Diese Dimension wird als Inter-Traceability bezeichnet und beinhaltet Überlappungen, Konkretisierungen oder auch Konflikte zwischen Anforderungen (Pohl 2010; Pinheiro 2004). Für PSS ist Inter-Traceability von besonderer Bedeutung, da die Anforderungsspezifikation ein komplexes, domänenübergreifendes und multihierarchisches Netzwerk aus interdependenten Anforderungen darstellt (Berkovich et al. 2011a). Figure 10 fasst die Dimensionen der Anforderungsverfolgung zusammen:

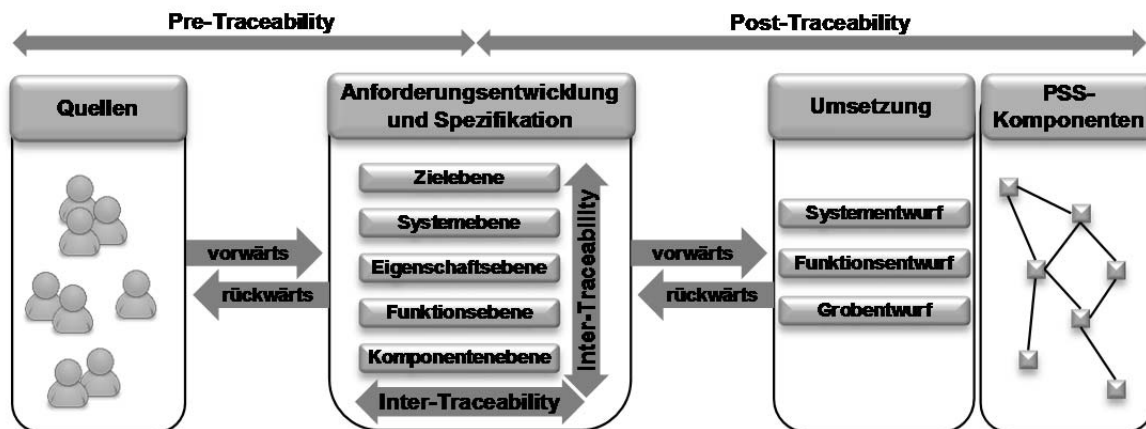


Figure 10: Dimensionen der Anforderungsverfolgung bei PSS
Quelle: In Anlehnung an (Gotel and Finkelstein 1994)

Durch eine systematische Anforderungsverfolgung lässt sich gezielt nachprüfen, ob und wie Anforderungen tatsächlich umgesetzt wurden. Des Weiteren können Komponenten oder Funktionen identifiziert werden, die keine Anforderung erfüllen und daher möglicherweise überflüssig sind (Kotonya and Sommerville 1998; Watkins and Neal 1994). Gleichzeitig wird das nachträgliche Einschleichen von Anforderungen in die Spezifikation verhindert, da für jede Anforderung die Quelle sowie der Grund deren Aufnahme nachvollziehbar ist (Kirova et al. 2008; Ebert 2012). Bei einer bevorstehenden Änderung lässt sich analysieren, welche Artefakte ebenfalls betroffen sind und gegebenenfalls angepasst werden müssen (Watkins and Neal 1994; Pohl 2008). Außerdem fördert die Anforderungsverfolgung die Wiederverwendung von Artefakten, da festgestellt werden kann, welches Artefakt verwendet werden könnte und welche Anpassungen für die Verwendung in einem anderen Kontext nötig sind (Kotonya and Sommerville 1998; Spanoudakis and Zisman 2005). Ferner werden die Überwachung des Projektfortschritts, die Ressourcenallokation sowie das Controlling unterstützt (Ebert 2012; Torkar et al. 2012). Zudem kann geprüft werden, welche Testfälle eine Anforderung verifizieren und ob jede Anforderung durch einen Test abgesichert wird (Kirova et al. 2008; Pohl 2008).

3. Forschungsdesign

Um verschiedene Anforderungsverfolgungsmethoden zu identifizieren, wurde eine Literaturstudie nach den Richtlinien von Webster und Watson [24] durchgeführt. Dabei wurde zunächst in den Publikationsdatenbanken Google Scholar, IEEE Xplore, ACM Digital Library, Springer Link und Emerald Insight nach relevanten Schlüsselwörtern wie „Requirements Traceability“, „Traceability“, „Tracing“ oder „Anforderungsverfolgung“ gesucht. Zudem wurden diese Begriffe in Verbindung mit Suchbegriffen wie „Product Development“, „Service“, „Product Service System“ oder „Hybrides Produkt“ geprüft. Anschließend bewerteten die Autoren getrennt voneinander die Relevanz der Veröffentlichungen, die in den Datenbanken jeweils unter den 500 besten Treffern lagen und mindestens einmal zitiert wurden. Im Fokus standen Publikationen, in denen Anforderungsverfolgungsmethoden beschrieben wurden. Grundlage der Publikationsauswahl

für die detaillierte Prüfung war der Durchschnitt der Relevanzbewertung von Titel und Abstract seitens der Autoren. Danach wurden die Literaturverzeichnisse der identifizierten Beiträge systematisch geprüft. Gleichzeitig wurde kontrolliert, welche Quellen wiederum die bisher betrachteten Beiträge zitieren. Dieses Vorgehen wurde iterativ wiederholt, bis keine weiteren Methoden ausfindig gemacht werden konnten. Insgesamt wurden nach der Streichung von Duplikaten sowie derjenigen Publikationen, in den die Anforderungsverfolgung nur eine untergeordnete Rolle spielt, 15 Methoden aus 118 Veröffentlichungen identifiziert.

Die identifizierten Anforderungsverfolgungsmethoden wurden hinsichtlich ihrer Eignung für PSS anhand von zehn Kriterien analysiert. Hergeleitet wurden die Kriterien aus den Eigenschaften von PSS und den Aufgabenbereichen der Anforderungsverfolgung im Kontext von PSS. Die Methodenbewertung erfolgte anhand einer drei-stufigen Skala, unabhängig voneinander durch die Autoren im Hinblick darauf, ob und in welchem Umfang die Kriterien in der Beschreibung der Methoden und deren Diskussion thematisiert wurden. Dabei ergab die Bewertung in 84,7 % der Fälle eine exakte Übereinstimmung (Krippendorffs Alpha: 0,709). Bei unterschiedlichen Bewertungen wurden gemeinsam Argumente abgewogen und eine Konsensentscheidung herbeigeführt. Voll erfüllt bedeutet in diesem Beitrag, dass höchstens nur leichte Erweiterungen für die Verwendung im PSS-Kontext nötig sind. Bei einer teilweisen Erfüllung wird zwar ein Kriterium in den Publikationen erwähnt, allerdings sind weitreichende Änderungen erforderlich. Wenn ein Kriterium überhaupt nicht thematisiert wurde, erfolgte eine Einstufung als nicht erfüllt.

4. Methoden der Anforderungsverfolgung

Im Rahmen der Literaturstudie konnten 15 Methoden der Anforderungsverfolgung identifiziert werden. Viele Methoden stammen dabei aus dem Umfeld der Softwareentwicklung. Ein möglicher Grund dafür ist, dass in diesem Bereich das Anforderungsmanagement innerhalb der Forschung bislang stärker als bei den Ingenieurwissenschaften adressiert wurde (Berkovich et al. 2011a). Obwohl explizit danach gesucht wurde, konnte keine spezifische Methode für Dienstleitungen gefunden werden. Dies kann damit begründet werden, dass die Dienstleistungsentwicklung ein relativ junges Forschungsfeld darstellt (Leimeister 2012) und dort oft Ansätze aus anderen Domänen adaptiert werden.

Insgesamt handelt es sich bei diesen Methoden um heterogene Ansätze, die sich auf unterschiedliche Aspekte der Anforderungsverfolgung konzentrieren. So legen einige Methoden den Fokus darauf, wie Trace Links identifiziert (z. B. Antoniol et al. 2002), dokumentiert oder gepflegt werden können (z. B. Cleland-Huang et al. 2002), während sich andere damit beschäftigen, wie diese Informationen visualisiert (z. B. Cleland-Huang 2005) oder wie Auswirkungen von Änderungen prognostiziert werden können. Konzeptuelle Ansätze hingegen beschreiben, welche Artefakte bei der Anforderungsverfolgung berücksichtigt werden müssen, welche Arten von Beziehungen existieren und welche Informationen über jedes Artefakt benötigt werden (z. B. Ramesh and Jarke 2001). Daneben gibt es Prozessmodelle, die definieren, welche Aktivitäten bei der Anforderungsverfolgung

durchzuführen sind. Die in diesem Beitrag als Methode verstandenen Ansätze beschreiben hingegen, wie diese Aktivitäten durchgeführt beziehungsweise unterstützt werden können. Die den einzelnen Methoden zugrundeliegenden Ideen werden nachfolgend kurz vorgestellt:

- (1) **Information Retrieval** basiert auf einem Ähnlichkeitsvergleich zweier Artefakte zur automatischen Identifikation möglicher Trace Links. Ein Artefakt fungiert als Anfrage und ein anderes als Dokument, das hinsichtlich der Anfrage durchsucht wird. So agiert beispielsweise der Softwarecode als Anfrage und die Anforderungsspezifikation als Dokument. Artefakte mit hohen Ähnlichkeitswerten gelten dabei als Kandidaten für Trace Links. Um Unterschiede zwischen verschiedenen Artefaktversionen zu erkennen, kann dieser Ansatz mehrfach angewandt werden (Antoniol et al. 2002).
- (2) Bei **Event-based-Traceability**, einem Architekturkonzept für Softwaretools, werden Trace Links als Publisher-Subscriber-Beziehungen abgebildet. Diese Methode setzt existierende Trace Links voraus. Sobald eine Änderung eintritt, werden davon betroffene Stellen benachrichtigt, um mögliche Auswirkungen zu analysieren (Cleland-Huang et al. 2002).
- (3) Der Grundgedanke von **Rule-based-Traceability** ist die Verwendung von XML-basierten Regeln zur automatischen Generierung von Nachvollziehbarkeitsinformationen. Dazu müssen neben den Regeln selbst, das Anwendungsfalldokument und das Analyseobjektmodell im XML-Format vorliegen (Zisman et al. 2002).
- (4) Während andere Methoden keine Unterscheidung bei den Anforderungen bezüglich ihrer Bedeutung für die Anforderungsverfolgung vornehmen, setzt das **Value-based-Traceability** genau an dieser Stelle an. Ziel ist es, besonders wichtige Anforderungen detaillierter zu verfolgen als solche mit niedriger Priorität. Dadurch ergeben sich Kostensenkungspotentiale bei der Anforderungsverfolgung (Heindl and Biffel 2005).
- (5) Der Ausgangspunkt von **Feature-Model-based-Traceability** ist, dass der Unterschied im Abstraktionsgrad und der Formalität zwischen Anforderungen und Features geringer als zwischen einer Anforderung und einem Lösungsartefakt ist. Ein Feature beschreibt eine Produkteigenschaft aus Sicht des Kunden und stellt das Bindeglied zwischen Anforderungen und Lösungsartefakten dar. Die Trace Links verlaufen somit von den Anforderungen zu den Features bis hin zu den Lösungselementen (Riebisch 2004).
- (6) Das Ziel des **Feature-oriented-Traceability** ist das Identifizieren von Nachvollziehbarkeitsinformationen basierend auf priorisierten Anforderungen hinsichtlich Kosten und Aufwand. Daran lässt sich erkennen, dass dieses Verfahren eine Erweiterung des Value-based-Traceability darstellt (Ahn and Chong 2006).
- (7) **Scenario-based-Traceability** gleicht statische Informationen aus Entwicklungsmodellen mit dynamischen Informationen aus dem aktuell implementierten Softwareprodukt ab. Dazu wird das Verhalten der Software mit Testszenarios, die während der Entwicklung definiert wurden, beobachtet. Da die meisten Beobachtungen direkt mit Szenarien korrespondieren, können auf diese Weise Trace Links zwischen Szenarien und dem System identifiziert werden (Egyed 2001).
- (8) **Hypertext-based-Traceability** verwendet ein Hypertextmodell, welches das komplexe Verbinden und Versionieren von Nachvollziehbarkeitsbeziehungen ermöglicht. Zur Generierung der Trace Links wird das Information Retrieval eingesetzt. Es nutzt XML

- als Datenformat zur Repräsentation der Modelle und der generierten Trace Links. Diese müssen daher in einer XML-Darstellung vorliegen (Maletic et al. 2003).
- (9) **Goal-centric-Traceability** zielt darauf ab, nicht-funktionale Anforderungen zu verfolgen. Nicht-funktionale Anforderungen, wie Zuverlässigkeit, Sicherheit oder Wartbarkeit, gelten als besonders anspruchsvoll zu verfolgen, da sie Auswirkungen auf das System als Ganzes haben und zwischen ihnen zahlreiche Abhängigkeiten und Zielkonflikte bestehen (Cleland-Huang et al. 2005).
 - (10) Der Ausgangspunkt von **Pre-Requirements Specification Traceability** ist die Annahme, dass Pre-Traceability im Vergleich zu Post-Traceability deutlich schwieriger ist, da Trace Links zwischen dem Problemraum – den Bedürfnissen – und dem Lösungsraum – der Anforderungsspezifikation – etabliert werden müssen. Da die Distanz zwischen beiden Räumen groß ist, wird ein Übergangsraum eingefügt, um diese Komplexitätslücke zu reduzieren (Ravichandar et al. 2007).
 - (11) Einen weiteren Ansatz, der speziell zur Unterstützung des Pre-Traceability entwickelt wurde, stellen **Contribution Structures** dar. Dabei werden Trace Links zwischen Anforderungen und Stakeholdern generiert. Beispielsweise wird die Anforderungsquelle oder die Verantwortlichkeit für die Umsetzung einer Anforderung dokumentiert (Gotel and Finkelstein 1995).
 - (12) Bei **Traceability-Matrizen**, wie Design-Structure- oder Domain-Mapping-Matrizen, müssen die Trace Links manuell erzeugt werden. Sie werden häufig in der Praxis verwendet, um sowohl Beziehungen zwischen Anforderungen untereinander als auch zwischen Anforderungen und anderen Entwicklungsartefakten, wie Testfällen, Codemodulen und dem Design, zu etablieren und visualisieren (Cleland-Huang 2005).
 - (13) Das **Reference Model for Requirements Traceability** definiert bestimmte Typen von Artefakten und Trace Links. Dabei wird zwischen verschiedenen Detaillierungsstufen der Anforderungsverfolgung unterschieden (Ramesh and Jarke 2001).
 - (14) Beim **Quality Function Deployment** handelt es sich um einen strukturierten Ansatz zur Übersetzung von Kundenanforderungen in Designziele sowie zur Analyse der Abhängigkeiten zwischen Komponenten. Damit lässt sich darstellen, welche Kundenanforderung durch welche Produktkomponente umgesetzt wird (Akao 1990).
 - (15) Die **Fehlermöglichkeits- und Einflussanalyse** wird eingesetzt, um eine möglichst fehlerfreie Gestaltung von Produkten und Prozessen unter Einhaltung aller Kunden- und Qualitätsanforderungen zu erzielen. Sie basiert auf dem Prinzip der vorausschauenden Fehlervermeidung. Wie auch beim Quality Function Deployment lässt sich abbilden, welche Komponente welche Anforderung erfüllt (Teng and Ho 1996).

(16)

5. Bewertungskriterien

Der Methodenbewertung liegen folgende zehn Kriterien zugrunde, die anhand der Literaturstudie aus den Eigenschaften von PSS und den Aufgabenbereichen der Anforderungsverfolgung im Kontext von PSS hergeleitet wurden:

- (1) Verfolgung der Anforderungsherkunft: Ist dieses Kriterium erfüllt, so kann unter anderem nachvollzogen werden, warum eine Anforderung spezifiziert wurde und wer welche Verantwortung trägt (Pinheiro 2004; Maeder et al. 2006). Allerdings wird häufig darauf hingewiesen, dass Pre-Traceability in der Praxis die größte Herausforderung darstellt und deshalb auch nur selten angewendet wird (Gotel and Finkelstein 1994; Ravichandar et al. 2007). Im Kontext von PSS ist Pre-Traceability besonders wichtig und zugleich schwierig, da meist viele, heterogene Interessensgruppen mit unterschiedlichen Anforderungen berücksichtigt werden müssen (Berkovich et al. 2011a).
- (2) Verfolgung der Beziehung zwischen Anforderungen: Bei der Detaillierung von Anforderungen an ein PSS werden abstrakte Geschäftsziele iterativ heruntergebrochen bis konkrete, domänenspezifische Anforderungen an jede PSS-Komponente spezifiziert werden können. Die strukturierte Abbildung der Beziehungen zwischen Anforderungen an Sach- und Dienstleistungskomponenten eines PSS dient somit der Koordination der verschiedenen Domänen (Berkovich et al. 2011a). Bei der Anforderungsverfolgung für PSS muss daher berücksichtigt werden, dass mehrere Abstraktionsebenen, von Geschäftszielen bis hin zu detaillierten Komponentenanforderungen, existieren (Wolfenstetter et al. 2013). Anforderungen können hierbei in Konflikt zueinanderstehen, sich gegenseitig einschränken oder erweitern (Pohl 2008; Winkler and Pilgrim 2010). Durch die Erfassung dieser Beziehungen kann somit die Anforderungsbasis strukturiert werden (Wolfenstetter et al. 2013; Pinheiro 2004).
- (3) Verfolgung der Anforderungsumsetzung: Dieses Kriterium umfasst Trace Links einer Anforderung von ihrer Spezifikation bis zur Umsetzung durch eine Lösungskomponente des PSS (Gotel and Finkelstein 1994). Zu jedem Zeitpunkt der Entwicklung muss erkennbar sein, zu welchem Grad Anforderungen umgesetzt oder umsetzbar sind (Sahraoui 2005). Außerdem sollen Komponenten oder Funktionen identifiziert werden können, die keine Anforderung erfüllen und daher möglicherweise überflüssig sind (Ramesh and Jarke 2001; Kirova et al. 2008).
- (4) Versionsmanagement: Da Artefakte bei der Entwicklung von PSS einer ständigen Veränderung unterworfen sind, muss eine Methode zur Anforderungsverfolgung in der Lage sein, verschiedene Versionen eines Artefakts abzubilden und miteinander zu verknüpfen (Spanoudakis and Zisman 2005; Berkovich et al. 2011a). Nur anhand dieser Evolutionskette kann nachvollzogen werden, welche Änderungen aus welchem Grund und zu welchem Zeitpunkt vorgenommen wurden. Unterschiedliche Versionen müssen dabei dokumentiert und gegebenenfalls auch wiederhergestellt werden können (Pinheiro 2004).

- (5) Varianten- und Konfigurationsmanagement: PSS zielen meist auf die Lösung eines spezifischen Kundenproblems ab und müssen daher individuell gestaltet werden. Bei einer unsystematischen Individualisierung und Variantenbildung ist es für den PSS-Anbieter jedoch schwierig, profitabel zu wirtschaften (Sawhney 2006; Galbraith 2002). Um gewisse Komponenten wiederverwenden zu können und damit Kosten zu sparen, wird das Konzept der Modularisierung eingesetzt. Dadurch können individuelle Kundenanforderungen oft durch die Kombination bestehender Module abgedeckt werden (Böhmman et al. 2008). Die Anforderungsverfolgung sollte dabei aufzeigen, wie verschiedene Komponenten kombiniert werden müssen, um sämtliche Anforderungen zu erfüllen (Wolfenstetter et al. 2013). Hierdurch ergeben sich zusätzliche Herausforderungen, da sich die Menge der Informationen und die Komplexität, Änderungen zu managen, erhöht (Mohan and Ramesh 2006).
- (6) Integration des Wertschöpfungsnetzwerks: Nachdem PSS oftmals aus vielen Komponenten bestehen, ist es für Anbieter zumeist nicht möglich, sämtliche Bestandteile selbst zu entwickeln oder anzubieten (Gebauer et al. 2013). Auch wenn sie in der Regel alleine die Geschäftsbeziehung mit ihren Kunden unterhalten, sind PSS-Anbieter von der Qualität ihrer Modullieferanten abhängig (Reichwald et al. 2009). Daher müssen Anbieter die Lieferanten und Partner aus ihrem Wertschöpfungsnetzwerk in die Entwicklung einbinden. Für die Anforderungsverfolgung bedeutet dies, dass sie bei PSS ebenfalls unternehmensübergreifend durchführbar sein sollte. So muss Partnern der aktuelle Stand, der für sie relevanten Teilmenge, der Anforderungsbasis zugänglich gemacht werden können.
- (7) Simultane Entwicklung und unterschiedliche Sichten: Entwicklungsaufgaben werden häufig so aufgeteilt, dass Teams gleichzeitig und weitgehend autonom an bestimmten Teilaufgaben arbeiten können (Sharafi et al. 2010b). Dies sollte durch die Anforderungsverfolgung unterstützt werden. Daneben muss es möglich sein, verschiedene Sichten auf das Anforderungsmodell und das gesamte PSS-Modell zu definieren. Dabei sollten die verschiedenen Rollen, wie Projektmanager oder Entwickler, gezielt mit Informationen versorgt werden können, die sie auch tatsächlich benötigen (Berkovich et al. 2010b).
- (8) Robustheit gegenüber Unsicherheiten: Artefakte, wie Designmodelle oder Anforderungen, zeichnen sich oftmals durch ungenaue oder sogar gänzlich fehlende Daten aus. Diese Unsicherheit stammt aus dem Umfeld des Entwicklungsprojekts, der Entwicklungsarbeit selbst oder der Projektdefinition (Ebert and Man). Unsicherheit tritt zum Beispiel dann auf, wenn die Dauer eines Dienstleistungsprozesses nur ungenau bestimmt werden kann. Bei der Anforderungsverfolgung im Kontext von PSS sollten deshalb derartige Unsicherheiten und Informationslücken adressiert werden können (Wolfenstetter et al. 2013).

- (9) Berücksichtigung des gesamten PSS-Lebenszyklus: PSS-Anbieter sind meist für den kompletten Lebenszyklus verantwortlich (Baines et al. 2007; Herzfeldt et al. 2012). Auch die Anforderungsverfolgung sollte daher den gesamten Lebenszyklus erfassen. Dabei muss ein breites Spektrum von Lösungsartefakten berücksichtigt werden. So muss etwa abgebildet werden, wie Hardwarekomponenten hergestellt und instandgehalten, wie Dienstleistungen erbracht oder wie Softwaremodule aktualisiert werden. Oft können Erfahrungen aus späten Lebenszyklusphasen Anforderungen an die nächste Generation eines PSS aufzeigen. Dazu muss aber der Anforderungslebenszyklus nachvollziehbar sein (Knethen et al. 2002).
- (10) Reduzierung des Aufwands: Da allgemein die Erfassung sämtlicher Anforderungsverfolgungsinformationen mit einem hohen Aufwand verbunden ist und dabei nicht jede Anforderung gleich wichtig ist, wäre eine vollständige, aber unsystematische Anforderungsverfolgung in vielen Fällen unwirtschaftlich (Jarke 1998; Ebert 2012). Dies gilt insbesondere für PSS, die oftmals durch eine hohe Anzahl und einer hohen Verflechtung von Anforderungen gekennzeichnet sind (Wolfenstetter et al. 2013). Daher sollte es möglich sein, kritische Anforderungen zu identifizieren und genauer zu verfolgen als unkritische.

6. Bewertung der Anforderungsverfolgungsmethoden

Die Bewertung der 15 Methoden zeigt, dass keine der untersuchten Methoden alle Kriterien erfüllt. Dennoch werden bereits, wie Table 7 verdeutlicht, alle der für PSS relevanten Kriterien von den Methoden, zumindest teilweise, abgedeckt.

Table 7: Zusammenfassung der Methodenbewertung

Kriterium Methode	(1) Verfolgung der Anforderungsherkunft	(2) Verfolgung der Beziehung zwischen Anforderungen	(3) Verfolgung der Anforderungsumsetzung	(4) Versionsmanagement	(5) Varianten- und Konfigurationsmanagement	(6) Integration des Wertschöpfungsnetzwerks	(7) Simultane Entwicklung und unterschiedliche Sichten	(8) Robustheit gegenüber Unsicherheiten	(9) Berücksichtigung des gesamten PSS-Lebenszyklus	(10) Reduzierung des Aufwands
Information Retrieval (IR)	-	+	+	+	-	-	-	-	-	-
Event-based-Traceability (EBT)	-	+	+	+	-	+	+	-	-	-
Rule-based-Traceability (RBT)	-	+	+	+	+	-	-	-	-	-
Value-based-Traceability (VBT)	-	-	+	-	-	-	-	-	-	+
Feature-Model-based-Traceability (FMT)	-	-	+	+	+	-	-	+	-	-
Feature-oriented-Traceability (FOT)	-	-	+	-	+	-	-	-	-	+
Scenario-based-Traceability (SBT)	-	-	+	-	-	-	-	-	-	-
Hypertext-based-Traceability (HBT)	-	+	+	+	-	-	-	-	-	-
Goal-centric-Traceability (GCT)	-	+	+	+	-	-	-	-	-	-
Pre-Requirements Specification Traceability (PRST)	+	+	-	-	-	-	-	-	-	-
Contribution Structures (CS)	+	+	-	-	-	-	-	-	-	-
Traceability-Matrizen (TM)	-	+	+	-	-	-	-	-	-	-
Reference Models for Requirements Traceability (RMRT)	+	+	+	+	-	-	-	-	-	+
Quality Function Deployment (QFD)	-	-	+	-	+	-	-	-	-	-
Fehlermöglichkeits- und Einflussanalyse (FMEA)	-	-	+	-	+	-	-	-	-	-
Anzahl positiver Bewertungen	3	9	13	7	5	1	1	1	0	3

Das erste Kriterium, (1) Verfolgung der Anforderungsherkunft, wird nur von wenigen Techniken abgedeckt. Allerdings können die Ansätze der positiv bewerteten Methoden auf PSS übertragen werden, da Beziehungen zwischen der Anforderungsquelle und der Spezifikation als weitestgehend domänenneutral angesehen werden können.

Die (2) Verfolgung der Beziehung zwischen Anforderungen berücksichtigen zehn der 15 Methoden, zumindest aus ihrer domänenspezifischen Perspektive heraus. Für die Strukturierung von Anforderungen an PSS wurde durch ein Artefaktmodell (Berkovich et al. 2011a) bereits ein erster, domänenübergreifender Ansatz vorgeschlagen. Dieser unterstützt durch die Aufteilung von Anforderungen in mehreren Abstraktionsebenen die interdisziplinäre Erhebung und die Konkretisierung der Anforderungen. Durch eine Erweiterung dieses Modells um verschiedene Arten von Trace Links kann es für die Verfolgung der Beziehung zwischen Anforderungen eingesetzt werden.

Die (3) Verfolgung der Anforderungsumsetzung steht bei den meisten der Methoden im Mittelpunkt. Jedoch werden auch hier nur domänenspezifische Artefakte betrachtet. Bei der Entwicklung von PSS müssen allerdings Beziehungen zwischen heterogenen Artefakten in allen beteiligten Domänen abgebildet werden können. Beispielsweise muss analysierbar sein, wie sich eine Änderung im Konstruktionsentwurf für ein physisches Produkt auf die Steuerungssoftware auswirkt. Insgesamt bilden die Verfahren eine geeignete Basis, die für PSS allerdings erweitert werden müsste.

Das (4) Versionsmanagement nimmt in etwa bei der Hälfte der Methoden, jedoch ausschließlich im Softwarebereich, einen hohen Stellenwert ein. Für die Anforderungsverfolgung im Kontext von PSS ist das Managen von Änderungen und somit der Umgang mit verschiedenen Versionen der Artefakte in Anbetracht der zyklischen Wechselwirkungen von enormer Wichtigkeit. Für die Anforderungsverfolgung bei der PSS-Entwicklung gilt es deshalb zu prüfen, wie die Versionsverwaltung in einer domänenübergreifenden Umgebung aussehen sollte.

Der Bereich (5) Varianten- und Konfigurationsmanagement stand bisher innerhalb der Forschung zur Anforderungsverfolgung weniger im Fokus. Wie aus der Analyse hervorgeht, unterstützen zwar einige Methoden das Kriterium, aber bezogen auf PSS, nur im weiteren Sinne. Dabei ist die Modularisierung die Voraussetzung, um als PSS-Anbieter profitabel arbeiten zu können (Sawhney 2006). Es ist davon auszugehen, dass das Anbieten unterschiedlicher Varianten, basierend auf einer modularen Architektur, die Komplexität bei der Anforderungsverfolgung erheblich steigert (Mohan and Ramesh 2006).

Dass die Entwicklung und die Erbringung von PSS oft innerhalb eines (6) Wertschöpfungsnetzwerks erfolgt und dies daher ebenfalls von der Anforderungsmethode unterstützt werden sollte, geht aus den untersuchten Publikationen nur unzureichend hervor. Denkbar wäre, Event-based-Traceability unternehmensübergreifend einzusetzen. Bei dem eng damit verwandten Kriterium der (7) simultanen Entwicklung zeigt sich wiederum, dass nur diese Methode einen zufriedenstellenden Ansatzpunkt liefert. Diese Forschungslücke der simultanen Entwicklung ist bereits bekannt und gilt als einer der größten Schwachstellen der existierenden Methoden (Winkler and Pilgrim 2010).

Bezüglich der (8) Robustheit gegenüber Unsicherheiten wird ersichtlich, dass lediglich drei Methoden in gewisser Weise mit Informationslücken umgehen können. Ein PSS-spezifischer Ansatz sollte im Idealfall in der Lage sein, Informationslücken auszugleichen und Unsicherheiten bei der Analyse von Änderungsauswirkungen zu berücksichtigen. Davon scheint die Anforderungsverfolgung aber weit entfernt zu sein. Bevor allerdings dieses Spezialkriterium angegangen wird, sollten zunächst Lösungen für grundlegendere Probleme erarbeitet werden.

Die untersuchten Methoden beschränken sich meist auf spezielle Phasen der Entwicklung oder sind nur in den Lebenszyklus einer domänenspezifischen Komponente abzubilden. Da jedoch bei PSS unterschiedliche Lebenszyklen von Komponenten in einer integrierten, domänenübergreifenden Lösung aufeinandertreffen, deckt keine der betrachteten Methoden den (9) kompletten PSS-Lebenszyklus vollständig ab. Die Methoden müssen für einen Einsatz im PSS-Kontext in dieser Richtung erweitert werden, da die Verantwortung für den gesamten Lebenszyklus zu den elementarsten Anforderungen der PSS-Entwicklung zählt (Baines et al. 2007). Grund für diese Lücke ist, dass der Verantwortungsbereich innerhalb des Produktlebenszyklus bei einem klassischen Produkthersteller zumeist deutlich früher als bei einem PSS-Anbieter endet (Reichwald et al. 2009).

Das letzte betrachtete Kriterium, (10) Reduzierung des Aufwands, erfüllen nur zwei der untersuchten Ansätze vollständig. Sie versuchen einen Kompromiss dahingehend zu finden, ausschließlich die kritischen Anforderungen einzubeziehen und wichtige Anforderungen detaillierter zu verfolgen, um die anfallenden Kosten zu senken. Die geringe Anzahl an unterstützenden Methoden ist erstaunlich, da nach (Torkar et al. 2012; Egyed et al. 2005) zahlreiche Praktiker den mit der Anforderungsverfolgung verbundenen Aufwand beklagen.

7. Diskussion

Insgesamt gesehen ist keine der Methoden uneingeschränkt für die PSS-Entwicklung geeignet. Gleichzeitig kann aber auch kein Verfahren als besonders nachteilig, bezogen auf PSS, ausgewiesen werden. Schließlich unterstützen sie alle auf ihre Art einen Teilaspekt der Anforderungsverfolgung. Insbesondere weisen einige der Methoden bei den Kriterien (1) Verfolgung der Anforderungsherkunft, (2) Verfolgung der Beziehung zwischen Anforderungen, (3) Verfolgung der Anforderungsumsetzung und (4) Versionsmanagement bereits vielversprechende Ansätze auf. Allerdings muss an dieser Stelle betont werden, dass die betrachteten Methoden nicht direkt miteinander vergleichbar sind, da jede einen etwas anderen Fokus aufweist. Es zeigt sich aber, dass jedes Kriterium durch mindestens eine Methode, zumindest teilweise, erfüllt wird. Somit ergibt sich ein großes Potential für eine gezielte Kombination und Erweiterung der Methoden in Bezug auf die Herausforderungen der PSS-Entwicklung.

Für die Forschung ergeben sich neben der aufwendigen Neuerstellung eines PSS-spezifischen Ansatzes zwei Möglichkeiten: Entweder wird versucht, bestehende Verfahren tiefgreifend zu erweitern oder sie in geeigneter Weise zu kombinieren. Dies wirft die Frage auf, wie bestimmte Verfahren kombiniert werden können, damit sie die Anforderungen von ihrer

Quelle bis zu ihrer Umsetzung verfolgen und gleichzeitig Informationen aus späteren Lebenszyklusphasen einbinden, um diese bei der Weiterentwicklung des PSS nutzen zu können. In diesem Zusammenhang ist es sinnvoll, die Anforderungsverfolgung als Prozess mit den Aufgaben (1) Identifikation und Dokumentation der relevanten Informationen und Trace Links, (2) die fortlaufende Aktualisierung dieser und (3) die Nutzung der Informationen und Erkenntnisse daraus, beispielsweise im Änderungsmanagement, zu betrachten. Bezogen auf diese Phasen sollten die untersuchten Methoden in geeigneter Weise kombiniert und erweitert werden. So wäre es denkbar, mit Information Retrieval mögliche Trace Links zu identifizieren und Event-based-Traceability einzusetzen, um diese zu pflegen und Entwickler über sie betreffende Änderungen zu informieren. Inwieweit diese Strategie allerdings umsetzbar ist, sollte in einer weiteren Forschungsarbeit genauer untersucht werden, da eine unstrukturierte, nicht-integrierte, gleichzeitige Verwendung mehrerer Verfahren zu Überlappungen und redundanter Arbeit führen kann.

8. Zusammenfassung und Ausblick

In diesem Beitrag wurde der Frage nachgegangen, inwieweit sich die existierenden domänenspezifischen Anforderungsverfolgungsmethoden für den Einsatz bei PSS eignen. Dieser Analyse lagen zehn Kriterien zugrunde, anhand derer die Anforderungsverfolgungsmethoden im Kontext von PSS bewertet wurden. Bei dieser Bewertung wurde deutlich, dass keine Anforderungsverfolgungsmethode alle der betrachteten Kriterien erfüllt. Dennoch zeigen einige bei den Kriterien (1) Verfolgung der Anforderungsherkunft, (2) Verfolgung der Beziehung zwischen Anforderungen, (3) Verfolgung der Anforderungsumsetzung und (4) Versionsmanagement vielversprechende Ansätze. Insgesamt erscheint es sinnvoll und auch möglich, die Methoden so zu kombinieren und zu erweitern, dass alle geforderten Kriterien hinreichend erfüllt werden.

Um Entwickler von PSS in der Praxis bei der Anforderungsverfolgung zu unterstützen, bedarf es dreier Bausteine: Erstens ist ein Datenmodell erforderlich, das spezifiziert, welche Artefakte bei der Anforderungsverfolgung berücksichtigt werden müssen und welche Arten von Abhängigkeiten, Trace Links, zwischen diesen bestehen. Ein solches Datenmodell könnte beispielsweise als Ontologie realisiert werden, welche die semantischen Beziehungen zwischen verschiedenen Artefakten definiert. Dies sollte jedoch so gestaltet sein, dass es, beispielsweise hinsichtlich des Detaillierungsgrads, für unternehmens- beziehungsweise projektspezifische Zwecke angepasst werden kann. Daneben wird ein Prozessmodell benötigt, das darlegt, welche Aktivitäten in welchen Phasen des Entwicklungsprozesses in Unternehmen stattfinden müssen, um die Trace Links und weitere Informationen, die im Kontext der Anforderungsverfolgung wichtig sind, zu dokumentieren, zu pflegen und zu nutzen. Abgerundet werden sollte dies durch eine Art Methodenbaukasten, der in Abhängigkeit verschiedener Faktoren, wie dem Projektkontext oder der Art des PSS, aufzeigt, welche Methoden bei der Anforderungsverfolgung angewandt werden sollten.

Danksagung

Diese Veröffentlichung entstand im Rahmen des Sonderforschungsbereichs 768 „Zyklusmanagement von Innovationsprozessen – Verzahnte Entwicklung von Leistungsbündeln auf Basis technischer Produkte“. Das Forschungsvorhaben wird gefördert durch die Deutsche Forschungsgemeinschaft (DFG).

Publication 3

Towards Cycle-Oriented Traceability in Engineering Change Management

Nepomuk Chucholowski^b, Thomas Wolfenstetter^a, Martina Wickel^b, Helmut Krcmar^a, Udo Lindemann^b

^a Chair for Information Systems Technische Universität München Munich, Germany {thomas.wolfenstetter, markus.boehm, krcmar}@in.tum.de	^b Institute for Product Development, Technische Universität München Munich, Germany {nepomuk.chucholowski, martina.wickel, udo.lindemann}@pe.mw.tum.de
--	---

Abstract

Engineering changes and requirement changes strongly interfere with each other. Traceability helps to formalize this interface on a process and organizational level. We propose a data model that includes information elements for different change stages and all related requirement artifacts, solution artifacts or production artifacts. It facilitates the access to necessary information about relations or dependencies between these artifacts. Hence, change effects can easily be estimated and relevant people/information for the decision about a change and its implementation can be identified.

Keywords: engineering change management, requirements management, traceability

1. Introduction

Dynamic markets, regulatory environments and other external and internal cyclic influence factors force companies to make changes and adaptations to their products (product as a general term also standing for product service systems, PSS) (Langer et al. 2011). For instance, revised laws, emerging technologies, changing market needs, rising competition, errors or uncertainties during development lead to target deviations. That means the actual state of a product does not meet the desired nominal state anymore. Hence, companies need to adapt their products, processes and production continuously among the product lifecycle (Huang and Mak 1999). Those processes of adaptation are addressed by the concept of engineering change management (ECM) (Huang et al. 2001) and are interpreted as internal cycles within product development (Langer and Lindemann 2009).

These engineering changes (ECs) not only affect the product and its design but also other related artifacts such as production processes (Huang and Mak 1999). While changed requirements are commonly seen as possible triggers for engineering changes, the effect of an engineering change on other requirements is considered only by little literature. However, changing one part of a product can have unforeseen effects on the fulfilment of requirements which are not directly related to that part. Changed requirements are handled by requirements management. In this work we examine how an integration of engineering change management and requirements management could look like and how companies can benefit from traceability in their development cycles.

The following section describes our research methodology before section 3 gives an overview on requirements management and engineering change management, ending with the description of issues on the interface between the two topics. In section 4 we then present implications for the interface before the approach towards traceability in engineering change management is described in section 5. Section 6 concludes the paper and gives an outlook for future research.

2. Research Methodology

In order to analyze the current state of the art in research on how engineering changes and requirement changes are considered integrally we conducted a systematic literature review about engineering change management and requirements management. We started our analysis by initially investigating already known publications (Pohl 2010; Jarratt et al. 2011; Lindemann and Reichwald 2013; Wiegers 2009), that give a good overview about the respective topics. In order to find publications which directly associate engineering change management and requirements management, we additionally searched in online literature databases. As recommended by (Webster and Watson 2002) we further conducted a backward and forward search within the publications initially selected. In total we selected 95 publications that form the sample for our analysis. For this set of publications, we performed a qualitative content analysis in order to draw a summarizing picture of the state of the art in engineering change management and requirements management.

Additionally to the analysis of literature, we got insights into current practice through discussions with practitioners who are part of our industrial focus group on engineering change management.

In literature and in the focus group discussions, we particularly investigated how engineering change management and requirements management interface with each other. Altogether we want to answer the following research questions:

1. Which knowledge items are produced and required in engineering change management and requirements management?
2. How do cycles in engineering change management and requirements management interface with each other in terms of process management and organization?
3. How can traceability among different development artifacts contribute to managing engineering changes?

3. Overview: Requirements Management and Engineering Change Management

3.1. Requirements Management and Requirements Engineering

In the early phases of product development, the problem needs to be stated in a form that can be understood by engineers and used to find a solution. The part of product development that is concerned with defining the problem domain is commonly known as requirements engineering (RE). When talking about RE it is important to clear out what is meant by the term requirement. Requirements describe qualitative and/or quantitative properties or conditions for a product (Ehrlenspiel 2009). A more detailed definition of the term requirement can be found in the IEEE standards: “Requirements are statements of what the system must do, how it must behave, the properties it must exhibit, the qualities it must possess, and the constraints that the system and its development must satisfy” (Radatz et al. 1990). These requirements not only stem from customers but also from other stakeholders like the engineers developing the product (Pohl 2010).

Leffingwell and Widrig (2003) do not distinguish between the terms RE and requirements management (RM) and describe it as the eliciting, organizing, and documenting of the requirements of the system in a systematic way. It further aims to establish and maintain agreement between customers and the project team on the changing requirements of the system. As illustrated in Figure 11, requirements engineering can also be divided into requirements development and requirements management (Berkovich et al. 2009). Since we focus on requirement changes after they already have been developed, we further use this definition of RM which is described in the following.

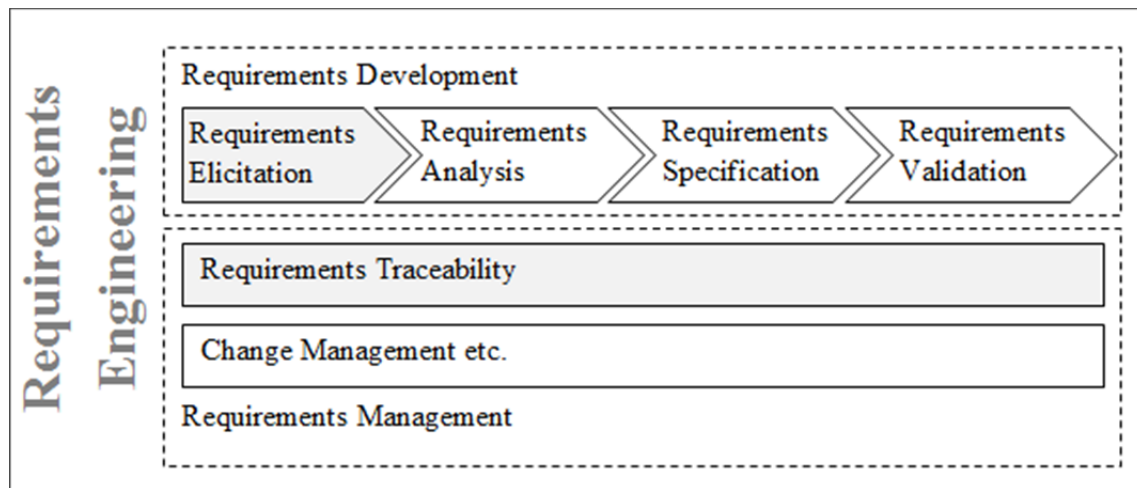


Figure 11: Considered activities of RE

Source: Based on (Berkovich et al. 2009)

The management of requirement changes (RCs) and the traceability of requirements are the two main tasks of requirements management. For instance, when a customer desires a new or changed functionality or product property a requirement is addressed and a potential change is identified. After identifying the encountered situation, a change request leads to several activities in the change management process: Create change request, determine attainability, plan, implement and evaluate changes. Each activity consists of several sub-activities to ensure customers', stakeholders' and producers' requirements. This step is supported by requirements traceability, which deals with the life cycle of a requirement (traceability will be explained in more detail in section 5.1.). Changes in requirements are continuous and inevitable, because - while the product is being developed - customer needs evolve, competitors introduce products and processes that help to give them a competitive advantage, and political, organizational and technical environments change.

In general, changes are noticeable in two ways. Either existing requirements change or new requirements emerge after the requirements specification has been considered complete. Requirements usually change with a monthly rate up to 5%, normalized to the total project effort. Over the entire project progress requirements vary between 30% and 60% of the initial requirements analysis (Ebert 2012). New or changed requirements during the project period must follow the same process and procedures as original requirements. Any change in requirements must be analyzed, evaluated and decided. For a controlled change of requirements in running projects a fixed change process must be defined (Versteegen 2003:). Any change must be agreed through a specified instance. If changes are accepted or received in an uncontrolled manner, this might lead to even more requirements or a constantly re-tuning of previously recorded requirements by stakeholders. As a result, there will be additional costs and delays in the project process. The setting of deadlines for the adoption of requirements and a transparent communication of the effects of changes are possible solutions.

3.2. Engineering Change Management

The definitions of the term engineering change (EC) slightly differ in literature. While in the past ECs often only referred to modifications made to product components that were already in production [e.g. (Wright 1997)], nowadays any alteration to released parts, documents or software during the design process is considered as an EC (Jarratt et al. 2011). The handling of these changes is called engineering change management (ECM) (Jarratt and Clarkson 2005). Other authors additionally see the documentation of all impacted product data (Rouibah and Caskey 2003), the documentation of the history of all changes of products and its associated documents (Huang and Mak 1999), or also the avoidance and anticipation of ECs (Lindemann and Reichwald 2013) as part of ECM. This shows that there are different perspectives on ECM in literature. Jarratt, et al. (Jarratt et al. 2011) categorize the EC literature by a process perspective, tool perspective and product perspective. This categorization neglects the view on the different strategies regarding all perspectives and the fact that in some literature the documentation is reckoned as the main objective of ECM. Hence, we differentiate between a process, documentation and strategic perspective in the following. For a more detailed categorization we refer to (Hamraz et al. 2013) who performed a comprehensive literature review and categorized 427 ECM publications.

From a process perspective, ECM is the processing of engineering changes, i.e. starting with an EC request and finishing with its successful implementation (or its disapproval before) [e.g. (Jarratt and Clarkson 2005)] and providing respective methods and tools within the process steps. Jarratt and Clarkson (2005) suggest a generic change process in six basic steps shown in Figure 2.

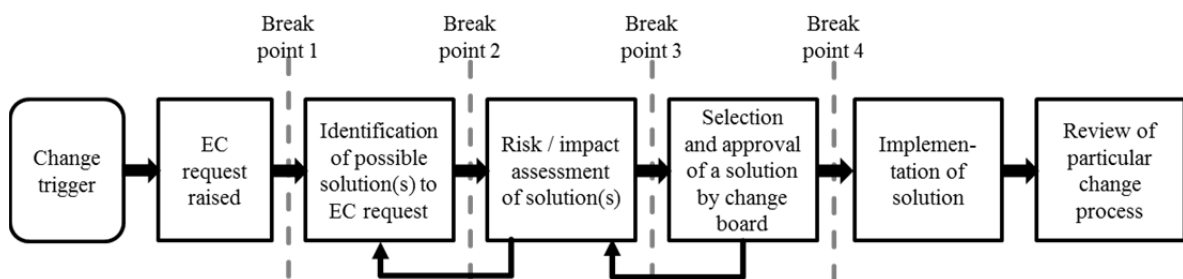


Figure 12: Generic engineering change process

Source: Based on (Jarratt and Clarkson 2005)

ECM also deals with the constant documentation of changes, regarding change activities (EC processes) as well as old and new states of product documents (e.g. Huang and Mak 1999; Rouibah and Caskey 2003). This can also be seen as the view of industrial standards (e.g. ISO 26262 or the German DIN 199-4 and DIN 6789-3) which aim to ensure that companies comply with other industrial standards such as ISO 9001 or with the requirements of their OEM customer (e.g. VDA 4965). Yet, as indicated above, ECM additionally pursues strategies besides the effective and efficient processing and documentation of ECs. It further aims to avoid and anticipate ECs and to learn from ECs from the past.

3.3. Issues regarding the interface between requirements management and engineering change management

In order to refer to an artifact that can be regarded as the object of an EC (e.g. parts, components or product documentation) we use the term ‘solution artifact’ (SA) which is part of the solution domain. The term ‘requirement artifact’ (RA) represents a goal or a requirement. The problem domain is the counterpart to the solution domain and characterizes the problem that is addressed by the developed product by requirement artifacts.

A planned change of a solution artifact (i.e. an EC) should not be considered without looking at the requirements, because several requirements can be affected indirectly and lead to the need to change another solution artifact. The same is valid the other way around. When a requirement artifact is changed (i.e. a RC) it can affect other requirement artifacts directly or indirectly via several solution artifacts. As illustrated in Figure 3, there are known interrelations within requirement artifacts and solution artifacts and also dependencies in-between. For instance, RA 1 and RA 2 are related to each other, SA 1 and SA 2 fulfill RA 1. When there is an EC of SA 1, it could affect RA 1, which is related to RA 2. Then, if RA 2 has to be adapted, also SA 4 could be affected. Hence, there is an unknown dependency between SA 1 and SA 4 (an illustrative example for these correlations is given in Koh et al. 2012).

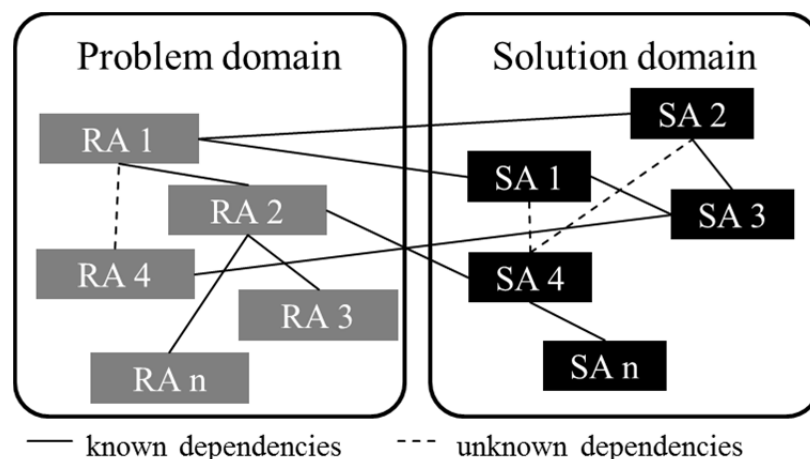


Figure 13: Dependencies between requirement artifacts (RAs) and solution artifacts (SAs)

Within the often referenced ECM and RM literature there is no interface directly addressed. Only the link of ECM to configuration management gives a small hint to consider both, solution artifacts and requirement artifacts, since configuration management aims to overlook all functional and physical characteristics and their changes and to verify the compliance with product specifications (i.e. requirements) (Jarratt et al. 2011).

Only few publications are found that directly associate RM and ECM (Koh et al. 2012; Koh et al. 2008; Morkos et al. 2012). They initially come from ECM and research on change propagation. Indirect dependencies through requirement and solution artifacts are investigated with the help of matrices, where known dependencies are modeled and indirect dependencies

are derived. However, using matrices bears some weaknesses, as they quickly become confusing if they grow in size or if weightings for the dependencies, different kinds of dependencies, conditional dependencies and several propagated indirect dependencies have to be considered. Moreover, the authors look at the dependencies in order to estimate change propagation, without describing implications for ECM and RM from the process and organizational perspective.

Further articles regarded as relevant mainly deal about product lifecycle management and product data management (e.g. Andersson et al. 2003). They refer to RM as well as to ECM, but they rather just mention the terms without investigating the implications for the interface.

When looking into practice, the treatment of RCs and ECs can be very different. For instance, in one company that is represented in our focus group RCs are not considered integrated with ECs, besides seeing them as a trigger for ECs. In another company, requirements also have a code number and are thereby treated the same as product components or its documentation in their product data management (PDM) system. In a third company, requirements are stored in a special IT system. However, the consideration of all necessary solution artifacts and requirement artifacts that could be affected by one change request does not follow a structured formal process. On an organizational level, teams are built who have to assess and decide on the request.

4. Implications for the interface between requirements management and engineering change management

This paper does not investigate change propagation in order to assess change effects, but looks at it from a procedural and organizational perspective. This is important, since every propagated change theoretically leads to another change request (regardless if it concerns a RC or EC), which somebody has to decide upon. The conceivable procedural interactions between ECs and RCs are depicted in Figure 4. When there is a target deviation, the change procedure is triggered. ECs are either triggered by a deficient actual state (i.e. product does not meet requirements) or by a deficient or changed nominal state (for example misunderstood or changed customer requirements) (Fricke et al. 2000; Herberg et al. 2010). In both cases an initial change is necessary and requested either in the solution domain (EC) or in the problem domain (RC). However, the change can make other changes necessary both in the solution domain and in the problem domain (i.e. further ECs or RCs are requested as depicted with gray arrows in Figure 4). This leads us to the conclusion, that the management of ECs and RCs should not be seen separated.

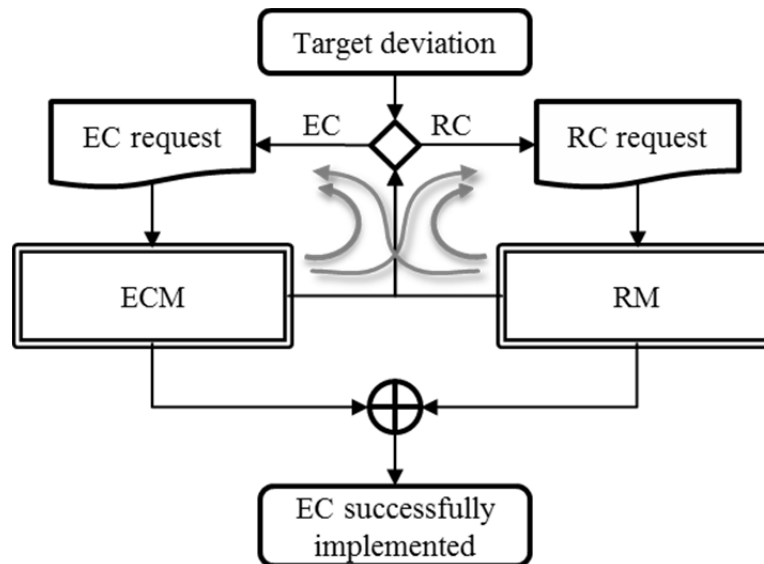


Figure 14: Change cycles within and between RM and ECM

The information that is needed for a holistic estimation of the effects of one change is often distributed over a large variety of documents or it is only tacit knowledge that single engineers have. With the concept of traceability this information on each artifact that is generated during the development process can be stored systematically. If companies are able to use this hidden and distributed information they can improve the efficiency of their change processes. Furthermore, they can avoid errors or additional work by reducing unforeseen changes and hence avoid or shorten their development cycles.

5. Cycle-oriented traceability for the management of changes

Even though neither ECM literature nor RE/RM literature addresses the synchronization of ECs and RCs directly, it is encountered indirectly. There are approaches (Koh et al. 2012; Morkos et al. 2012) to assess change propagation not only on a component level, but also via requirement relationships. Furthermore, in ECM it is suggested to build an engineering change board including people from all relevant domains who could be affected by the change. Members in our focus group from industry state that people who control the requirements are also part of that committee in their companies. However, since the synchronization is not prescribed in a formal process, there is potential for errors.

Errors may originate e.g. because some domains affected by the change can be forgotten or people (and thereby their knowledge) exit the company. Traceability bears the potential to formalize the necessary investigation of interrelations between ECs and RCs.

5.1. Theoretical background to traceability

With the ongoing digitalization more and more know-how is stored in documents as presentations, reports or construction plans. The implementation of a systematic organization and reuse of those documents awards companies with a competitive advantage (Liebowitz 1999). Traceability aims to reuse such knowledge. Hence, experiences of individual and

organizational knowledge become available and can be provided for future activities in order to improve processes and engineering designs (Hicks et al. 2002). Moreover, knowledge-based product development aims to reuse best practices, reduced cycle times and improvements in product quality and variety (Rezayat 2000). Traceability jointly connects single knowledge items or fragments in order to generate, dispose, retrieve, transform and apply them.

Especially the ability to follow the life of software artifacts has been used as a quality attribute for software (Winkler and Pilgrim 2010). Defining, describing, capturing and following traces from and to artifacts of a software development are driven by requirements. Therefore, the requirements engineering community has been the largest driver of traceability research. Traceability is defined in the IEEE Standard Glossary of Software Engineering Terminology (Radatz et al. 1990) as:

1. “The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor–successor or master–subordinate relationship to one another. [...]
2. The degree to which each element in a software development product establishes its reason for existing.”

To document the various dependencies, known or unknown, direct or indirect traceability links show the influence between the artifacts (Winkler and Pilgrim 2010). Two kinds of linkages must be differentiated: A unidirectional depends-on or a bidirectional alternative-for link. Both can indicate an order in time or causality. Spanoudakis and Zisman (2005) define eight different classes of traceability links listed in Table 8. They are partly used in our data model for traceability in change management in the following section.

Table 8: Classes of traceability links
Source: (Spanoudakis and Zisman 2005)

Traceability links	Explanation
Dependency	Indicates that the existence of an artifact depends on another
Generalization/Refinement	Shows the complexity of an artifact
Evolution	Reflects the change of an artifact
Satisfaction	Indicates, that an artifact was satisfied by another
Overlap	An intersection of two artifacts
Conflicting	Shows conflicts and inconsistencies between artifacts
Rationalization	Specifies the justification of the evolution of artifacts
Contribution	Shows the relationship between requirements and stakeholders

Traceability information helps to assess the effect of a requirements change and links to related requirements. It is the ability to verify an item by documented recorded identification. Model-driven development (MDD), an area where parts of the software development process are executed automatically, is able to leverage traceability by automatically generating these documented recorded identifications (Winkler and Pilgrim 2010). It can be identified at any time, where, when and by whom solution artifacts are developed, manufactured, processed,

stored, transported, used or disposed. This linking of requirements to system models increases the comprehensibility of the system. The impact of changing stakeholder requirements during the project is easier to assess. An explicit linking and traceability between requirement artifacts and solution artifacts facilitate crosschecking of system models with the associated requirements.

Outside the software and requirements engineering community the concept of traceability has attracted less attention among researchers. However, recently there has been significant progress in this area, for example on the field of tracing engineering information (Pavković et al. 2013). While software development by its very nature produces artifacts (e.g. code or documentation) that are interpretable by machines so that traceable elements can be mostly created automatically and managed by common requirements engineering tools, the situation with the development of mechanical products is quite different. Štorga (2004) argues that there are difficulties in achieving traceability in product development projects due to the incompatibility of information among heterogeneous design tools as well as human factors and the design process itself. An important issue regarding the implementation of traceability in engineering design is that only things that leave traces are traceable. This challenge is met by explicitly documenting different change states of an EC or RC on the one hand and all kinds of artifacts that are affected by the change on the other hand.

5.2. A data model for traceability in engineering change management

Štorga et al. (2011) present a reference model for traceability records that considers product and process related traceability elements regarding the four perspectives requirements, change, characteristics and decision traceability. Hence, they provide a useful holistic framework to achieve traceability of information objects within engineering design. However, the reference model does not address the different stages a change can evolve to. The evolving stages of ECs or RCs can be summarized as follows: A change proposal is triggered by an issue (i.e. target deviation), which then evolves to a change request if it is decided to go after it. Finally, when there is a promising option to solve the change issue, the change request becomes a change order which is the starting shot to implement the change. For every evolving stage, every domain directly or indirectly affected by the change has to be integrated, regardless if the initially triggered change was an EC or RC. This view can even be extended to other domains in the product lifecycle, such as production. Thus, the reference model by Štorga, et al. (2011) can be complemented by expanding the understanding of a 'change' from a single product component modification to a network of several related changes to requirement artifacts, solution artifacts or production artifacts. These correlations are modeled in a data model shown in Figure 15.

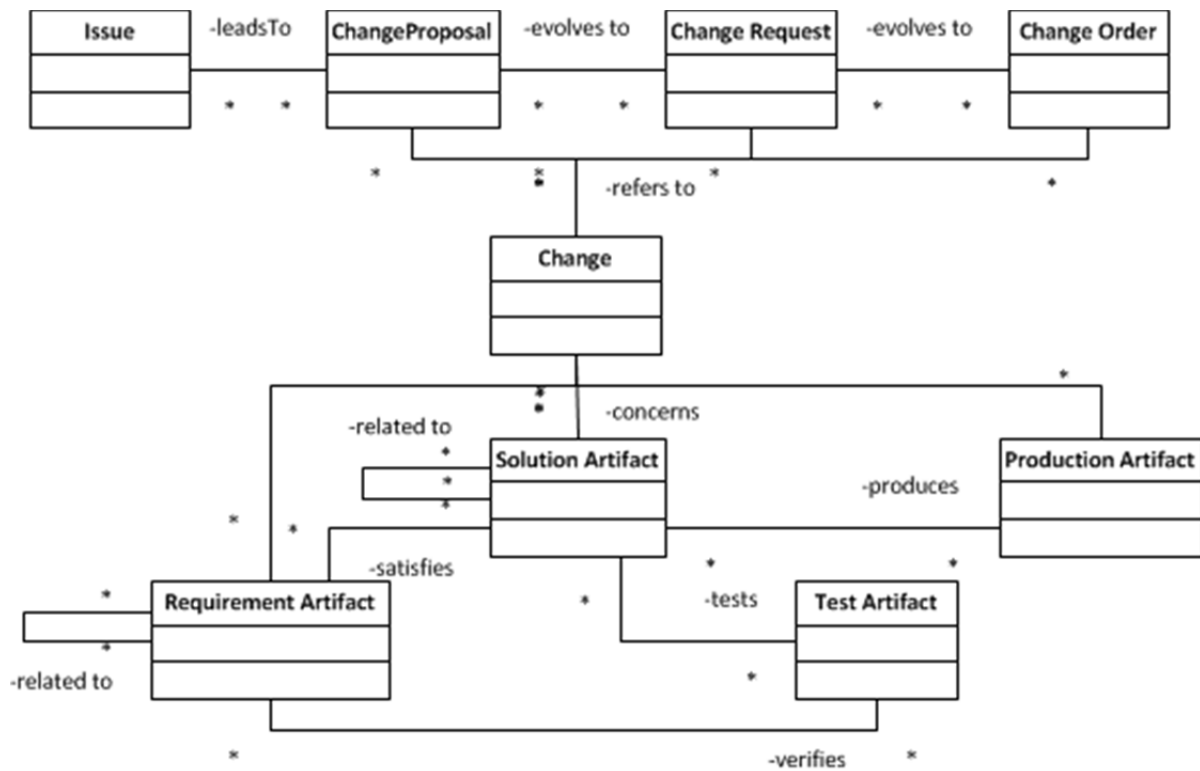


Figure 15: Data model for traceability in engineering change management

The data model for traceability in ECM does not describe the change process itself but gives a static structure of how information regarding different artifacts within the ECM process is interconnected. An upcoming issue (i.e. target deviation) leads to a change proposal which then is connected to the entity change. The change proposal then evolves into a change request in the static model. The change request process determines attainability and plans the change. A successful change request then evolves into a change order where all information of issue, change proposal and change request are collected and associated with change request responsibilities, also for sub-activities for each part. The change order is also directly linked to the entity change through a refers-to connection. A change is further linked to three artifact types: requirement artifact, solution artifact and production artifact. The latter refers to all artifacts within production that are related to a product such as production processes or tools. They also have to be considered, since ECs can not only have direct impact on production processes or tools, but also changes in production artifacts often require other changes. The solution artifact is often related to itself and can offer various solution options. Requirement artifacts and solution artifacts are linked through a relationship named 'satisfies', denoting which of the final solution artifacts are based on which requirements. A requirement artifact can additionally be related to another one. The solution artifact is tested through a test artifact, which verifies if the requirement artifact is satisfied by the solution artifact.

We believe that the tight connection between ECs and RCs should be reflected by deeply integrated tools or even a single tool for both RM and ECM. This way, analyzing the impact of ECs on requirements and vice versa can be facilitated. For solution, test and production artifacts on the other hand it seems promising to manage them in separate specialized tools and represent those artifacts as traceability records as proposed by Štorga et al. (2011). The

presented data model builds a basis to show the interrelations between different artifacts in the context of changes. With its help, traceability in ECM can be established, where not only ECs on a component level but all kind of product changes (including changes in service artifacts of a PSS), changes in requirements, changes in production processes or tools and changes in testing are considered. The following section gives an illustrative example of the data model.

5.3. Academic example

For an illustrative explanation of the data model we refer to an academic example of a pick and place unit of an industrial plant (for more details see AIS, Institute for Automation and Information Systems 2014). The unit grabs different work pieces (WP) that are stored in a stack and puts it into a stamping module where a specific note is stamped on the WP. Afterwards the WPs are transported to a sorting belt and are distributed to different slides depending on their material (white plastic, black plastic or metal). During the development of the system a customer changed one requirement so that not every WP had to be stamped anymore but only those made of metal. The changed customer requirement as an upcoming issue leads to a change proposal where the artifact that has to be changed is identified. This is the requirement “R1: All WPs have to be stamped” in our example. The change proposal then evolves into a change request that suggests changing the requirement to “R1*: Metal WPs have to be stamped, other materials can be stamped optionally”. The required change leads to other potential changes of the already developed system. For instance, the three options “EC1: install additional inductive sensor that differentiates material”, “EC2: modify software and use other existing sensors” or “EC0: no modification” are considered. The first two alternatives require further RCs, ECs or changes in production. Information about the affected artifacts and their respective relationships is stored in the entity change and thus can be retraced for the different options. After the decision on which option to implement, the change request evolves into a change order. The concept of cycle-oriented traceability helps to react to further cyclic influence factors that lead to changes by providing information about the history of past changes not only to solution artifacts, but also to requirement artifacts and production artifacts.

6. Conclusion and outlook

Engineering changes and requirement changes strongly interfere with each other. With the help of traceability in engineering change management this interface can be formalized on a process and organizational level. Thereby, unforeseen changes promoted by the initial change, errors and forgotten dependencies can be avoided. Further, the processing time of a change can be shortened.

By describing the organizational structure of potential interrelations of changes within the requirement and solution domain we aim towards a cycle-oriented management of changes. At the same time, the investigation of the underlying procedures for changes shows that the processes are very similar for all kinds of changes. Based on these findings, we presented a data model as research in progress where different evolving stages of a change and different artifacts related to the change are included. The data model should be seen as complementary

to the reference model for traceability records elaborated by Štorga, et al. (2011). The elements of the data model indicate knowledge items that are required or produced in a change process. The concept of traceability facilitates the access to necessary information about relations and dependencies between solution artifacts, requirement artifacts and production artifacts. Hence, the effects of a change to one artifact can easily be estimated and relevant people for the decision about a change and the implementation of a change can be identified. This leads to less errors and unforeseen effects of a change.

The approach towards a cycle-oriented, integrated management of any kinds of changes with the help of traceability bears potential and has to be further developed. Moreover, also changes in other company departments such as sales, marketing, quality management, etc. that are influenced by various internal and external factors could be managed by a cycle-oriented traceability. Therefore, we plan to extend the data model with artifacts regarding these departments and with their respective dependencies. The data model will then be tested as initial evaluation in a student research project. Another potential given by the use of traceability in ECM regards the learning from previous changes. Sharafi et al. (2010a) argue that especially in large organizations and in the context of complex products, engineering change management can be supported by hidden, but valuable knowledge. This knowledge can be discovered in the history of former change processes. The knowledge from former cycles can be utilized through successful data management to speed up iterative change and engineering processes (Sharafi et al. 2010a). An important factor for the reuse of knowledge is traceability. Through the compound of elements, the knowledge becomes contextualized. This in turn enables its transfer and successful reuse (Ramesh 2002).

Acknowledgement

We thank the German Research Foundation (Deutsche Forschungsgemeinschaft – DFG) for funding this project as part of the collaborative research center ‘Sonderforschungsbereich 768 – Managing cycles in innovation processes – Integrated development of product-service-systems based on technical products’. Additionally, we thank the practitioners in our industrial focus group on engineering change management for the prosperous collaboration.

Publication 4

Traceability von Anforderungen und Tests in agilen Softwareentwicklungsprojekten

Thomas Wolfenstetter, Jonas Zitzelsberger, Markus Böhm, Helmut Krcmar

^a Chair for Information Systems
Technische Universität München
Munich, Germany
{thomas.wolfenstetter, jonas.zitzelsberger, markus.boehm, krcmar}@in.tum.de

Abstract

Eine wesentliche Herausforderung bei Softwareentwicklungsprojekten besteht oft in der Notwendigkeit, mittels eines strukturierten Testmanagements zu überprüfen, ob die Anforderungen sämtlicher Stakeholder hinreichend erfüllt werden. Um dieses Ziel zu erreichen, muss transparent sein, wie Anforderungen, Lösungskomponenten und Tests miteinander in Beziehung stehen. Diese Eigenschaft wird allgemein als Traceability bezeichnet. Um Traceability sicherzustellen, bedarf es einer umfassenden Spezifikation und kontinuierlichen Aktualisierung der Abhängigkeiten zwischen den Artefakten. In der Praxis werden jedoch zunehmend agile Entwicklungsmethoden eingesetzt, bei welchen die dokumentierte Spezifikation zugunsten der flexiblen Kommunikation zwischen Entwicklern und Anwendern in den Hintergrund rückt. Bisherige Ansätze zur Sicherstellung der Traceability beziehen sich häufig auf traditionelle, phasenorientierte Projektvorgehensweisen und lassen sich daher nur eingeschränkt auf den agilen Kontext übertragen. Wie Traceability in agilen Projekten sichergestellt werden kann, wurde bisher nicht ausreichend betrachtet. In dieser Arbeit wird daher untersucht, wie Traceability in agilen Softwareprojekten gewährleistet werden kann. Dazu werden die besonderen Herausforderungen im agilen Kontext anhand einer Fallstudie aus der Industrie analysiert. Anschließend wird ein Datenmodell vorgestellt, welches die relevanten Artefakte abbildet und miteinander verknüpft.

1. Motivation

Bei der Erstellung von individueller oder der Anpassung standardisierter Unternehmenssoftware für spezifische Problemstellungen muss das zu entwickelnde System meist nahtlos in eine bestehende Systemlandschaft eingefügt werden können. Zu diesem Zweck müssen die Anforderungen vieler verschiedener Anwender und anderer Interessensgruppen berücksichtigt werden. Da es sich hierbei oft um relativ abstrakte Anforderungen handelt, können diese zu Beginn eines Entwicklungsprojekts oft nicht genau spezifiziert werden (Grande 2013). Oftmals werden die spezifischen Anforderungen erst im Laufe des Projekts ersichtlich und sind dabei ständigen Änderungen unterworfen (Highsmith 2002). Daneben werden solche Anforderungen im Verlauf eines Entwicklungsprojekts immer vielfältiger, zunehmend komplexer und ändern sich in schnelleren Zyklen (Schaffry 2008). Um flexibel auf solche Änderungen zu reagieren und auch spät identifizierte Anforderungen noch umsetzen zu können, werden bei der Softwareentwicklung agile Vorgehensmodelle immer beliebter (Ghazarian 2008).

Im Gegensatz zum phasenorientierten Vorgehen werden im agilen Kontext Individuen und deren Interaktion statt Prozesse und Tools fokussiert. Funktionierende Software steht vor verständlicher Dokumentation und die Fähigkeit, auf Änderungen reagieren zu können, steht vor der Verfolgung eines straffen Zeitplans (Williams and Cockburn 2003). Abgesehen von den kurzen Iterationszyklen und der daraus resultierenden Flexibilität, unterscheidet sich die agile Softwareentwicklung in weiteren Punkten von der phasenorientierten Entwicklung. Statt einer individuellen Zuweisung spezieller Projektrollen, agieren im agilen Kontext sich selbst organisierende Entwicklungsteams, innerhalb derer Rollen zwischen den einzelnen Personen getauscht werden können (Nerur et al. 2005). Eine weitere, zentrale Eigenschaft der agilen Entwicklung ist die kontinuierliche, enge Zusammenarbeit der Entwickler untereinander und mit externen Interessensgruppen. Diese Zusammenarbeit manifestiert sich in zahlreichen Meetings entlang des gesamten Entwicklungsprozesses. Im Gegensatz zu phasenorientierten Vorgehensmodellen, bei welchen die Anforderungsspezifikation bereits in einem frühen Projektstadium fixiert und anschließend nur noch in Ausnahmefällen geändert wird, finden bei agilen Methoden ständige Neuplanungen auf Basis der aktuellen Problemstellungen und Erkenntnisse statt, um so flexibel auf Änderungen oder neu identifizierte Anforderungen reagieren zu können (Highsmith 2002).

Um dies zu erreichen, wird weitgehend darauf verzichtet, umfangreiche Spezifikationsdokumente, wie sie aus phasenorientierten Vorgehensmodellen bekannt sind, anzulegen und zu pflegen. Stattdessen setzen agile Methoden auf bilaterale Kommunikation zwischen den Projektbeteiligten und damit einhergehend, auf informellen Wissenstransfer (Cockburn and Highsmith 2001). Genau hier zeigt sich jedoch eine mögliche Schwachstelle der agilen Entwicklung. Weil der Fokus auf der Kommunikation zwischen den Projektbeteiligten liegt, werden Ergebnisse oftmals nicht strukturiert oder nur sehr oberflächlich dokumentiert. Diese oberflächliche Dokumentation führt jedoch im weiteren Verlauf des Projekts häufig zu Mehrarbeit und kostet damit zusätzliche Ressourcen.

Gerade bei Software, die eine zentrale Rolle im Geschäftsbetrieb eines Unternehmens einnimmt und daher vom ersten Tag an tadellos funktionieren muss, gilt es sicherzustellen, dass alle essentiellen Anforderungen identifiziert und umgesetzt wurden, bevor das System in den laufenden Betrieb integriert wird. Dies ist durch Testfälle für sämtliche Anforderungen zu verifizieren. Hierzu muss jedoch transparent sein, wie Anforderungen, Code und Testfälle miteinander in Beziehung stehen. Diese Fähigkeit, nämlich den Lebenszyklus einer Anforderung und deren Beziehungen zu anderen Artefakten der Softwareentwicklung zu erfassen, und damit nachvollziehen zu können, wird als Traceability bezeichnet. Traceability sollte sowohl vorwärts als auch rückwärts entlang des Entwicklungsprozesses gegeben sein - das heißt sowohl von der Quelle einer Anforderung über die Detaillierung und Spezifikation zu der letztendlichen Umsetzung der Anforderung und den korrespondierenden Testfällen als auch in der entgegengesetzten Richtung (Gotel and Finkelstein 1994).

Traceability schafft Transparenz über die Umsetzbarkeit von Anforderungen sowie über die Zusammenhänge zwischen den Anforderungen untereinander und weiteren Entwicklungsartefakten, wie beispielsweise Tests. Hierdurch verbessert sich meist auch die Qualität der Lösung und deren Anpassbarkeit an Änderungen (Lee et al. 2003). Durch das explizite Erfassen der Beziehungen und Abhängigkeiten zwischen den Artefakten kann jederzeit nachvollzogen werden, welche Anforderungen bereits umgesetzt wurden oder zu welchem Grad diese umsetzbar sind (Ramesh and Jarke 2001). Bei Anforderungsänderungen können, auf Basis der durch Traceability verfügbaren Informationen, die betroffenen Entwicklungsartefakte, wie beispielsweise bestimmte Softwarefeatures oder Testfälle, identifiziert und entsprechend angepasst werden (Sommerville and Kotonya 1998). Weiterhin können diese Informationen genutzt werden, um Lücken oder unklar formulierte Anforderungen aufzudecken oder sie können für das Projektcontrolling herangezogen werden (Ramesh and Edwards 1993). Da sich insgesamt die Transparenz verbessert, können auch weitere Personen leichter in das Projekt integriert werden (Ghazarian 2008). Diese Transparenz sollte auch bei agilen Softwareentwicklungsprojekten gewährleistet sein. Wie die im folgenden Abschnitt dargestellte Analyse der existierenden Publikationen auf diesem Gebiet zeigt, beziehen sich bisherige Ansätze zur Gewährleistung von Traceability jedoch überwiegend auf phasenorientierte Projektvorgehensweisen. Lediglich vereinzelte Arbeiten befassen sich im Ansatz mit Traceability im agilen Kontext.

2. Methodik

Zur Entwicklung eines möglichst praxistauglichen Ansatzes für Traceability im agilen Kontext ist es von zentraler Bedeutung, die in der Realität auftretenden Herausforderungen genau zu kennen. So gestalten sich beispielsweise reale Entwicklungsprojekte meist anders als ursprünglich in der Theorie vorgeschlagen. In der Praxis erfolgt meist eine Anpassung an die Spezifika des Unternehmens sowie des Projekts. Die in dieser Arbeit angewandte Methodik richtet sich daher nach dem Design Science Ansatz (Hevner 2007). Bei diesem Ansatz werden relevante Konzepte in der wissenschaftlichen Theorie identifiziert und auf Herausforderungen angewendet, welche in der Praxis bestehen. In mehreren, schrittweisen Iterationen kann so ein möglichst praxisnaher Lösungsansatz für eine bestimmte Problemstellung gefunden werden.

Zu diesem Zweck wurde ein konkretes Beispielprojekt eines deutschen Industriekonzerns (Alpha), bei welchem ein Product-Lifecycle-Management System in die bestehende Softwarelandschaft integriert wurde, untersucht. Um existierende Methoden und Werkzeuge zur Gewährleistung der Traceability auf den agilen Kontext übertragen zu können, müssen Beziehungen zwischen den generischen Entwicklungsartefakten sowie weitere Herausforderungen der agilen Vorgehensweise untersucht werden. Zu diesem Zweck wurde die relevante Literatur nach dem Vorgehen von Webster und Watson (2002) identifiziert. In mehreren Iterationen wurden durch eine stichwortbasierte Literaturrecherche relevante Publikationen zum Thema Traceability in agilen Entwicklungsprojekten identifiziert und ausgewertet, um deren Kernkonzepte auf das Fallbeispiel übertragen zu können.

Um die zentralen Herausforderungen zu erheben und die angewendete Scrum-Vorgehensweise im Detail zu verstehen, wurden semi-strukturierte Experteninterviews mit sieben Vertretern des Entwicklungsteams und externer Fachbereiche bei Alpha geführt. Anschließend wurden die Interviewtranskripte nach den Richtlinien von Gläser und Laudel (2010) inhaltlich qualitativ ausgewertet. Hierdurch konnten der Entwicklungsprozess, die dabei entstehenden Artefakte sowie Lücken in der Traceability identifiziert und nachvollzogen werden. Besonderer Fokus lag dabei auf der Frage, welche Artefakte zu welchem Zeitpunkt der Entwicklung aus welchen Prozessschritten resultieren und für welche Schritte diese Artefakte wiederum den Input darstellen.

Auf Basis der in der Praxis identifizierten Herausforderungen sowie der Erkenntnisse aus der Literaturanalyse wurde ein konzeptuelles Datenmodell entwickelt, welches die Artefakte in agilen Entwicklungsprojekten miteinander in Beziehung setzt und die erforderlichen Relationstypen (Trace Links) beschreibt. Dieses wurde in Abstimmung mit den befragten Experten von Alpha in mehreren, iterativen Schritten bewertet und optimiert.

3. Bestehende Ansätze zu agiler Traceability

Die Analyse der Literatur zum Thema Traceability zeigt, dass sich der Großteil der untersuchten Publikationen auf phasenorientierte Entwicklungsmodelle bezieht. Hinsichtlich der agilen Entwicklungsmethodik existieren hingegen bisher nur vereinzelte Ansätze aus einer relativ generischen Perspektive.

Bouillon et al. (2013a) beschreiben einen leichtgewichtigen Ansatz zur Gewährleistung von Traceability in agilen Entwicklungsprozessen. Der Fokus dieser Arbeit liegt in erster Linie darauf zu zeigen, dass Traceability auch im agilen Kontext wichtig ist sobald das Projekt eine kritische Komplexität aufweist. Hierbei werden wichtige Entwicklungsartefakte angeführt. Insgesamt gehen die Autoren jedoch nicht darauf ein, wie die Beziehungen zwischen diesen Artefakten im Detail aussehen und welche Informationen benötigt werden, um die dargestellten Abhängigkeiten von Anforderungen und Testfällen nachvollziehen zu können.

Ghazarian (2008) vertritt den Standpunkt, dass gewisse Arten von Anforderungen immer wieder zu ähnlichen Softwareentwurfsmustern führen. Vor diesem Hintergrund wird die Möglichkeit diskutiert, Traceability von Anforderungen dadurch sicherzustellen, dass die Struktur des Software Codes gewissen Regeln unterliegt. Dieser Ansatz eignet sich jedoch nur

für sehr detaillierte Anforderungen, die bereits in der Sprache des Entwicklers beschrieben sind. Zudem wird Traceability zwischen Anforderungen und Testfällen durch diesen Ansatz nicht abgedeckt.

Ein weiterer Ansatz zur Gewährleistung von Traceability in agilen Projekten beschäftigt sich mit der Herausforderung, wie Anforderungen, die oft nur informell in Meetings diskutiert und nicht formal spezifiziert werden, explizit bestimmten Softwarefeatures zugeordnet werden können. Hierzu wird ein Werkzeug vorgestellt, mit dessen Hilfe Verknüpfungen zwischen Abschnitten aus Meeting Transskripten und zu erstellenden Softwarefeatures angelegt und verwaltet werden können (Lee et al. 2003). Dieser Ansatz ermöglicht jedoch lediglich die Identifikation zu implementierender Features und ist somit der Anforderungserhebung zuzuordnen. Im Gegensatz dazu liegt der Fokus dieser Arbeit auf der Abbildung des gesamten, agilen Entwicklungsprozesses und somit auf der Nachvollziehbarkeit ausgehend von Anforderungen über die Implementierung bis hin zu den Testfällen. Nur durch diese übergreifende Perspektive kann das volle Potenzial der Traceability realisiert werden (Wolfenstetter et al. 2013).

Espinoza und Garbajosa (2011) beschreiben ein ausführliches, konzeptuelles Datenmodell zur Sicherstellung der Traceability in Abhängigkeit vom jeweiligen Projektkontext. Dabei wird jedoch nicht aufgezeigt, inwiefern die, durch die Traceability gewonnenen Erkenntnisse, verwendet werden können, um die Auswirkungen von Anforderungsänderungen zu antizipieren. Ebenso versuchen die Autoren einen eher generischen, von der agilen Entwicklungsmethodik unabhängigen, Ansatz zu definieren, der zudem stark von phasenorientierten Entwicklungsansätzen beeinflusst wird. Hierbei werden Anforderungen, Code und Tests unabhängig voneinander betrachtet.

Die hier diskutierten Publikationen sollen den Fokus und die Perspektiven auf Traceability in agilen Projekten ausreichend widerspiegeln, um die offenen Herausforderungen aufzuzeigen. Ein Anspruch auf Vollständigkeit soll jedoch hier nicht erhoben werden. Ziel der vorliegenden Arbeit ist es, basierend auf der Analyse eines Fallbeispiels aus der Praxis, ein konzeptuelles Datenmodell für Traceability in agilen Projekten abzuleiten. Ein zentraler Punkt hierbei ist, zu analysieren, welche Artefakte und Informationen zu welchen Zeitpunkten während der Entwicklung entstehen (Hayes et al. 2009) und wie diese genutzt werden können, um Änderungen zu antizipieren und Projektressourcen zu schonen. Dies betrifft beispielsweise die Detaillierung der groben Anwendungsfälle, aus denen detailliertere Anforderungen (User Stories) entstehen. Weiterhin wird in dieser Arbeit Traceability sehr nahe am agilen Entwicklungsprozess beschrieben, wodurch es beispielsweise möglich wird, für die Implementierung fehlende Anforderungen oder gewisse Inkonsistenzen zu identifizieren.

4. Fallstudie: Agile Softwareentwicklung bei Alpha

Inhalt des untersuchten, agilen Entwicklungsprojekts ist die Anpassung einer kommerziellen Product-Lifecycle-Management Systems eines Drittanbieters und dessen Integration in die bestehende Softwarelandschaft von Alpha. Alpha ist ein multinationaler Industriekonzern mit Sitz in Deutschland, der mechatronische Produkte entwickelt, produziert und vertreibt. Der betroffene Anwenderkreis umfasst mehrere tausend Personen aus verschiedenen Abteilungen und an unterschiedlichen Standorten.

4.1. Der agile Entwicklungsprozess

Im Unterschied zu den phasenorientierten Vorgehensmodellen, bei denen die eigentliche Implementierung erst nach einer detaillierten Spezifikationsphase beginnt, setzt der agile Entwicklungsprozess bei Alpha auf kontinuierliches Prototyping. Hierbei wird in kurzen Iterationszyklen (Sprints) die zu entwickelnde Lösung schrittweise erweitert. Den Anwendern wird dabei regelmäßig ein aktueller, lauffähiger Prototyp zur Verfügung gestellt. So können diese prüfen, ob ihre Anforderungen richtig verstanden und umgesetzt wurden. Anschließend können gegebenenfalls Anforderungen weiter detailliert oder angepasst werden. Figure 16 illustriert den rekonstruierten Prozessablauf bei Alpha und zeigt, welche Artefakte durch die zyklisch ablaufenden Prozessaktivitäten entstehen und bei welchen Aktivitäten diese verwendet werden.

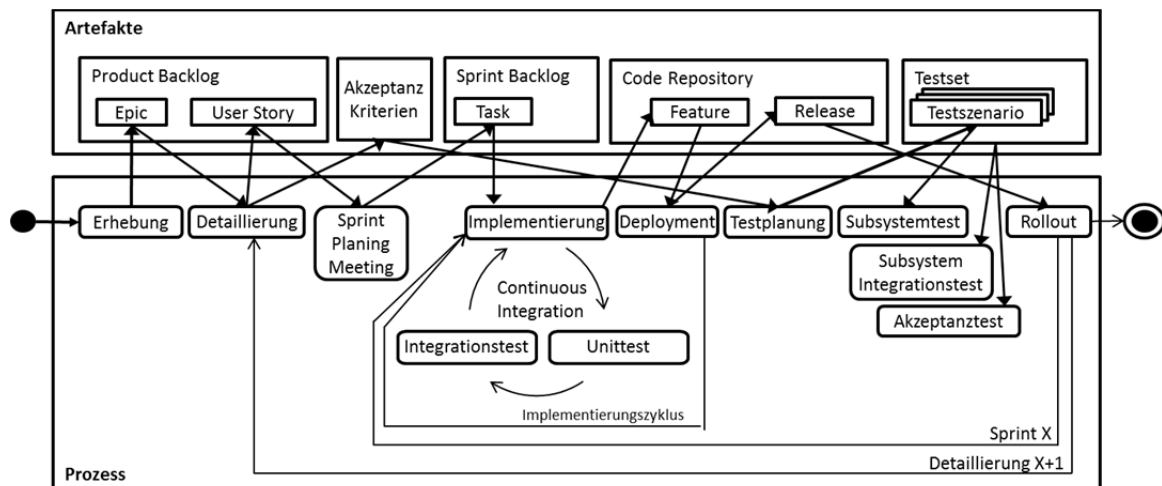


Figure 16: Schematischer Ablauf des agilen Entwicklungsprozesses bei Alpha

Zunächst werden übergreifende Anwendungsfälle (Epics) initial erhoben und im Product Backlog aufgelistet. Diese werden zu Beginn in sehr grober Form erhoben und nicht im Detail spezifiziert. Epics werden textuell, nach folgendem Muster definiert:

Als <Benutzerrolle> will ich <das Ziel>[, sodass <Grund für das Ziel>].

Beispiel: Als Ingenieur will ich einen Testfall mit entsprechenden Werten hinterlegen, sodass ich spezielle Konfigurationen testen kann.

Die spezifizierten Epics werden in einem Product Backlog aufgelistet. Für die Reihenfolge der Einträge in dem Product Backlog ist der Product Owner verantwortlich. Der Product Owner vertritt die fachliche Sicht auf die Software und ist sowohl für das Entwicklungsteam, als auch für das Management des Product Backlogs verantwortlich. Er entscheidet ebenso, welche User Stories in das Sprint Backlog übernommen werden. Bevor dies geschieht, müssen die Epics detailliert werden. Hieraus ergeben sich User Stories, welche ebenfalls textuell, durch das oben beschriebene Muster, allerdings deutlich detaillierter spezifiziert, und durch Akzeptanzkriterien konkretisiert werden. Anhand dieser, kann am Ende eines Sprints beurteilt werden, ob eine User Story erfolgreich umgesetzt wurde. Die Akzeptanzkriterien, welche zudem die Testfälle einer User Story darstellen, werden nach der Methodik „Specification By Example“ (Adzic 2011) definiert.

In sogenannten Sprint Planning Meetings werden die, durch den Product Owner priorisierten und detaillierten, User Stories durch das Entwicklungsteam in feingranulare Tasks zerlegt, welche im Sprint Backlog aufgelistet werden. Diese Tasks stellen die wesentlichen Arbeitspakete dar, welche anschließend durch das Entwicklungsteam in einem Sprint umgesetzt werden. Ein Sprint dauert maximal 30 Kalendertage und stellt den Kern der agilen Scrum-Methode bei Alpha dar.

Der Entwickler implementiert einen spezifischen Task und führt anschließend einen Unittest durch. Tritt ein Defect auf, muss der Entwickler den Fehler suchen und den Softwarecode anpassen. Dies wird so lange wiederholt, bis im Unittest kein Defect mehr auftritt. Bei der Implementierung entsteht so auf Basis des Tasks ein erstes Softwarefeature.

Nachdem der Unittest fehlerfrei verlaufen ist, führt der Entwickler den Integrationstest durch. Hierbei wird versucht, den soeben implementierten Task mit den bereits vorher umgesetzten Tasks zu integrieren. Hierbei wird das Zusammenspiel der isoliert voneinander fehlerfreien Tasks getestet. Tritt ein Defect auf, muss der zugehörige Code angepasst und auf dessen Basis ein erneuter Unittest durchgeführt werden. Erst danach kann der Task wieder integriert werden.

Nachdem auch im Integrationstest kein Defect mehr auftritt, kann der Entwickler die implementierten Features in die Testumgebung ausrollen. Dieser Implementierungszyklus wird so lange wiederholt, bis keine Tasks mehr vorhanden sind. Die implementierten Softwarefeatures werden anschließend schrittweise in der Testumgebung zu einem Release integriert. Das kontinuierliche Zusammenfügen der einzelnen Features zu einem Release bzw. das automatische Testen des ganzen Systems bis zum aktuellen Entwicklungsstand, entspricht dem Prozess der „Continuous Integration“ (Fowler and Foemmel 2006). Dieser wird durch das Einbinden des Codefragments in das Repository automatisch ausgelöst.

Nach Abschluss des Implementierungszyklus sowie der Entwicklertests werden in der Testumgebung, durch ausgewiesene Tester, manuelle Tests durchgeführt. Hierzu werden Akzeptanzkriterien zu Testszenarien gebündelt, auf deren Basis Subsystemtests durchgeführt werden. Eine weitere Testebene stellen die Subsystem Integrationstests dar. Hierfür werden mehrere Testszenarien zu einem Testset gebündelt. Ebenso werden auf Basis der Testsets Akzeptanztests für die letztendliche Abnahme in den laufenden Betrieb durchgeführt.

Nachdem diese Tests erfolgreich abgeschlossen sind, kann das Release in die Produktivumgebung ausgerollt werden.

Sind nach dem Ausrollen noch User Stories in dem Product Backlog vorhanden, können diese in einem weiteren Sprint X implementiert werden. Damit jedoch immer ausreichend detaillierte User Stories vorhanden sind, muss die Detaillierung der initial erhobenen Epics immer einen weiteren Sprint (X+1) vor deren Umsetzung stattfinden.

4.2. Herausforderungen für Traceability im agilen Entwicklungsprozess

Viele der in der Literatur existierenden Traceability Ansätze basieren auf detaillierten Anforderungsspezifikationsdokumenten. Diese Dokumente sind charakteristisch für phasenorientierte Vorgehensmodelle. Aus diesem Grund eignen sich existierende Traceability Ansätze vor allem für solche Entwicklungsmethoden. Figure 17 illustriert, welche Trace Links (1-7) zwischen den Artefakten im Fallbeispiel des agilen Entwicklungsprozesses bei Alpha nicht dokumentiert werden. Wie sich aus den durchgeführten Experteninterviews zeigt, ergeben sich dadurch die nachfolgend diskutierten Herausforderungen.

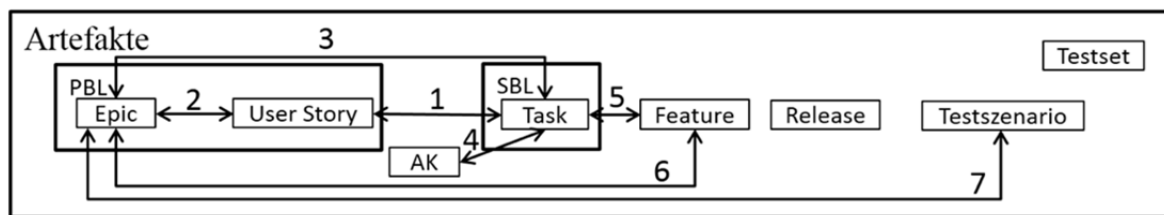


Figure 17: Herausforderungen für Traceability im agilen Entwicklungsprozess

Durch die informelle Anforderungsspezifikation im agilen Kontext, welche die direkte Kommunikation zwischen Fachbereich (Anwender bzw. Anforderungssteller) und Entwickler statt einer detaillierten Anforderungsspezifikation vorsieht, lassen sich bisherige Ansätze nur eingeschränkt auf agile Modelle anwenden. Dies betrifft vor allem den Prozessschritt der Detaillierung, bei dem die initial erhobenen Epics in einem Workshop zwischen Fachbereich und Anforderungsmanager verfeinert werden. Die Anforderungsmanager haben anschließend eine detaillierte Vorstellung über die Bedürfnisse des Fachbereichs und können auf dieser Basis die Akzeptanzkriterien für jede User Story festlegen. Dies geschieht oftmals in einem Workshop mit dem Entwicklungsteam. Durch diese Workshops und die direkte Kommunikation zwischen den beteiligten Rollen, wird eine detaillierte Dokumentation auf den unterschiedlichen Abstraktionsebenen oft als überflüssig erachtet (2).

Bei phasenorientierten Entwicklungsmethoden ist das Resultat der Spezifikationsphase ein vollständiges Anforderungsprofil. Im Gegensatz zu diesem Vorgehensmodell, existiert im agilen Kontext ein solches Anforderungsprofil nicht. Erst nachdem das Projekt abgeschlossen ist und alle Anforderungen in das Softwareprodukt umgesetzt worden sind, existiert eine vollständige, detaillierte Spezifikation der Anforderungen. Der Grund hierfür ist, dass bei der agilen Entwicklung immer nur die Anforderungen detailliert werden, welche anschließend in einem Sprint implementiert werden. Somit ist Traceability, wenn auch nur eingeschränkt,

innerhalb der Sprints und zu den bereits umgesetzten Anforderungen sichergestellt, nicht jedoch übergreifend über alle bzw. zu den noch ausstehenden Anforderungen des Softwareprojekts. Trace Links zu Anforderungen, welche zwar bereits erhoben, aber noch nicht detailliert wurden, existieren nicht. Bei industriellen Großprojekten mit einer langen Laufzeit besteht die Gefahr, dass Inkonsistenzen zwischen den Anforderungen der einzelnen Sprints auftreten. Dieses Problem betrifft auch phasenorientierte Entwicklungsmethoden, bei denen in der Spezifikationsphase Anforderungen seitenweise detailliert werden, wodurch die Übersichtlichkeit meist leidet. Dennoch ist bei solchen Entwicklungsmethoden, aufgrund der detaillierteren Dokumentation, Traceability gewährleistet. So ist es im Nachhinein möglich, Redundanzen, hinsichtlich einzelner User Stories oder Tasks bzw. Inkonsistenzen, zu erkennen (1, 2, 3).

Abgesehen von Redundanzen erschwert eine übermäßige Detaillierung, den Überblick über alle feingranularen Tasks zu behalten. In der Praxis ist es den Projektmitgliedern aufgrund des Zeitdrucks oft nicht möglich, für jeden einzelnen Task, Trace Links manuell zu dokumentieren und diese fortlaufend zu aktualisieren (2, 3).

Aufgrund fehlender Trace Links zwischen einzelnen Artefakten, die in unterschiedlichen Tools dokumentiert und verwaltet werden, ergeben sich meist weitere Probleme. Im untersuchten Fall werden beispielsweise die initial erhobenen Epics und die zugehörigen detaillierten User Stories in Form von Use Case Diagrammen sowie die von einer User Story abgeleiteten Tasks in unterschiedlichen Tools verwaltet. Aufgrund dieses Bruches kann nach Änderungen nur durch manuelle Prüfung nachvollzogen werden, welche Tasks zu einer bestimmten Epic gehören. Wegen fehlender Schnittstellen zwischen diesen Tools, kommt es bei Änderungen somit häufig zu Inkonsistenzen (1, 3). In der Folge bedarf es für die Sicherstellung übergreifender Traceability eines erheblichen, manuellen Aufwands.

Akzeptanzkriterien werden für User Stories definiert und stellen zugleich die Testfälle für diese dar. Diese Testfälle besitzen jedoch keine Beziehungen zu den abgeleiteten Tasks, welche die konkreten Arbeitspakete für die Implementierung darstellen. Zur Validierung spezifischer Akzeptanzkriterien können die dafür benötigten Tasks nicht identifiziert werden (4). Wie bereits angesprochen, werden die Tasks in einem separaten Tool für die agile Entwicklung dokumentiert. Nach der Implementierung eines Tasks, muss jedoch, beim Einbinden in das Coderepository, die zugehörige User Story angegeben werden. Somit kann nicht eindeutig nachvollzogen werden, welches Feature aus welchem Task resultiert (5). Nach der Implementierung muss manuell geprüft werden, ob bestimmte Epics vollständig umgesetzt sind. Dennoch kann nur schwer nachvollzogen werden, welche spezifischen Features, bzw. welche Codefragmente zu einer Epic gehören (6). Die Testszenarien für eine User Story werden in einem Testmanagementtool spezifiziert. Innerhalb dieses Tools können jedoch die einzelnen User Stories nicht zu übergreifenden Epics zusammengefasst werden. Somit kann nicht nachvollzogen werden, welche Testszenarien, welche Epics adressieren, bzw. welche Testszenarien zur Validierung einer spezifischen Epic benötigt werden (7).

5. Ein konzeptuelles Datenmodell für Traceability in agilen Projekten

Um den in Abschnitt 4.2 diskutierten Herausforderungen zu begegnen, ist es notwendig, die bei der agilen Entwicklung entstehenden Artefakte, sowie die notwendigen Trace Links zwischen diesen, strukturiert abzubilden. Hierzu wurden die in der Literatur identifizierten Konzepte auf den, in der Fallstudie analysierten, agilen Entwicklungsprozess übertragen und in fünf Iterationsschritten durch einen intensiven Austausch mit Alpha evaluiert und erweitert. Das in Figure 18 dargestellte Klassendiagramm stellt das resultierende, konzeptionelle Datenmodell für Traceability in agilen Projekten dar.

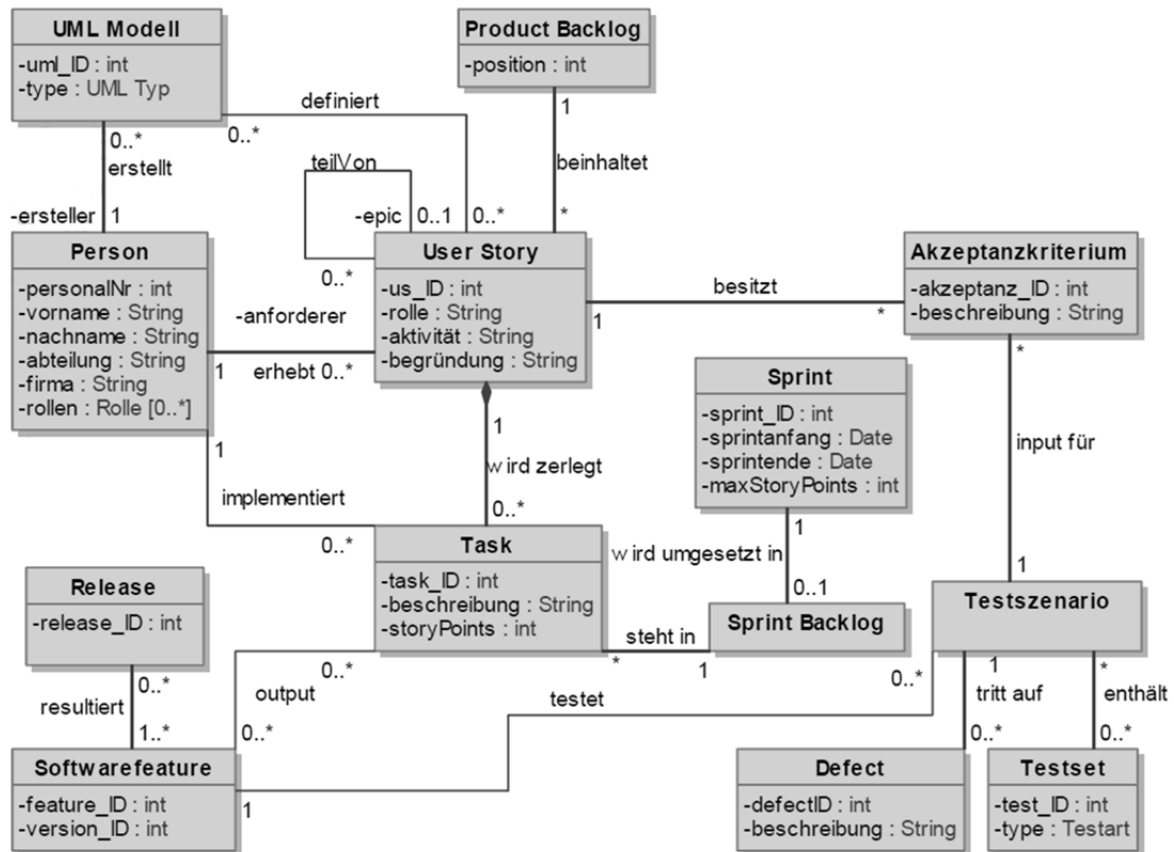


Figure 18: Konzeptuelles Datenmodell für Traceability in agilen Projekten

Eine User Story wird von einer Person, welche in dieser Beziehung die Rolle eines Anforderungsstellers einnimmt, erhoben. Eine Person kann dabei auch mehrere Rollen, annehmen. Durch die Beziehung erhebt kann nachvollzogen werden, welche Person, welche User Stories erhoben hat.

Bei der Klasse User Story handelt es sich um fachliche Rollen, wie beispielsweise einen Werkstattmeister. Diese dürfen nicht mit den Rollen der Klasse Person verwechselt werden. Eine User Story wird durch das textuelle Muster beschrieben und stellt somit eine Metaebene dar. Die Beziehung teilVon stellt die Granularität von User Stories dar. Initial erhobene User Stories werden durch grobe Epics repräsentiert, welche bei der Detaillierung in mehrere, kleinere User Stories zerfallen. Anhand dieser Beziehung kann nachvollzogen werden, welche detaillierten User Stories welchem Epic zugeordnet werden können.

Die exakte Definition einer User Story findet in einem UML Modell statt. Beispielsweise können in einem Use Case Diagramm mehrere User Stories definiert werden und eine User Story kann durch mehrere Diagrammtypen definiert sein. Durch die Beziehung erstellt, kann diejenige Person identifiziert werden, welche das UML Modell erstellt hat. Anhand der Beziehung definiert kann nachvollzogen werden, welche User Story, in welchem UML Modell definiert ist. Alle User Stories werden im Product Backlog dokumentiert. Das Attribut Position bezieht sich auf die Reihenfolge, in welcher die User Stories im Product Backlog stehen.

Bei der Detaillierung werden für eine User Story mehrere Akzeptanzkriterien festgelegt. Aufgrund der Methodik Specification By Example stellen die Akzeptanzkriterien zugleich die Testfälle einer User Story dar. Durch den Trace Link besitzt können die, zu einer spezifischen User Story zugehörigen, Akzeptanzkriterien identifiziert werden.

Das Entwicklungsteam zerlegt eine User Story in mehrere, noch detailliertere Tasks. Ein Task, welcher eine gewisse Anzahl an Story Points aufweist, wird von einer Person mit der Rolle Entwickler implementiert. Durch den Trace Link wird zerlegt können die, zu einer spezifischen User Story zugehörigen, Tasks identifiziert werden.

Die Tasks stehen in einem Sprint Backlog, das in einem Sprint abgearbeitet bzw. umgesetzt wird. Pro Sprint kann eine bestimmte Anzahl an Story Points umgesetzt werden. Über das zwischengeordnete Artefakt Sprint Backlog kann implizit nachvollzogen werden, welche Tasks, und somit welche User Stories, in welchem Sprint umgesetzt werden.

Nach der Implementierung eines Tasks können als Output mehrere Softwarefeatures resultieren. Durch die Versionierung können Änderungen innerhalb eines Features, anhand der verschiedenen Versionen nachvollzogen werden. Anhand des Trace Links output, welcher die Implementierung darstellt, kann nachvollzogen werden, welches Softwarefeature welche Tasks adressiert. Ein Softwarefeature resultiert in mindestens einem Release. Durch die Beziehung resultiert ist die Nachvollziehbarkeit der Softwarefeatures in das resultierende Release gewährleistet.

Ein Softwarefeature kann von mehreren Testszenarien getestet werden, welche von einer Person mit der Rolle Tester durchgeführt werden. Ein Testszenario hat als Input mehrere Akzeptanzkriterien. Durch den Trace Link Input für kann nachvollzogen werden, welche einzelnen Akzeptanzkriterien in welchem Testszenario gebündelt werden. Ein Testszenario kann in keinem, aber auch in mehreren Testsets enthalten sein. Ein Testszenario wird sowohl für das Testset des System Integrationstests, als auch für das Testset der Akzeptanztests benötigt. Hingegen enthält ein Testset meistens mehrere Testszenarien. Durch die Beziehung getestet kann nachvollzogen werden, welches Testszenario welches spezifische Softwarefeature testet.

Zu einem Testszenario können mehrere Defects auftreten. Anhand des Trace Links tritt auf kann nachvollzogen werden, zu welchem spezifischen Testszenario ein Defect aufgetreten ist.

6. Diskussion

Etliche der in Abschnitt 4.2 angesprochenen Probleme resultieren aufgrund von Brüchen in der zugrundeliegenden Toollandschaft. Diese Tools sind auf Unternehmensentscheidungen für die einzelnen Bereiche, wie beispielsweise das Anforderungsmanagement oder das Testmanagement, zurückzuführen. Hierbei wäre es für agile Projekte besser, wenn solche Brüche zwischen den einzelnen Tools, welche während der Entwicklung eingesetzt werden, vermieden werden. Innerhalb der einzelnen Tools ist Traceability sichergestellt - nicht jedoch toolübergreifend. Somit lassen sich Anforderungen nicht über deren kompletten Lebenszyklus hinweg konsistent nachverfolgen. Bei geeigneten Schnittstellen könnten auch toolübergreifende Trace Links identifiziert werden. Diese globale Nachvollziehbarkeit ist, vor allem bei Änderungen in großen, unübersichtlichen Projekten, von Vorteil, da aufgrund dieser Links, alle weiteren Entwicklungskomponenten angepasst werden können (Lucia and Qusef 2010).

Durch die Sicherstellung von Traceability zu den atomaren Tasks können Redundanzen bereits vor deren Umsetzung erkannt werden. Zudem wird, durch die Rückverfolgung der Trace Links zu redundanten, aber bereits implementierten Tasks, die Wiederverwendung bereits implementierter Features möglich. Ein bereits implementierter Task kann komplett oder zum Großteil für den neuen Task wiederverwendet werden. Somit wird es durch Traceability möglich, den Implementierungsaufwand zu verringern und dadurch schneller und kosteneffizienter das Sprint- bzw. Projektziel zu erreichen. Abgesehen hiervon kann mit Hilfe des in Kapitel 5 beschriebenen konzeptionellen Datenmodells und des resultierenden Beziehungswissens der agile Entwicklungsprozess in weiteren Punkten optimiert werden. Diese Optimierung findet auf Basis geeigneter Anwendungsszenarien für Traceability aus der Praxis statt (Bouillon et al. 2013b).

Bei der initialen Erhebung weist die grobe User Story den Status einer Epic auf. Bei der späteren Detaillierung werden die Epics in detailliertere User Stories zerlegt. Durch den Trace Link zwischen den verschiedenen Versionen einer User Story kann nachvollzogen werden, welche detaillierten User Stories aus welcher Epic resultieren.

Da ein Task nur von einem Entwickler umgesetzt wird, kann durch Traceability nachvollzogen werden, welcher Task von welchem Entwickler implementiert wird. Parallel zur Implementierung eines spezifischen Tasks in einem Sprint kann der Product Owner, mit Hilfe der Trace Links, den Implementierungsstatus dieses Tasks prüfen. Dies ist aufgrund der Beziehungen zwischen Entwickler, Task und Sprint Backlog möglich. Der Product Owner kann nachverfolgen, welcher Entwickler, zu welchem Zeitpunkt des Sprints einen spezifischen Task aus dem Sprint Backlog implementiert.

Nachdem einige User Stories bzw. die zugehörigen Tasks erfolgreich in ein Release umgesetzt wurden, kann der Product Owner den Projektfortschritt anhand dieser implementierten User Stories ableiten. Dies ermöglichen die Beziehungen zwischen den User Stories und dem Product Backlog, in dem alle Anforderungen aufgelistet sind. Bei der Implementierung werden die zugehörigen User Stories von dem Product Backlog in das

Sprint Backlog übernommen, wodurch es dem Product Owner ermöglicht wird, anhand der restlichen User Stories im Product Backlog den Projektfortschritt abzuleiten.

Hat sich eine bereits implementierte User Story geändert, können aufgrund des Trace Links zwischen User Story und Task, die zu der geänderten User Story zugehörigen Tasks identifiziert und entsprechend modifiziert werden.

Für die Navigation mittels Trace Links zwischen Spezifikation, Design, Code und Test müssen diverse Mappings der eindeutigen IDs stattfinden. Die exakte Spezifikation einer textuellen User Story, findet in einem UML Modell statt. Hierzu muss zur Gewährleistung der Nachvollziehbarkeit zwischen Spezifikation und Design Artefakten die ID des Modells mit der textuell spezifizierten User Story verknüpft werden. Für die Nachvollziehbarkeit zwischen Design und Code muss die ID der modellierten User Stories mit den zugehörigen Tasks verknüpft werden. Diese werden von einem Entwickler in Softwarefeatures umgesetzt. Somit ergibt sich über zugehörigen Tasks implizit Traceability zwischen Designartefakten und Code. Nach der Implementierung eines Tasks werden die resultierenden Softwarefeatures in das Repository übertragen, wobei jeweils eine neue Version entsteht. Zur Gewährleistung von Traceability zwischen Code und Test muss nun die neu entstandene Versions ID mit der ID der Entwicklertests verknüpft werden. Dadurch kann nachvollzogen werden, bei welcher Version eines Softwarefeatures, der zugehörige Test erfolgreich bestanden wurde.

Nachdem eine User Story an den Anfang des Product Backlog platziert wurde, prüft der Product Owner, ob diese mit einer bereits implementierten User Story in Konflikt steht. Diese Prüfung wird durch die Traceability einer User Story über die Tasks hin zu den Softwarefeatures möglich. Der Product Owner kann somit nachvollziehen, ob für die spezifische User Story bereits Softwarefeatures existieren. Besteht ein Konflikt, kann anhand der Trace Links der zugehörigen Stakeholder, welcher die User Story erhoben hat, identifiziert und über den Konflikt benachrichtigt werden.

Besteht kein Konflikt, kann der Product Owner die spezifische User Story bewerten. Hierzu bedient er sich Trace Links zu ähnlichen, bereits umgesetzten User Stories, wodurch er den Umfang für die aktuelle User Story abschätzen kann. Ist sie zu umfangreich, wird sie in detailliertere User Stories zerlegt. Nach diesem Prozess, bei dem jede zerlegte User Story versioniert wird, können dadurch diese zu übergeordneten User Stories bzw. Epics zurückverfolgt werden. Zusätzlich wird es durch die Zerlegung und der Trace Links zu bereits umgesetzten Softwarefeatures möglich, redundante User Stories zu identifizieren und diese zu entfernen.

Kann ein Defect eigenständig behoben werden, lokalisiert der Entwickler anhand der Versions ID der zuvor eingeeckten Softwarefeatures, den zugehörigen Code und kann diesen modifizieren. Kann der Defect aufgrund der Komplexität nicht eigenständig behoben werden, muss zur Unterstützung der Stakeholder herangezogen werden, der die zugehörige User Story erhoben hat. Diese Rückverfolgung ist mittels der Trace Links zwischen Softwarefeature und Task, zwischen Task und User Story, sowie zwischen User Story und Person gewährleistet.

Durch Traceability kann überprüft werden, ob alle Anforderungen der Stakeholder in dem Softwareprodukt vorhanden sind, bzw. ob diese deren Vorstellungen entsprechen. Auch können redundante Testfälle identifiziert werden, wodurch Ressourcen und Zeit im Testmanagement eingespart werden können. Sind alle Testfälle abgedeckt und erfolgreich bestanden, wirkt sich dies positiv auf die Qualität der Software und dies wiederum positiv auf die Zufriedenheit der Stakeholder aus.

Für zukünftige Softwareentwicklungsprojekte können Informationen bezüglich der Trace Links zwischen Anforderungen, Code und Testfällen vorausgegangener Projekte genutzt werden, sodass mögliche Problemstellungen des neuen Projekts frühzeitig identifiziert werden können.

7. Limitationen und Ausblick

Obwohl die, in dieser Arbeit vorgestellten Ergebnisse sich auf eine spezifische Fallstudie beziehen, gehen wir davon aus, dass sich die Ergebnisse problemlos bzw. mit geringen Modifikationen auf andere Unternehmen übertragen lassen. Zum einen befanden sich unter den befragten Personen auch Entwickler externer Dienstleister, zum anderen zeigt ein Vergleich des rekonstruierten Vorgehensmodells bei Alpha mit Vorgehensmodellen aus der Literatur weitreichende Übereinstimmung. Somit können abgesehen von den unternehmensspezifischen Teststufen, die dargestellten Artefakte und deren Zusammenhänge auf Anwendungsfälle und agile Entwicklungsprozesse anderer Unternehmen übertragen werden.

Für die Sicherstellung von Traceability müssen alle Beziehungen von den Anforderungen zu den spezifischen Entwicklungskomponenten, wie beispielsweise Tests, initial erstellt und im Laufe des Projekts fortlaufend aktualisiert werden. In komplexen Projekten ist dies fehleranfällig und oftmals mit großem Aufwand für die Projektbeteiligten verbunden. Für diejenigen, die diese Links dokumentiert, ergeben sich zunächst keine Vorteile. Sind jedoch Trace Links zwischen den verschiedenen Artefakten umfassend dokumentiert, resultieren zahlreiche Vorteile hinsichtlich des Projektmanagements oder einer Verbesserung der Entwicklungsqualität (siehe Kapitel 1). Während der Entwicklung können, beispielsweise im Fall von Anforderungsänderungen, alle betroffenen Komponenten identifiziert und entsprechend modifiziert werden. Wenn diese Trace Links jedoch nicht korrekt erstellt bzw. laufend aktualisiert werden, können auf dieser Basis fehlerhafte Artefakte erzeugt oder falsche Managemententscheidungen getroffen werden. Aufgrund fehlerhafter Trace Links kann es weiterhin zu falschen Designentscheidungen oder Prioritäten kommen, was wiederum zu Unzufriedenheit der Stakeholder führen kann (Lee et al. 2003).

Auch die Ergebnisse der Experteninterviews spiegeln diese Problematik wider. Um Projektverantwortliche und Entwickler in der Praxis von den Vorteilen zu überzeugen und den Aufwand für die Dokumentation und Pflege der Traceability Links möglichst gering zu halten, bedarf es daher Traceability Ansätze, die an den Bedürfnissen der Praxis ausgerichtet sind sowie geeigneter IT-Unterstützung. Das in dieser Arbeit vorgestellte konzeptionelle

Datenmodell kann hierbei als Grundlage für ein Softwaretool zur Sicherstellung von Traceability im agilen Entwicklungskontext dienen.

Danksagung

Diese Veröffentlichung entstand im Rahmen des Sonderforschungsbereichs 768 „Zyklusmanagement von Innovationsprozessen – Verzahnte Entwicklung von Leistungsbündeln auf Basis technischer Produkte“. Das Forschungsvorhaben wird gefördert durch die Deutsche Forschungsgemeinschaft (DFG).

Publication 5

Concept for an Integration-Framework to enable the crossdisciplinary Development of Product-Service Systems

*Konstantin Kernschmidt^a, Thomas Wolfenstetter^b, Christopher Münzberg^c, Daniel Kammerl^f,
Suparna Goswami^b, Udo Lindemann^c, Helmut Krcmar^b, Birgit Vogel-Heuser^a*

^a Institute of Automation and Information Systems Technische Universität München Munich, Germany (kernschmidt, vogel-heuser)@ais.mw.tum.de	^b Chair for Information Systems Technische Universität München Munich, Germany {thomas.wolfenstetter, suparna.goswami, krcmar}@in.tum.de
^c Institute of Product Development, Technische Universität München, Munich, Germany {christopher.muenzberg, daniel.kammerl, lindemann}@pe.mw.tum.de	

Abstract

Modern mechatronic Product-Service Systems (PSS), as a combination of mechanics, electrics, electronics, software and services, require an interdisciplinary system understanding and development process. During the development, each discipline uses specific modeling languages and tools, which focus on certain aspects of the system. However, much of the model information is commonly used in the different disciplines involved. Thus, it is inefficient to model these commonly used elements separately from scratch in every discipline and thereby keep the data of the system consistent. Therefore, in this paper a concept for an integration-framework is presented, which defines a specification of the relevant PSS elements and their attributes, in order to facilitate the crossdisciplinary use of model-information during the development process of mechatronic PSS.

Keywords – integration-framework, mechatronics, product-service system, interdisciplinary development

1. Introduction

The development and production of mechatronic PSS, consisting of mechanics, electrics/electronics, software and services, is a complex task and requires detailed knowledge in the different disciplines involved. During the detailed development each discipline separately creates specific models of the system, focusing on the relevant parts of interest. Thereby, different modeling languages and tools are used, which are optimized for their specific development task. As the various models only represent different viewpoints of the same system, many model elements are common across multiple models, developed for different purposes. However, these components are often defined anew in each discipline and created in every tool from scratch, therefore failing to utilize mutual information exchange. A reason for this is the lack of efficient communication of discipline-specific system-knowledge between different disciplines (Laurel and Mountford 1990), because the involved groups miss a common terminology (Borchers 2000). As most disciplines use software tools during the development phase, these tools in particular should enable and facilitate the crossdisciplinary sharing and integration of knowledge (Curtis et al. 1988). Next to the required components, a shared understanding of the included system functions is also indispensable in the development process of a mechatronic PSS (Eisenbart et al. 2012).

The main contribution of this paper is therefore developing a concept for an interdisciplinary integration-framework, which enables the different involved disciplines to map the elements of their specific modeling approaches to a joint representation of a PSS. Therefore, the relevant elements of a mechatronic PSS and their relations have to be specified. We expect that this framework will improve interdisciplinary system understanding, enable the reuse of information in the different modeling languages of the respective disciplines, and establish a basis for automatic model transformations. In this context the disciplines mechanics, electrics/electronics, software and service are considered. Through the usage of the framework the interdisciplinary development process is facilitated and a more comprehensive view of the PSS can be achieved.

2. State of the Art

In order to define the elements of an integration-framework for PSS, which includes the aggregated information from the different disciplines, their common modeling approaches are introduced in the following. Furthermore, an analysis of existing approaches for interdisciplinary system modeling and information exchange is presented.

2.1. Modeling-Approaches

Mechanics: In literature, there are several modeling approaches, which are applied during the development process of mechanical products. On the one hand they are used to analyze the given problem statement. Therefore, TRIZ and its specific tools from the field of problem analysis, i.e. function analysis and subsequent methods, are used (Orloff 2006). Equally, flow, user- and relation-oriented models are created (Lindemann 2009). Furthermore, the Function-Behavior-Structure (FBS) model of designing from Gero (1990) is used for task clarification.

The FBS gives a holistic view on a system. This approach displays the interdependencies between functions and structures, mapped with the help of expected and actual behavior.

On the other hand, function modeling aims to support designer in developing solutions. Thereby flow-, user-, or relation-oriented modelling as well as the function diagram are used. The function diagram depicts the in- and output flows of the product or system (Ulrich and Eppinger 2012). Based on this knowledge the overall functions of a product or system can be displayed and in further development iterations lower hierarchic functions with respect to the higher level in- and output flows can be determined. Equally, TRIZ function models are used to develop solution ideas. Additionally, the depicted function modeling approaches are used for the evaluation of different solution models with help of evaluation methods, i.e. scoring or FMEA.

Accompanying the different modeling approaches in the different application fields hierarchic modeling is used, e.g. function trees or function lists. This kind of modeling allows to structure components, systems, functions or solutions and gives with this a structured overview of the different levels of the system architecture. Depending on the modeling, a bottom-up or top-down approach is applied.

Software/mechatronics: In a wide range of modern products and systems, software plays a decisive role to fulfill the desired functionality. The software, especially in large-scale and complex projects, is created usually in teams rather than by individuals and thus, requires, instead of simple programming, a professional software development, which is the goal of software engineering. Software engineering contains all activities related to the software production, containing the specification, development, validation and evolution of software (Sommerville 2011). In order to consider these aspects at different levels of detail, discuss them with various stakeholders (other software developers as well as non-developers), and facilitate the reuse of software parts for new projects or the evolution of existing systems, model driven software engineering focuses on the creation of models of different parts of the system (structure, behavior) prior to programming (Brambilla et al. 2012). In order to develop and depict these models, the Unified Modeling Language (UML), specified by the Object Management Group (OMG) (Object Management Group 2011b), is a wide spread modeling language in model-driven software engineering. In order to facilitate and accelerate the programming and reduce faults, efforts have been made to generate (parts of) the software code automatically from the models (e.g. to C, or IEC61131-3 Vogel-Heuser et al. 2005).

The strong interactions between software, electrics/ electronics, and mechanics in mechatronic products and production systems requires a more comprehensive view of the system and its material, energy, and information flows. Therefore, the OMG specified the Systems Modeling Language (SysML Object Management Group 2012), which is based on UML and poses a multipurpose modelling language for a wide range of systems. Through profiles or extensions of the meta-model, SysML can be adapted for its specific scope of usage (e.g. SysML4Mechatronics for mechatronic systems Kernschmidt and Vogel-Heuser 2013). The requirement diagram and use case diagram specify the scope and the requirements of the system and stakeholders. The block definition diagram enables a course grained

modeling of the system structure as ‘black box’, which is detailed with connections and flows in the internal block diagram and the parametric diagram. The system behavior is modeled in the activity or state machine diagram, as well as in the sequence diagram for chronological interactions (Object Management Group 2011b). Through the integration of different disciplines, SysML can be used for the identification and analysis of interdisciplinary dependencies in mechatronic systems (Kernschmidt and Vogel-Heuser 2013). For the detailed development of the electrics/electronics usually wiring diagrams or E-CAD are used.

Service: Within service engineering, a service is often structured into three dimensions – structure, process and outcome. The structure dimension defines what is needed in order to provide the service. The process dimension addresses the issue how the service is performed and finally the outcome dimension specifies what the result of the service is. Following this logic in the context of service modeling one can differentiate between resource models, process models and (service-) product models. The resource model deals with the production factors that are necessary for service delivery including human resources, material resources and immaterial resources (Bullinger et al.). Modeling approaches that are often used in this context are for example hierarchy diagrams, organigrams, entity-relationship models, class diagrams or other kind of structural modeling approaches. In contrast to the resource model the process model focuses on the dynamics of a service. It defines which activities have to be carried out in which order and where there are logic branches (Bullinger and Scheer 2003). In service engineering popular languages for process modeling are event-driven process chains (EPC), activity diagrams, petri nets or the business process modeling notation (Krcmar 2009). Product models have been used for physical products for quite a while. A product model is part of the enterprise data model and comprehends all product information and characteristic attributes and data about a product over the whole lifecycle (Bullinger and Scheer 2003). The product model is especially important for modular services as it allows configuring the service offerings according to individual customer demands. Additionally, the service business model as a whole can be modeled using certain ontologies, e.g. the e3-service ontology which describes how value is created through the interaction of different parties in the service delivery process (Zolnowski et al. 2013).

2.2. Methods for interdisciplinary system modeling and information exchange¹¹

In the fields of machine and plant manufacturing, mechatronic systems and PSS various meta-models and modeling methods exist, which address the integration of knowledge of the different disciplines during the development process. Table 9 shows a summary of the conducted literature research. While some approaches propose to use only one integrated model for all disciplines during the development (Ferrarini et al. 2011, Eigner 2012; Anderl et al. 2012; Thramboulidis 2010; Klingner and Becker 2012), other approaches (Drath et al. 2008; VDI/VDE 2005, Shah et al. 2009; Shah et al. 2010) show the necessity to enable a mapping to discipline-specific models.

¹¹ This section is based on preliminary work by Konstantin Kernschmidt

Table 9: Summary of existing approaches for interdisciplinary modeling and information exchange

	Considered disciplines				Modeling aspects			"Development artifacts"	Semantic definition of PSS model elements	(Cycle-relevant) Element-attributes	Mapping to discipline-specific modeling languages
	Mech.	E/E	SW	Service	Structure	Behavior	Flows				
(Drath et al. 2008)	●	●	●	○	●	◐	○	○	◐	○	●
(VDI/VDE 2005)	◐	◐	○	○	○	●	●	○	○	●	●
(Ferrarini et al. 2011)	●	●	◐	○	●	○	◐	◐	○	●	○
(Shah et al. 2009), (Shah et al. 2010)	●	●	●	○	●	◐	○	○	○	●	●
(Anacker et al. 2011), (Gausemeier et al. 2010)	●	●	●	○	●	●	●	●	◐	●	◐
(Eigner 2012), (Anderl et al. 2012)	●	●	●	○	●	●	●	●	◐	◐	○
(Thramboulidis 2010)	●	●	●	○	●	○	◐	○	◐	○	○
(Klingner Becker 2012)	●	◐	○	●	●	◐	○	◐	○	◐	○
(Bochnig 2012)	●	●	◐	●	●	◐	○	◐	◐	●	○

Regarding the considered disciplines of a PSS, namely mechanics, electrics/electronics (E/E), software (SW), and service, up to now an approach, which aggregates the information from all disciplines equally, is missing. Thereby the structure of the system (including the interfaces of the system elements), its behavior (course of activities), and the flows (material/energy/information/control) between the elements should be included.

Furthermore, we analyzed if elements, which have to be considered during the development, e.g. requirements or test cases, but are not part of the final system (hereinafter called "development artifacts"), were integrated. The interdisciplinary semantic definition of the required PSS elements is essential in order to map the elements from/to discipline-specific modeling languages. By defining attributes for each model-element, they can be specified more detailed and each discipline can add its relevant information.

As described in the section above, various modeling approaches exist in the different disciplines involved in the development of mechatronic PSS. An approach that enables an interdisciplinary development process, which satisfies the required integration of all disciplines in the development of PSS and aggregates the information from their different specific modeling languages, is missing. Thus, an integration-framework is needed, which can be used by all stakeholders to define the elements of the PSS and which includes the information from the involved disciplines and on the other hand enables a mapping to the discipline-specific languages.

3. Integration-Framework

3.1 Approach

Interviews with experts from industry have shown that up to now there is a lack of a standardized, computer-aided information flow between disciplines, incompatibilities between the engineering systems and a resulting problem of data inconsistency (Li et al. 2012).

In order to develop a framework for PSS, the challenges imposed by the need for an integrative innovation process that comprises multiple engineering disciplines have to be taken into account. On the one hand, the characteristics of each specific modeling approach should not be restricted, but on the other hand, the connection of the different involved disciplines has to be integrated, as it poses a main benefit of the intended integration-framework. Furthermore, next to the PSS elements the 'development artifacts', such as requirement artifacts of different abstraction levels and representations (Berkovich et al. 2012) should be included in the framework. The framework enables the integration of different discipline specific modeling approaches through the interdisciplinary definition of the system elements before each discipline begins its discipline specific development.

The analytical comparison of modeling languages and approaches, which are used in each engineering discipline, as described in section II.A, showed that the most common modeling construct is the graphical representation of the considered system-view as a combination of nodes, edges and attributes. Thus, in order to fulfill the given requirements and hence improve the interdisciplinary development process of mechatronic PSS, a framework is needed which specifies the required elements of the PSS and its development and which can be mapped to different specific modeling languages. In this way the information of each element can be aggregated from the different specific modeling languages and on the other hand the disciplines are not restricted to use the best fitting modeling language for the specific modeling task.

3.2 Specification of mechatronic PSS

The framework for a crossdisciplinary development specifies the required elements of a mechatronic PSS and their connections. The goal of the specification however is not to define a new modeling language for developing PSS, rather the framework shall be used to aggregate the information of the different specific models, and thus, make them usable for other disciplines, keep the system information consistent and facilitate the reuse of system elements. Furthermore, communication barriers between the involved disciplines are reduced, as the different elements are considered jointly at the beginning of the development process. Each discipline then has to consider to which element in the integration-framework the objects of its specific modeling language correspond to.

According to the Meta-Object Facility (MOF) (Object Management Group 2011a) the specification is divided into the M3-, M2-, M1-Layer (Meta-Meta-Level, Meta-Level, Model-Level) as shown in Figure 19.

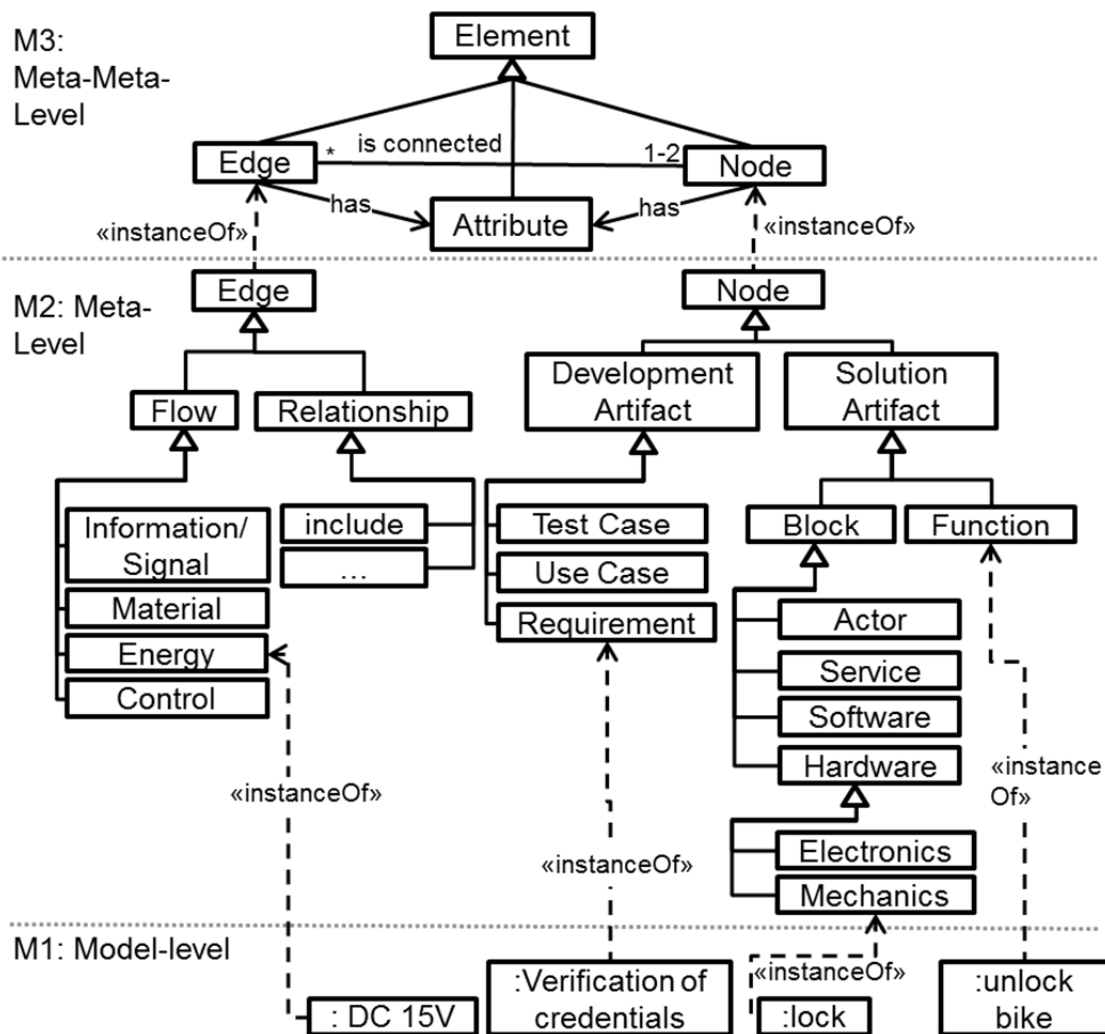


Figure 19: Specification of the elements for the integration-framework

According to the graph-based representation, the framework is defined on the meta-meta-level (M3-layer) as nodes, which are connected by edges. These graphs can be expressed as tuple $G=(V,E)$ wherein the elements of V represent the nodes (also called vertices) and the elements of E the edges (Diestel 2005). Each edge $\varepsilon \in E(G)$ is connected to exactly two nodes $v \in V(G)$ or connects one node with itself. Each ε and v has attributes where all required information of the element is stored (for lack of space the attributes are not shown in Fig. 2.). The most abstract object in the framework is "Element" containing general attributes, such as "ID", "name" etc. Each sub-object inherits the attributes from its higher-level object (depicted through the triangle-symbol in Fig. 2.), e.g. all ε and v inherit "name", "ID" from "Element".

The instances of ε and v define on the M2-layer the "domain model" of a mechatronic PSS and are described in the following: The nodes $v \in V(G)$ can be divided into "Development Artifact" and "Solution Artifact". Development artifacts describe what the final system shall or has to fulfill and how this can be verified. The development artifacts are not part of the actual PSS but are needed during its development. Development artifacts include "Requirements", "Use case" and "Test case". The elements of the PSS itself are summarized as solution artifacts. They contain the building blocks of the system, including hardware

(mechanics, electronics), software, services, and actors (stakeholders interacting with the system e.g. customers), and the specific functions of the system (e.g. the course of fulfilled activities).

In order to define how the nodes interact with each other they are connected with edges either as "Relationship" or as "Flow" from one block to another. Thus, the edges depict on the one hand flow connections between blocks, specifying which "Information/signal", "Material" or "Energy" flows in the system, and on the other hand the edges show logical connections, wherein the "Control"-flow is used for modeling successive steps of activities and the "Relationship"-edge defines dependencies between nodes (e.g. which "Block" satisfies which "Requirement").

These described elements on the M2-layer define the PSS on a high level of abstraction. Each discipline, focusing on certain system aspects, can add different information to the respective elements by defining them in their specific modeling language and mapping them to the framework. Therefore, as defined on the M3-layer, each element contains attributes, which specify the element more detailed. Next to general attributes, as described above, specific attributes were identified for each element. During the development of a specific PSS, attributes, which are of no relevance within the entire system (e.g. if the attribute "transport restrictions" defined for "mechanics" is not required) do not have to be taken into account, while other project specific attributes have to be added to the specification. In the same way flows, relationships, development or solution artifacts can be deleted or further ones can be added to the framework, if required. Thus, in the first step of a PSS-development the specification has to be checked for the specific system and if necessary has to be adapted.

In the M1-layer the actual model is set up. Fig. 2 shows exemplified some modeling elements. The modeling of these elements however is carried out in the different specific modeling-languages, representing always semantically the same defined elements. The attributes of each element are either stored as text or are depicted in the specific modeling language graphically, e.g. in SysML the input and output ports, which are defined in the integration-framework as attributes of the " blocks, are represented graphically in the diagrams, as SysML focuses on modeling connections and flows between the objects of a system.

4. Application Example

In order to convey how the proposed integration-framework can be applied practically we looked at a simplified example from the eBike sharing system "PSSycle" which was recently developed as a demonstrator within the context of the research project SFB768 (see acknowledgment). The idea behind this PSS is to sell mobility instead of the eBike itself and to offer the user additional features like navigation. Attached to the handlebar of the eBike, there is an onboard computer with a touchscreen, which serves as an interface to the user. Having registered for the service, the eBike can be rented by simply entering one's credentials and the integrated lock will open automatically. This process is modeled by means of an event-driven process chain (EPC) as it is often used in service engineering (see Fig. 3; for an explanation of the EPC modeling elements see Krcmar 2009).

On the other side, the system-structure has been modeled in a SysML internal block diagram (see Object Management Group 2011b for modeling elements). The structural model shows how the components of the system are linked with each other and how they interact. For example, the onboard computer and the back office server communicate via HTTP messages while the microcontroller interacts with the electric lock by applying a voltage.

In order to integrate the information of both models in the scheme of the framework the different elements of the EPC and the SysML model are mapped to the elements of the framework (Fig. 2). As can be seen in Figure 20, certain elements of the PSS are modeled in both specific modeling languages, e.g. the onboard computer. While the EPC defines more specifically which events lead to the functions, executed on the onboard computer, the structural SysML-model specifies the ports of the onboard computer and how it is connected to other elements

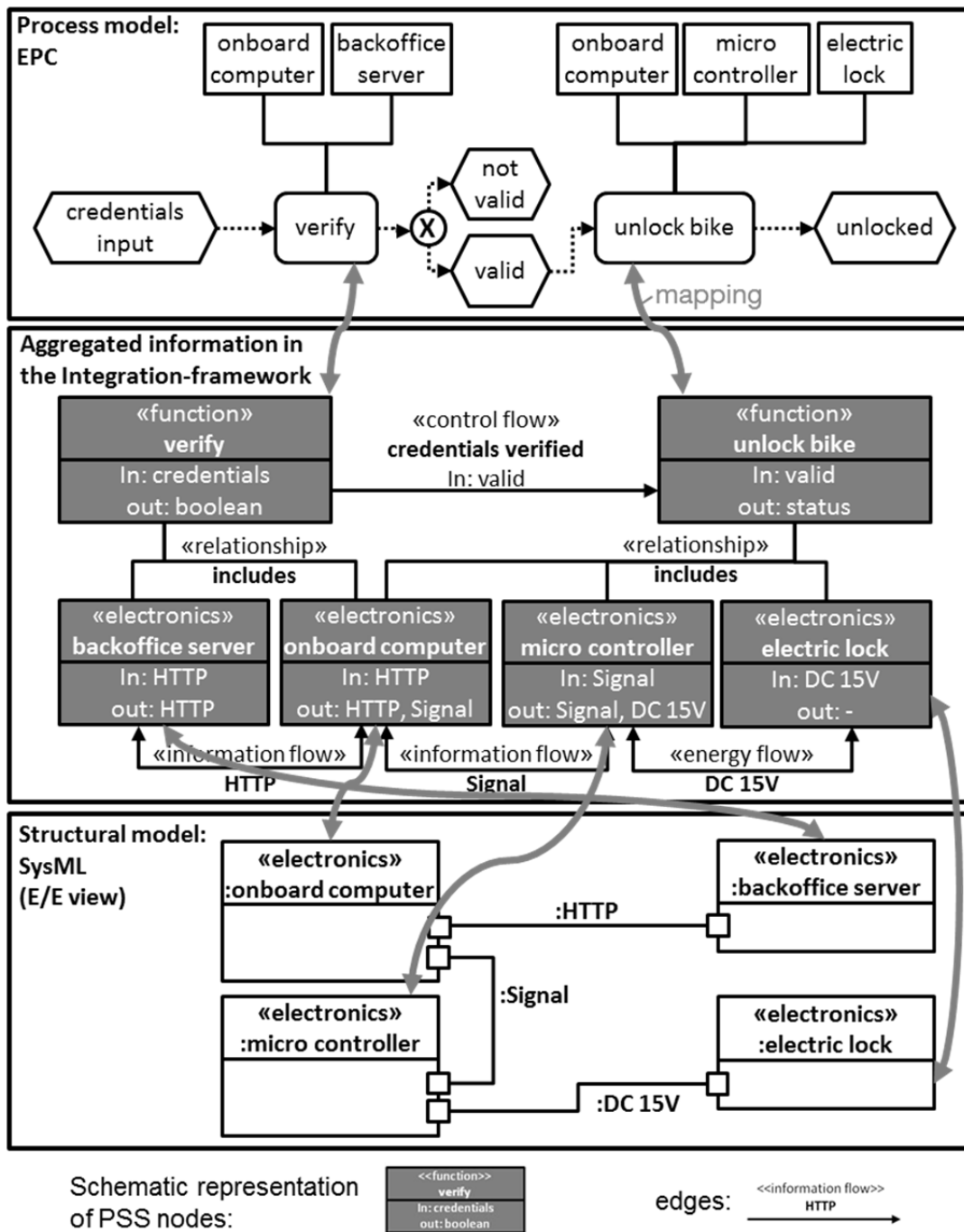


Figure 20: Excerpt of the mapping concept for the eBike-sharing-system

In each case the type of the element to which it was mapped is denoted by double brackets. E.g. a control flow from the EPC is mapped to an edge of the type «control flow». Thus, by using the framework, the jointly used modeling elements only have to be modeled once and can be used in other modeling languages.

5. Conclusions and Outlook

In this paper an integration-framework for the interdisciplinary development of PSS has been presented. By analyzing the different views on a PSS in the common modeling approaches and languages of the different involved disciplines, mechanics, electric/electronics, software, and service engineering, a specification of PSS was derived. The specification contains the jointly required modeling artifacts (nodes) and defines how they are connected with each other (edges). During the modeling task each discipline adds within their modeling language relevant information to the PSS-elements. By mapping the specific elements on the elements of the integration-framework, this information can be used also by the other involved disciplines. However, currently this information has to be transferred manually from the specific tools. Therefore, in future research the integration-framework will be implemented in a neutral data-format, such as XML, and an automatic transformation from the specific modeling-languages, using a suitable transformation method (Czarnecki and Helsen 2003), will be established. The framework can then be evaluated through the application of a use case, including different specific modeling languages. Furthermore, an extension of the framework to include further aspects of the PSS development, such as e.g. production process planning, will be considered. Through the usage of the framework an efficient interdisciplinary PSS development is achieved. Enabling a more sophisticated system view for all stakeholders, reducing faults, and minimizing required iteration steps, leads to significant time savings during the development and thus, to a faster time to market of the developed PSS.

Acknowledgment

We thank the German Research Foundation (Deutsche Forschungsgemeinschaft – DFG) for funding this work as part of the collaborative research centre ‘Sonderforschungsbereich 768 – Managing cycles in innovation processes – Integrated development of product-service-systems based on technical products’ (SFB768).

Publication 6

Supporting the cross-disciplinary development of product-service systems through model transformations

*Thomas Wolfenstetter^a, Konstantin Kernschmidt^b, Christopher Münzberg^c, Daniel Kammerl^c,
Suparna Goswami^a, Udo Lindemann^c, Birgit Vogel-Heuser^b, Helmut Krcmar^a*

^a Chair for Information Systems Technische Universität München Munich, Germany {thomas.wolfenstetter, suparna.goswami, krcmar}@in.tum.de	^b Institute of Automation and Information Systems Technische Universität München Munich, Germany (Kernschmidt and Vogel-Heuser 2013)@ais.mw.tum.de
^c Institute of Product Development, Technische Universität München, Munich, Germany {christopher.muenzberg, daniel.kammerl, lindemann}@pe.mw.tum.de	

Abstract

During the development of product-service systems (PSS), various artifacts are modeled by the different involved disciplines, e.g. mechanics, electrics, electronics, software and services. Each of these artifacts represents different aspects of the PSS as a whole. In order to reuse the respective information that are contained in each artifact but which are represented using different ways of modeling, transformations are needed. In this paper we present a conceptual methodology how the relevant PSS elements and their attributes can be transformed from one specific language to another, in order to facilitate the cross-disciplinary use of model-based information during the development process of mechatronic PSS.

Keywords – model transformation, product-service systems, cross-disciplinary development

1. Introduction

Producing companies increasingly shift from offering traditional products towards more sophisticated and integrated solutions that better meet the customers' needs. In doing so, these companies try to achieve competitive advantages. Such integrated offers, called product-service systems (PSS), contain product components (including mechanics, electrics/electronics and software) as well as service components (Schenkl et al. 2013). For the development of such integrated solutions, adequate cross-disciplinary model-based engineering is essential, as an entire PSS hardly can be developed by a single discipline. Collaborative modeling however, poses several challenges. On the one hand each discipline uses its own specialized modeling approaches to depict their specific view on the system. On the other hand, the system information is distributed among the different disciplines and their models. However, due to dependencies between the different solution components, inconsistencies between the different models can arise. Additionally, the traceability of artifacts through the different disciplines lacks with current methods. During the entire PSS-lifecycle (containing the planning, development, production, delivery and decomposition Hepperle et al. 2010) a strong interaction of the service and product lifecycle exists. In the development phase, knowledge of the different disciplines, e.g. requirements or model elements, has to be shared. To enhance cross-disciplinary knowledge sharing and to overcome the described challenges a concept for a PSS integration framework (PSS-IF), to enable the cross-disciplinary development of PSS, has been developed (Kernschmidt et al. 2013). In this paper the PSS-IF is extended and a transformation methodology between discipline specific models using the PSS-IF is shown. The open structure of the framework allows the integration of further modeling languages at a later point of time, which makes it flexible for modeling languages that have not been considered yet and reduces the barriers for introducing the framework into the existing development process. Through the application of the framework a sophisticated PSS development can be achieved.

The structure of the paper is as follows: In the next sections, existing model-based engineering approaches and model transformations are discussed. Subsequently, the architecture of the PSS-IF is introduced. In section 5 the transformation process using the framework is shown. Finally, the paper is concluded and an outlook on future work is given.

2. State of the Art

The development of PSS requires a tight collaboration of the disciplines mechanical, mechatronic and service engineering. In literature and practice a variety of modeling approaches for the different involved disciplines exist, focusing on different aspects, e.g. the analysis of systems, development of functions and solutions, the support of process management, or detailed discipline specific design. For mechanics (mechanical engineering) literature suggests several modeling approaches, which are applied for designing mechanical products throughout the complete product development process (Ponn and Lindemann 2008). However, today's systems are mostly no longer pure mechanical, but the percentage of electrical/electronic and especially software parts in a system increases. In mechatronic engineering the disciplines of informatics, mechanical and electrical/electronic engineering

are combined, so specific tools for e.g. mechanical or software engineering are not sufficient. The combination of different designing disciplines longs for a more comprehensive view of the system. In order to integrate the specific knowledge, model-based systems engineering was introduced and the Systems Modeling Language (SysML, Bernard et al. 2010) developed. SysML is based on UML, the standard modeling language in software development. As SysML was developed for a wide range of systems it can be adapted to its specific scope using profiles, e.g. SysML4Mechatronics (Kernschmidt and Vogel-Heuser 2013).

Table 10: Cross-Domain Modeling Approaches

Ferrarini et al. (2011)	specification and design of automated production systems, methods for model-based manufacturing plant specification and design
Boching (2012)	framework for IT-based support of planning and developing PSS
Maussang et al. (2009)	step-by-step PSS design process to capitalize the information during the design process
Anderl et al. (2012)	methods, processes and IT-solutions for interdisciplinary virtual product development
Thramboulidis (2010)	integrated framework for the construction of mechatronic systems using SysML
Becker and Pfeiffer (2008)	holistic modeling method for PSS, including product and service components and their interdependencies
Anackeret al. (2011)	specification technique to design advanced mechatronic systems
Gausemeier et al. (2010), Pahl et al.(2006)	specification technique for PSS, procedure model and tool support
Eisenbart et al. (2012)	matrix-based representation of cross-domain functional models
Sakao et al. (2009a)	consistent depiction of physical and human processes with a focus on customer's added value

To acquire new customers, industry tends to open up new business strategies. So there is a change from mainly sales oriented strategies to value delivery. This is realized by adding service to an existing product portfolio. This way, for designing these PSS an additional discipline, service engineering has to be incorporated in the design process. Applied modeling approaches in service engineering are for example service blueprints, e³-value and i* (Shostack 1982; Gordijn and Akkermans 2003; Yu 2009).

Hardly one discipline-specific modeling approach can address all identified perspectives, because each discipline has its own focus (Eisenbart et al. 2012; Eisenbart et al. 2013). Hence, in the fields of machine and plant manufacturing, mechatronic systems and PSS various modeling methods and languages exist, which address the integration of knowledge of the different disciplines throughout a development process. These cross-discipline modeling approaches concentrate information of different design disciplines. They allow for merging discipline-specific models in order to grant an overall system view. Cross-discipline modeling

offers improved actuality, the integrity of the overall model and a higher consistency (Becker et al. 2010). In comparison to discipline specific approaches these models usually work on a higher level of abstraction. In literature a great number of approaches for cross-discipline modeling can be found. Table 1 gives an overview of these approaches. These cross-discipline modeling approaches have different foci, so they support the project management, the planning or the design phase, because one complete model covering a large number of modeling perspectives can quickly become confusing (Eisenbart et al. 2013).

Discipline-specific as well as cross-discipline modeling approaches have advantages the other kind cannot offer. Based on the existing approaches a list of the characteristics of discipline-specific and cross-discipline models is given in Table 11.

Table 11: Characteristics of Discipline-Specific and Cross-Discipline Modeling

discipline-specific	cross-discipline
Possibility to display discipline-specific aspects	possibility of tracing artifacts across discipline borders
higher level of detail	display of domain interconnections
higher information level	consolidated/ cross-disciplinary modeling base
specialized software tools	common language needed
problem specific model	reduced communication barriers
	holistic understanding
	integration among all domains

(Eisenbart et al. 2012) state that an approach that enables a cross-disciplinary development process, which satisfies the required integration of all disciplines in the development of PSS and aggregates the information from their different specific modeling languages, is missing.

So we developed an integration-framework which can be used by all stakeholders to define the elements of the PSS, includes the information from the involved disciplines and also enables a mapping to discipline-specific languages (Kernschmidt et al. 2013). In this approach the advantages of discipline-specific and cross-discipline modeling approaches can be combined. In order to utilize such an approach, model transformations from the different DSLs are required. Therefore, in the next section the different types of model transformations are described and evaluated.

3. Model Transformations

In general, several types of graph-based model-to-model transformation methods can be considered for the intended purpose. We roughly categorize them into direct and indirect transformation methods and additionally regarding their flexibility regarding the syntax of a specific data format. Figure 21 gives an overview of the different possible transformation approaches.

Direct transformation methods define rules for the transcription from source to destination discipline-specific language (DSL) directly without producing an intermediate result. Direct syntax-dependent transformations require both DSL to use the same technical space, e.g. the Extensible Markup Language (XML). Syntax-independent direct transformations are not coupled to the technical space of a particular DSL. They can be realized by parsing the source model into the syntax of the target language's technical space and then mapping it onto a structure conforming to the target DSL's meta-model (Mens and van Gorp 2006). Direct transformation methods have the advantage that they can be defined to enclose the maximal possible transmittable detail from one DSL to another (Varró and Pataricza 2004). However, such approaches impose limitations to the entirety of languages that can be supported, due to their binding to a concrete syntax. Syntax-independent direct transformations resolve this issue by abstracting the transformation description from the technical space of the language involved. Still, all direct transformation methods require an explicit implementation for each pair of source and target DSLs. To address this issue of exponentially growing complexity, indirect transformation methods can be used, since they rely on a stable and well defined intermediate language (IL). A particular transformation between two DSLs is performed by first transforming from the source DSL into the IL and then transforming from the IL to the target DSL (Czarnecki and Helsen 2006). Once again, such methods can be either fixed or flexible. In the case of a fixed IL, its abstract syntax is directly implemented as a data structure. Thus, the IL is described directly with the vocabulary of the programming language in use. However, this approach becomes costly if the evolution of the IL is taken into consideration. In the case of flexible IL transformations, the IL is not bound by concepts of the underlying programming language, but rather is only expressed in those concepts. Consequently, the IL can be defined on a level of abstraction on which evolutionary changes of the IL which merely imply a structural change can be implemented by configuration. Flexible IL transformation therefore provides a viable, flexible and powerful approach with minimum costs for the consideration of additional DSLs (Bézivin et al. 2006).

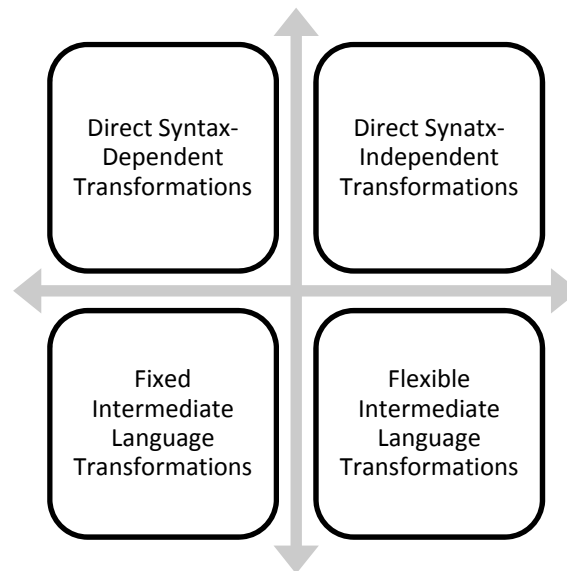


Figure 21: General types of transformation possibilities

4. Basic Concepts of the PSS-IF

In order to define an IL for the cross-disciplinary development of PSS those elements and their connections, which are required by the different involved disciplines, have to be specified and form the basis of the PSS-IF (Kernschmidt et al. 2013). In the following the necessary concepts for the transformation from one DSL to another, including all relevant information, are described.

PSS-IF meta-model

On the abstract level the meta-model defines the language in which the elements of a PSS-IF model are described. Figure 22 shows the taxonomy of elements that are part of PSS-IF. The PSS-IF meta-model basically consists of node types and edge types, both of which can have attributes. Node types are used for the description of the different entities in a model, whereas edge types are used to define the associations (relationships and interactions) between different types of nodes. Associations between node types are defined through connection mappings, which are always bound to a certain edge type, i.e. an edge type can have many connection mappings, allowing it to associate different pairs of node types. A connection mapping is an association assigned to a particular edge type, which includes an incoming and an outgoing node type. For instances, "State" and "Activity" can be connected by an edge of type "ControlFlow".

Furthermore, inheritance can be defined among the node and edge types. The root node and edge types are also the roots of the inheritance hierarchies for nodes and edges within the meta-model. Thus, it is guaranteed that the set of attributes provided above is automatically defined for all instances of both node and edge types. If a node or edge type inherits from a non-root node or edge type, the attributes are inherited transitively, together with all attributes of all ancestors throughout the generalization closure. In a PSS-IF model each node is an

instance of a node type and each edge is an instance of an edge type in the PSS-IF meta-model.

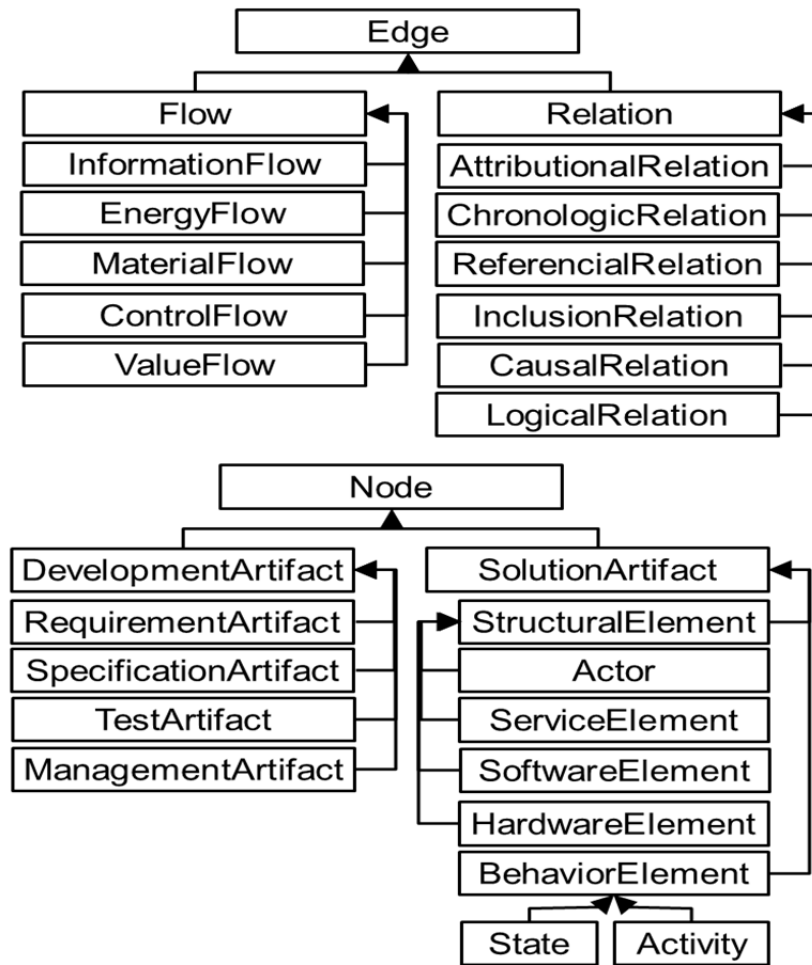


Figure 22: Excerpt of the PSS-IF meta-model

Viewpoints

For each DSL, a viewpoint is defined which captures only those parts of the PSS-IF meta-model that can be represented in the corresponding DSL. Viewpoints define how the PSS-IF meta-model is seen from the perspective of a certain DSL. More specifically, a viewpoint represents the meta-model of a DSL using the vocabulary, namely the node and edge types, specified in the PSS-IF meta-model. For this purpose, viewpoints are defined by applying a certain set of atomic transformation operations to the meta-model of a DSL. These atomic transformation operations comprise e.g. renaming, merging or splitting node or edge types. Since, any transformation results in a viewpoint, they can be applied consequently, each one operating on the result of the previous. Furthermore, viewpoints are defined in such a way that they can be used for both reading from and writing to a model.

Generic graph

The generic graph component of the PSS-IF is a simple undirected graph consisting of nodes and edges, which can have string-named attributes with string values, as well as the name of their assumed type. In this sense, the graph is an untyped and unstructured equivalent of a PSS-IF model. In the transformation process, the generic graph is used as an intermediate format, separating the concrete syntax and the abstract syntax of each supported language. The concrete syntax is handled by an I/O mapper, while the abstract syntax is defined by the viewpoint and is processed by a model mapper. By using a generic graph, it is possible to separate the handling of concrete and abstract syntax of each language from each other. Therefore, the concrete syntax is handled by a specific utility, which rather relates to the syntax than to the language.

I/O mappers

The I/O mapper components are responsible for the serialization and de-serialization of the generic graph. In this sense, they have the task to produce an abstract representation of the tool specific data structure in any particular DSL. As the output of I/O mappers is a generic graph representation, they can be used for more than one DSL if the same data format is used.

Model mappers

The task of the model mappers is to transform the generic graph into a PSS-IF model and back under the provision of a corresponding PSS-IF viewpoint. In this sense, model mappers handle the translation between the abstract syntax of the external representation and the abstract syntax of the DSL's viewpoint in PSS-IF. In the simplest case, a particular model mapper simply uses the provided viewpoint to directly transfer information between a model and a graph, processing all nodes, edges and attributes. In more complex cases, pre- or post-processing of the graph is necessary to receive a structure compatible with the viewpoint defined for the particular language.

The data from an external representation is read into a generic graph and then processed with the PSS-IF viewpoint of the respective DSL. In the same manner, when exporting a view, a model is converted to a generic graph in accordance with the DSL's viewpoint and the generic graph is then serialized in accordance with the specifics of the concrete syntax of the DSL.

Finally, mappers encapsulate the whole transformation process between the PSS-IF meta-model and a corresponding model. A mapper thus offers two functionalities, one for reading a model from an external representation and one for writing a model. Each DSL has its own mapper and each mapper combines, in the appropriate order, the DSL's viewpoint creation, data transformation, any pre- and post-processing strategies, and the correct serialization utility.

5. Transformation Process

As described above the PSS-IF enables the transformation from one DSL into another. The transformation process between two DSLs A and B is conducted as shown in Figure 23. A PSS-IF model instance is obtained from an external representation by transcoding the external representation (Model M in DSL A) into a generic graph, obtaining the viewpoint of the current DSL and finally mapping the generic graph to a model in accordance with the viewpoint. Symmetrically, a file is generated from a PSS-IF model by first obtaining the viewpoint for the current DSL, using it to translate the model into a generic graph and then serializing the graph using the corresponding I/O mapper. Each transformation process invokes both types of mappers. First, source data is de-serialized and transformed into a PSS-IF meta-model conformant representation. Second, the obtained model is written into an external representation.

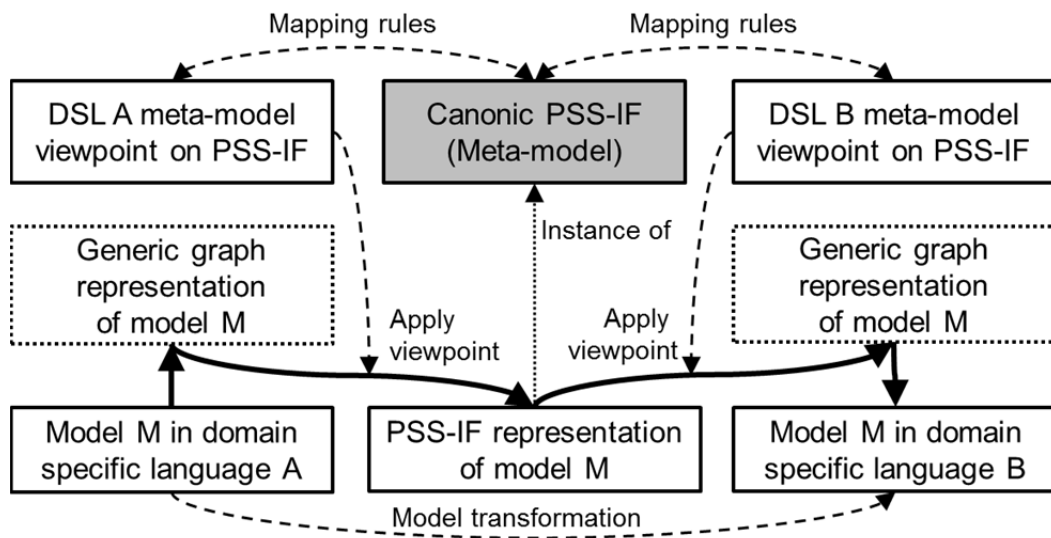


Figure 23: Schematic representation of the transformation process

6. Conclusion and Outlook

In this paper a conceptual methodology for model transformation using an integration-framework for the interdisciplinary development of PSS has been presented. By transforming the specific models of a PSS that are created by the different involved disciplines (mechanics, electrics/ electronics, software and services) the information and modeling artifacts can be (re-)used by the different disciplines. The presented framework-based transformation approach can handle both, structural as well as behavior models. For example, a mechanical engineer may model basic features and properties of the hardware components of a PSS. A service engineer will then build on this information to create business process models that define how customers interact with the system. And finally, a software developer enhances the overall system model by specifying which software components are used in certain activities of the business processes.

By using the framework, efficient interdisciplinary PSS development is supported. Enabling a more sophisticated and comprehensive system view for all stakeholders, reducing faults, and minimizing required iterations, leads to significant time savings during the development and thus, to a faster time to market of the PSS. In order to enhance the development of PSS further, a formal definition of the dependencies and relationships of the modeling artifacts of all involved disciplines is necessary. The PSS-IF and the presented transformation approach form a first step for enabling synchronization of different discipline-specific models. We will evaluate our approach regarding semantic and syntactic correctness as well as completeness (Mens and van Gorp 2006) by implementing a proof-of-concept prototypical tool that allows to integrate various DSL. Also, further research will be conducted to provide model management including appropriate formal methods.

Acknowledgment

We thank the German Research Foundation (Deutsche Forschungsgemeinschaft – DFG) for funding this work as part of the collaborative research center ‘Sonderforschungsbereich 768 – Managing cycles in innovation processes – Integrated development of product-service-systems based on technical products’ (SFB768).

Publication 7

Towards a Requirements Traceability Reference Model for Product Service Systems

Thomas Wolfenstetter^a, Simon Bründl^b, Kathrin Füller^a, Markus Böhm^a, Helmut Krcmar^a

^a Chair for Information Systems Technische Universität München Munich, Germany {thomas.wolfenstetter, fuellerk, markus.boehm, krcmar}@in.tum.de	^b Institute for Information Systems and New Media, Ludwig-Maximilians-Universität München Munich, Germany bruendl@bwl.lmu.de
--	---

Abstract

Differentiation opportunities for providers of traditional products and services are declining due to increasing global competition. As a result, companies are transforming into solution providers offering integrated bundles of products and services, so called Product Service Systems (PSS). The development of PSS requires intense collaboration of different disciplines (e.g. mechanical, software or service engineering) to produce a solution that fits the customers' needs. However, each discipline relies on specific engineering models, produces heterogeneous artifacts and uses different languages to describe them. For successful integration of the different PSS components, developers need a joint system model that allows understanding interdependencies and tracing the evolution of artifacts. In this context traceability is of utmost importance since requirements are specified solutions independent across different disciplines. Our research addresses this challenge by proposing a corresponding reference model. Based on a literature analysis, modeling workshops with experts in various engineering disciplines have been conducted. Integrating the insights from literature and workshops, a reference model has been iteratively developed and evaluated. This model contributes to research on cross-domain traceability of requirements and other artifacts. From a practical perspective, the reference model can be used to develop tools supporting collaborative PSS engineering and improving cross-disciplinary understanding.

Keywords

Requirements Traceability, Reference Model, Product Service Systems

1. Introduction

Companies in the manufacturing sector increasingly struggle to differentiate themselves from their competitors as in a highly globalized world the design and quality of products are often not enough to realize competitive advantages. A product-centric approach is therefore no longer sufficient to successfully and sustainably generate value for the customers. However, companies can create value by adding services to products. The increased service character of the product results in a stronger differentiation from competitors and therefore in an increased competitiveness in the market (Tukker 2004).

Consequently, many manufacturers expand their portfolio from products to so-called Product Service Systems (PSS).

According to Baines et al. (2007), a PSS combines tangible product and intangible service components. The trend towards PSS is mainly customer-driven. There is a growing customer demand for complete solutions to individual problems and needs, instead of solely buying goods or services (Sawhney et al. 2006). A PSS therefore strives to deliver a solution that specifically targets the customer's needs. To achieve this objective, the components of a PSS need to be adapted to individual customer needs which includes intense collaborations and integration of the customer in the design and development process. Additionally, the design of PSS includes various disciplines, such as product, software and service engineering (Wolfenstetter et al. 2014).

Further, the different components are likely to have different life cycles, facing companies with the challenge of managing inherent dependencies between the different components. Yet, cross-disciplinary integration does usually not take place in practice. The development of products and service is carried out in separate processes using discipline-specific models (Tukker 2004). Existing theory on development processes is highly specialized to the respective discipline, making integration and cooperation between the different disciplines a difficult task (Berkovich et al. 2012). In the development of PSS modifications to the product eventually lead to modifications to the service bundle. Vice versa, service related changes may lead to drastic changes to the product. However, the controllability of these interdependencies is absolutely essential. Therefore, a multidisciplinary approach for PSS development is required (Mont and Tukker 2006). In practice, however, the development of products and services is often carried out independently, which leads to an inadequate consideration of the mutual influences of product and service components. For successful development of a PSS it is not enough to understand the characteristics of these three disciplines, it is also necessary to address interfaces and interdependencies between the disciplines (Herzfeldt et al. 2011).

In conclusion, to design a seamlessly integrated PSS developers require an integrative traceability model uniting the different disciplines and their requirement and design artifacts. Artifacts of one discipline thus need to be linked to the artifacts of other disciplines in order to guarantee full integration and traceability. An important building block in this context is a traceability reference model for PSS. Reference models are an abstract framework to define a

set of concepts and to indicate the relationships among them. A reference model is influenced by the environment, and describes entities and relationships. A general reference model for requirements traceability has been proposed by Ramesh and Jarke (2001). This model, however, originated in the field of software traceability and thus takes a rather discipline-specific perspective on the traceability challenge. Although a number of other, mainly discipline-specific reference models and traceability information models have been proposed within the research community, none of these models focuses on PSS and inter-disciplinary development processes (Cleland-Huang et al. 2014). Although some might argue that the challenges related to traceability are largely the same in each context, we believe that a comprehensive traceability reference model that takes into account the special characteristics of integrated systems that involved multiple disciplines is of great value for the development of PSS. While traditional approaches mainly focus on providing an evidence for the customers that their requirements are fulfilled by the product as it is delivered, a traceability reference model for PSS needs to foreground continuous improvement or customization of the solution to changing customer needs, which is the key value proposition of PSS. In this regard we are among the first to explicitly consider the need for traceability of the service components.

2. Research Design

As reference models derive their relevance from the abstraction of best practice approaches, the combination of theoretical foundations and experiences from practice ensures that the reference model captures all types of artifacts that are relevant for tracing the development of a PSS. Thus, to develop the traceability reference model we followed a three-step approach, consisting of an initial literature review, subsequent workshops with domain experts and a ‘paper’-based evaluation.

The objective of both, the literature review and the expert workshops, was to identify relevant artifacts for the traceability reference model as well as the relevant relationships between artifacts. Artifacts can be defined as encapsulated information objects describing real world entities or abstract concepts that are relevant in the system lifecycle. In terms of model-based design and development approaches, an artifact is considered as a structured abstraction of model elements (Fernández et al. 2010). In principle, artifacts arise as a result of an activity in the development process, e.g. requirements elicitation or change management (Berkovich et al. 2012). By focusing on the result of a development activity that is condensed in form of an artifact it is possible to abstract from the different discipline-specific methods and processes that can be employed to provide the artifact (Fernández and Wieringa 2013).

For the literature review we searched in domain-specific literature databases, including Google Scholar, IEEE Xplore, and ACM digital library, to identify papers on data models for requirements traceability in the areas of PSS, product, software and service development. We identified 61 relevant papers from which we derived basic requirements engineering artifacts and their relationships. Based on our literature review, we derived a preselection of potential artifacts from the identified generic approaches for traceability.

In the second step, we conducted 9 expert workshops to gather practical experience in PSS design and development. The goal of these workshops was 1) to evaluate the importance of the pre-selected artifacts in the various domains from a practical perspective, and 2) to identify additional artifacts that are important in the context of traceability for PSS and 3) to specify the type of relationships (trace links) between those artifacts (Spanoudakis and Zisman 2005).

We acquired experts from requirements engineering, change management, product development, engineering management, information systems, systems engineering, software engineering as well as production and manufacturing technology. The workshops were performed individually with one expert at a time in order to obtain an independent picture for each domain. Each of the experts was given a list of domain-specific artifacts that were identified through the literature review. First, each expert evaluated whether or not a certain artifact is relevant from a traceability perspective. Second, they were asked to specify additional artifacts that should be included in the requirements traceability reference model. Third, all artifacts that were considered relevant were entered as nodes into a modelling tool (MagicDraw). On this basis participants should then specify the relationships between artifacts from their point of view. By doing so, each workshop resulted in a discipline-specific traceability model that reflects the respective expert's point of view.

These nine discipline-specific models were then merged into a comprehensive traceability reference model. For this purpose, similar artifacts were grouped and, if possible, mapped onto the generic artifacts specified in the PSS integration framework by Kernschmidt et al. (2013). In a third step, the resulting reference model was then circulated among the participating experts. The experts were asked to evaluate the reference model and provide their feedback. Seven experts approved the reference model directly, two suggested minor extensions which were then incorporated. The resulting integrated traceability reference model is presented in section 3. To improve readability, it is split into eight sub models.

3. Results

3.1. General Model Constructs

Based on the literature review, we concluded that each model that is used in the development of PSS can be represented as a graph. All of the experts that we consulted agreed on this perception. Furthermore, most of the modeling languages that are commonly used in product, software, service or systems engineering, such as the Unified Modeling Language (UML), the Systems Modeling Language (SysML) or the Business Process Modeling Language (BPMN) use a graph representation to describe the structure or behavior of the regarded system.

Following (Kernschmidt et al. 2013) the basic meta-model construct of such graph representations is an *Element*. *Elements* can be separated into *Nodes*, *Edges* and *Attributes* (see Figure 24). *Attributes* are used to capture the properties of *Nodes* and *Edges*. *Nodes* can be further categorized into *Development Artifacts* and *Solution Artifacts* and *Stakeholders*. *Development Artifacts* represent rather abstract artifacts, such as documents, that are merely used during the development of a PSS. *Solution Artifacts* on the other hand comprise all

structural, behavioral or functional entities that the PSS consists of. *Stakeholders* represent the different types of individuals that are involved in the development of the PSS or that are affected by the PSS in any life cycle stage and whose requirements thus need to be considered.

Edges can be further divided into *Flows* and *Relationships*. *Flows* denote some kind of transfer between two *Solution Artifacts*. They can be further categorized into *Control Flows*, *Information Flows*, *Energy Flows*, *Material Flows* and *Value Flows*. *Relationships* on the other hand can connect any types of *Nodes*. Here, we differentiate between *Inheritance*, *Inclusion* (e.g. <<contains>>, <<refines>>), *Referential* (e.g. <<has>>, <<relates to>>), *Chronologic* (e.g. <<evolves to>>) and *Causal* (e.g. <<causes>>, <<creates>>).

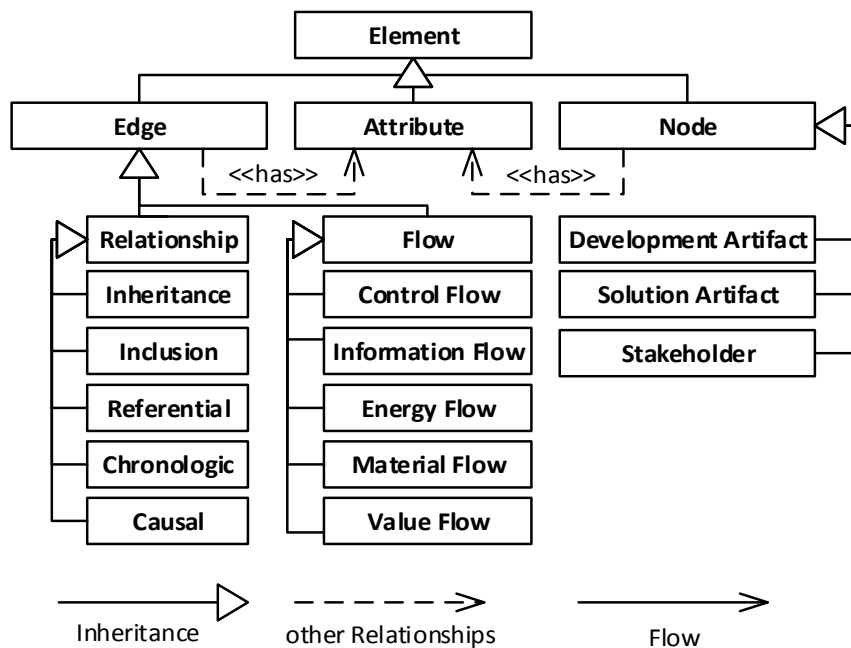


Figure 24: General Model Constructs

3.2. Development Artifacts

Development Artifacts capture information that is relevant to the development of a PSS, but that is not part of the final solution (functional, behavioral, or structural components of the PSS). As illustrated in Figure 25, *Development Artifacts* can be grouped into *Requirements*, *Specification Artifacts*, *Test Artifacts*, *Production Artifacts* and *Management Artifacts*.

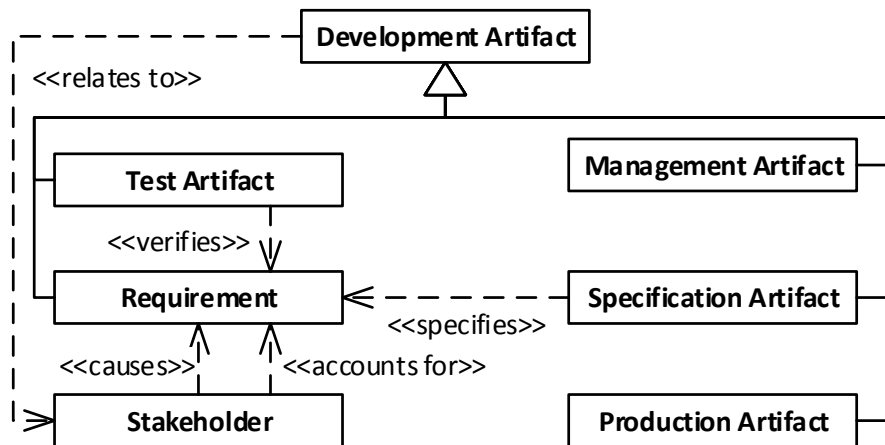


Figure 25: Development Artifacts Submodel

A *Requirement* is a container for meta-information about a certain requirement, such as ID, title, version or priority. The requirement itself, however, is described by a *Specification Artifact* (c.f. Section 3.4). Each *Requirement* should be verified by a *Test Artifact* that documents information about the criteria to test a *Solution Artifact* as well as test results and additional information. In the development of PSS any kind of verification procedure can act as a *Test Artifact*, e.g. simulations, physical experiments, or manual evaluation by a *Stakeholder*. In this regard a major challenge in PSS provision is to verify whether the key value propositions to the customer are fulfilled, since service can only be evaluated at the moment it is performed. Therefore, customer experiences constitute an important *Test Artifact*. In addition to that, *Requirements* are related to specific *Stakeholders*. Each *Requirement* can be connected via <<cause>> relationships to a certain *Stakeholder* denoting that the *Stakeholder* is the source of that *Requirement*. Another possible relation is <<accounts for>>. This relationship between a *Requirement* and a *Stakeholder* implies that the *Stakeholder* is responsible for the *Requirement*. Furthermore, some *Development Artifacts* may <<relate to>> certain *Stakeholders*, e.g. a *Production Artifact* <<relates to>> the production manager in charge.

3.3. Generic Stakeholders

In the development of a PSS or during service provision, a great number of *Stakeholders* are involved (see Figure 26). The complex network of stakeholders that need to be considered along the entire lifecycle is a challenge that is especially prominent with PSS for several reasons. Firstly, PSS do not only constitute a new product but in fact a new business model that impacts the entire organization of the PSS provider. Therefore, the requirements and capabilities of each department in the organization have to be considered. Secondly, when selling traditional products, the ownership and consequentially most responsibilities lie with the customer. Thus, the customer would have to deal with stakeholders that are merely relevant during usage, maintenance and disposal. With PSS the responsibility remains with the provider along the entire lifecycle. Thirdly, the PSS provider does not only have to

consider the stakeholders of the core product but also stakeholders of every service component offered in a PSS.

Overall we can differentiate between multiple generic types of *Stakeholders*. *Internal Stakeholders* refer to the different organizational units of the PSS provider, such as *Sales & Marketing*, *Manufacturing*, *Procurement*, *Disposal* or *Service Provision*. A big challenge is that service engineers are rather market- and customer oriented generalists while mechanical engineers and software developers are more technically oriented. To foster mutual understanding it is therefore crucial to be able to trace back any artifacts created in the development process to the person in charge. In contrast, *External Stakeholders* are not part of the organization. *External Stakeholders* can be categorized in e.g. *Society*, *Law & Regulation*, *Standards*, *External Systems*, or *Competitors*. In the development of PSS especially *Customers* and *Users* are a key source for *Requirements* since PSS providers are generally interested in stable and long-term customer relationships in order to minimize setup costs. Further, the frame conditions imposed by *Value Creation Partners*, such as suppliers or logistics providers that are involved in service delivery have to be considered. The relevance of value chain requirements is enhanced by the fact that while the supply chain of traditional products predominantly flows in one direction while PSS providers are faced with closed-loop supply chains. This means they have to manage the return, refurbishment and replacement of expendable components of the PSS along the value chain. While *Internal Stakeholders* are primarily responsible for realizing certain PSS requirements or components, *External Stakeholders* are in most cases the source of *Requirements* and are involved in the value creation process (Mont 2002).

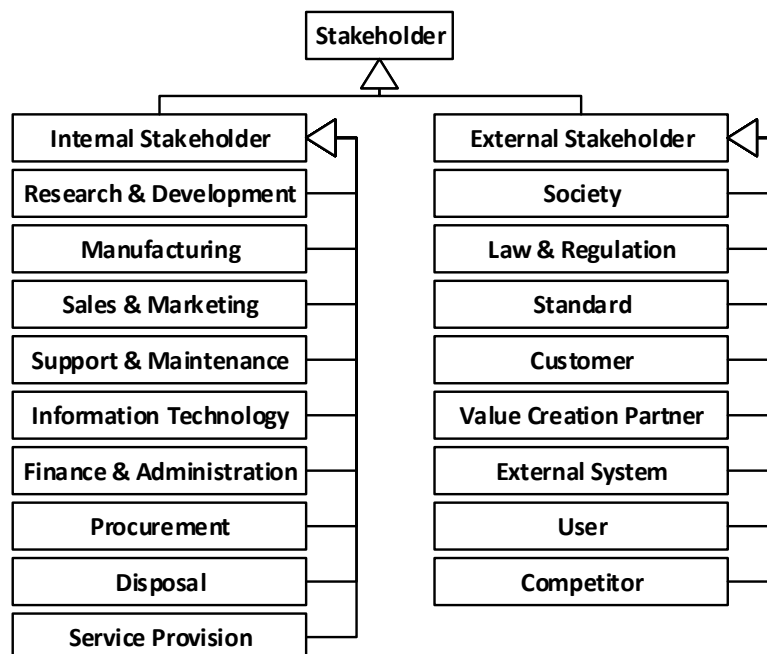


Figure 26: Generic Stakeholders Submodel

3.4. Requirements

According to Berkovich et al. (2012), *Requirements* of a PSS can be categorized based on the level of abstraction (see Figure 27). Unlike the requirements process in product development processes, the requirements elicitation for PSS usually does not start from the intended functionality or characteristics of the product. Instead, it is necessary to identify the general customer *Needs* and define the overall *Business Goals*.

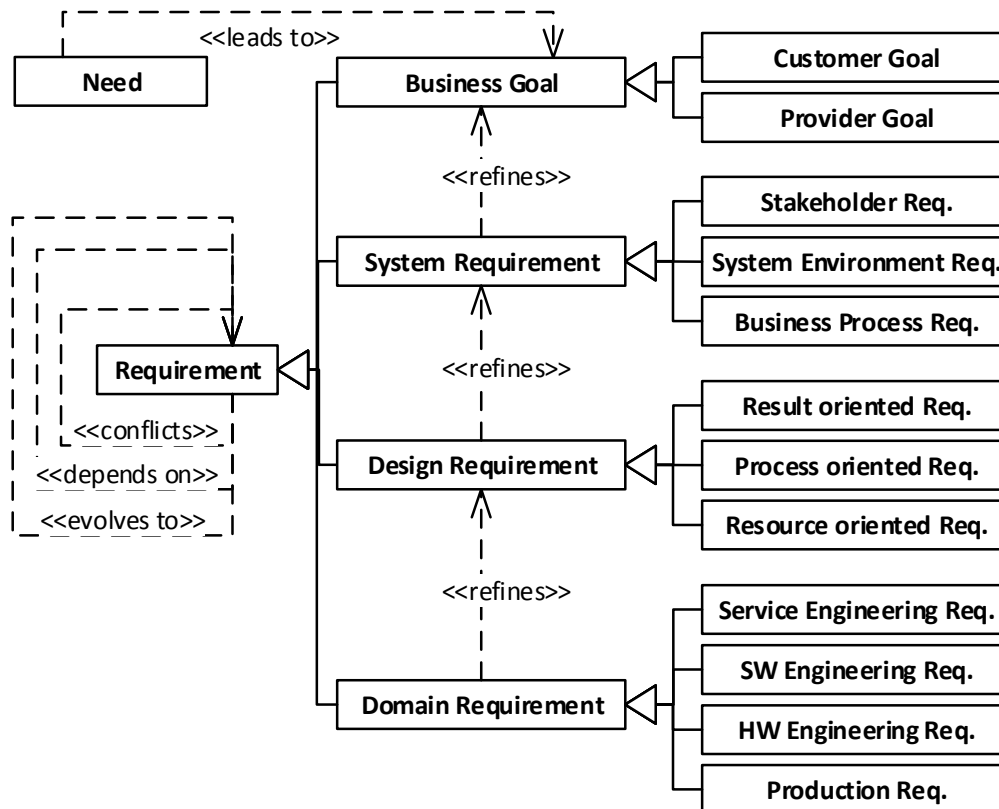


Figure 27: Requirements Submodel

The rather abstract *Business Goals* can be separated into *Customer Goals* as well as *Provider Goals*. Having defined the general business goals, the next step is to <<refine>> those goals and come up with *System Requirements*, i.e. identifying the *Stakeholder Requirements* as well as system requirements and *Business Process Requirements*. Again, *System Requirements* can be further <<refined>> and broken down according to the Function-Behavior-Structure principle. This is true for each of the disciplines involved in PSS development. On the system level *Requirements* are still discipline-neutral, i.e. it is not yet specified whether a requirement will be satisfied by hardware, software, service or a combination of those. This freedom of choice is characteristic for PSS and increases the effort of evaluating various conflicting design alternatives but it also allows greater flexibility in customizing the solution towards the actual customer needs. On the next refinement level, *Design Requirements* can be categorized in *Result oriented Requirements*, *Process oriented Requirements* and *Resource oriented Requirements*. *Resource oriented Requirements* mainly specify the structure of hardware components, the data and information necessary for the software or other potentials that are required for service delivery, such as employee skills. *Behavior oriented Requirements* refer

to the general behavior of hardware structure, the workflows in software or production systems as well as the service delivery process. In a last refinement step the *Design Requirements* can be broken down into *Domain Requirements* by translating them into the language of the developers. On this level, *Service Engineering Requirements*, *Software Engineering Requirements*, *Hardware Engineering Requirements* and *Production Requirements* are defined.

Requirements may <<conflict>> each other, meaning that the associated specifications are contradicting so that complete fulfillment of both requirements is not possible. A <<dependency>> relation denotes that the requirements have a strong interrelation so that changing one *Requirement* will likely impact the other *Requirement*. In this case the affected *Stakeholders* need to be notified. The <<evolves to>> relation refers to version management. It is used to document the evolution of *Requirements* over time, so that the different versions can be traced back to their original specification.

3.5. Specification Artifacts

Specification Artifacts serve the detailed description or conceptual modelling of *Requirements* or *Solution Artifacts*. They can occur in *Text* form, using some kind of *Illustration* or more structured forms of knowledge representation like *Diagrams*. Overall, the range of different types of *Specification Artifacts* that is used in PSS engineering is wider than in traditional engineering because of the heterogeneity of the disciplines involved. Figure 28 shows an exemplary selection of *Diagrams* that commonly serve as *Specification Artifacts*.

Use Case Diagrams are often used to describe the interaction of *Stakeholders* and the system. They primarily support the requirements engineer in identifying the overall functions and features that need to be provided. *Structure Diagrams* or *Entity Relationship Models* are used to formally specify the structural composition of the system respectively its' architecture. Furthermore, the intended system behavior or relevant processes can be specified using *Activity Diagrams*, *Business Process Models* or *Service Blueprints*. Moreover, *Value Flow Models* may be used for illustrating the *Value Flows* within the value creation network.

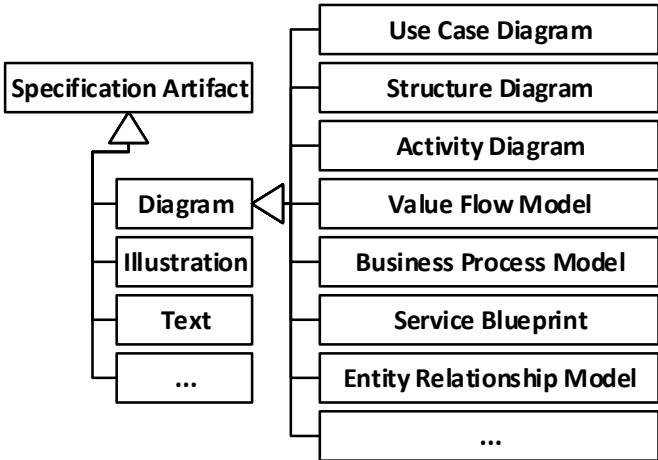


Figure 28: Specification Artifacts Submodel

3.6. Management Artifacts

In the context of requirements tracing *Management Artifacts* primarily capture changes that *<<affect>>* *Requirements*, *Solutions Artifacts* or *Production Artifacts*. A *Management Artifact* can be a *Change*, an *Issue*, a *Change Proposal*, a *Change Request*, a *Change Order* or a *Cycle*. Figure 29 gives an overview over the *Management Artifacts* and the according relations.

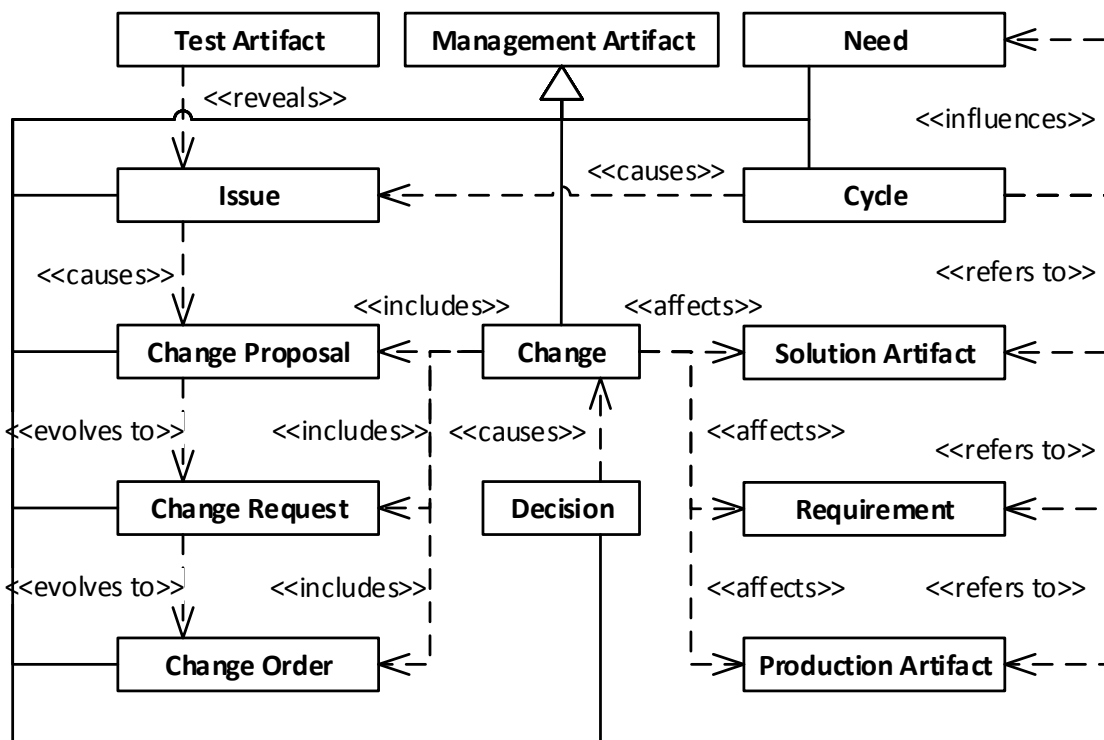


Figure 29: Management Artifacts Submodel

Changes can be related to *Cycles* (reoccurring patterns) that affect the development of a PSS, its' production or service provision. Prominent examples of *Cycles* are maturity of technology, the customer life cycle or the life cycles of components of the PSS. Especially these different component life cycles impose a challenge for PSS providers since the overall service delivery depends on the compatibility of the PSS's components. A *Cycle* can act as a trigger that *<<causes>>* an *Issue*. Additionally, *Issues* can be *<<revealed>>* through verification and validation procedures which are represented through *Test Artifacts*. As the responsibility of a PSS provider does not end with the shipment of the product to the customer the provider has to keep track of issues that evolve during the entire service delivery period which often has no temporal delimitation and requires continuous replacement of expendables, software updates or changes of the service level agreements. Following Chucholowski et al. (2014), an *Issue* (e.g. a goal derivation) then *<<causes>>* a *Change Proposal*. If the *Change Proposal* seems promising it *<<evolves to>>* a *Change Request*. At this stage further implications of the *Change* are evaluated. After a *Decision* by the *Stakeholders* that are responsible the *Change Request* further *<<evolves to>>* a *Change Order*

meaning that it is implemented. The artifacts *Change Proposal*, *Change Request*, and *Change Order* each refer to one single *Change*.

3.7. Solution Artifacts

Solution Artifacts refer to components that comprise the PSS itself, namely its functions, its behavior and its structure. Consequentially, *Solution Artifacts* can be differentiated into *Structure Elements*, *Behavior Elements* and *Function Elements*. In this conjunction, a *Structure Element* (c.f. III.H) can <<perform>> a certain *Behavior Element* (e.g. a car that drives). Again, a *Behavior Element* <<realizes>> a *Function Element* (e.g. Taking the passenger to the desired location). This *Function Element* <<creates>> a certain *Value* for the customer in order to <<satisfy>> a *Need*. The *Value* proposition is the pivotal element of each PSS business model as fulfilling customer *Needs* is the raison d'être of each PSS. Each *Solution Artifact* thus should be dedicated to fulfill the *Value* proposition. *Behavior Elements* can be used to specify the behavior of a system, respectively processes and workflows. They can be separated into *State* and *Activity*. These *Activities* may be <<performed>> by *Stakeholders* or certain components of the system. In turn, each *Activity* may <<produce>> or <<require>> certain *Structure Elements* (e.g. a data query may require a data base and produce a piece of information). In general, only *Solution Artifacts* that <<satisfy>> some *Requirement* should be part of a PSS in order to prevent over-engineering (see Figure 30).

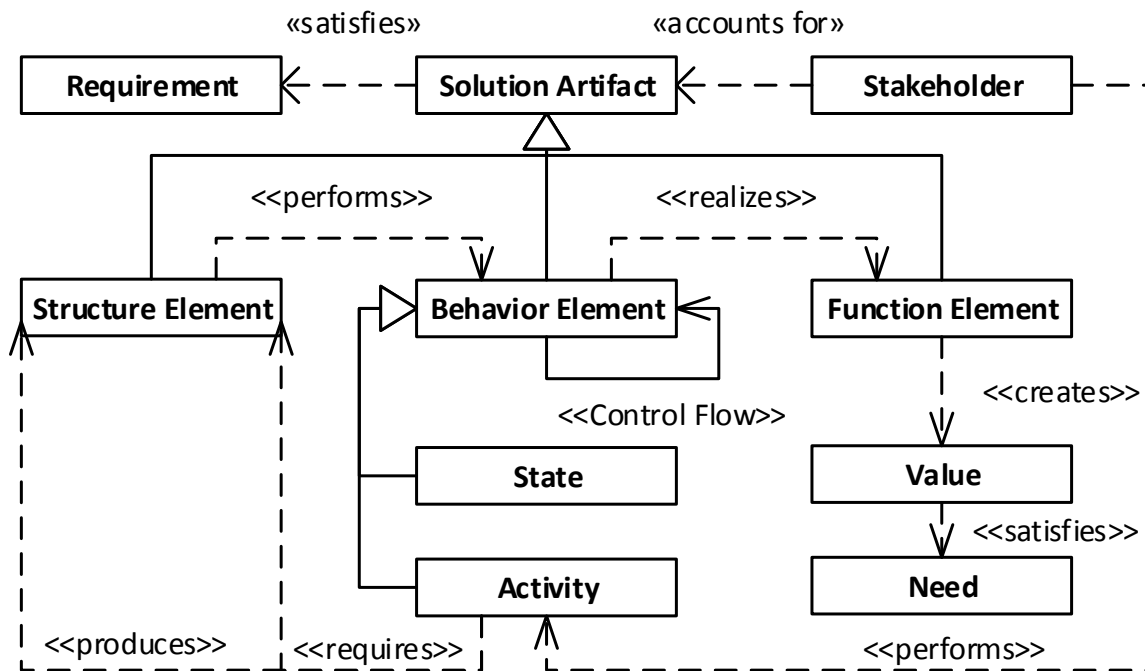


Figure 30: Solution Artifacts Submodel

3.8. Structure Elements

In general, the *Structure Elements* that constitute a PSS can be divided into *Product Elements* and *Service Resources* (See Figure 31). Overall, *Structure Elements* can be seen as the fundamental resources and potentials that enable the intended service delivery processes in order to create *Value* for the customer and satisfy a *Need*. Again, *Structure Elements* can be linked by <<Energy Flows>>, <<Material Flows>> or <<Information Flows>>. Further, structural decomposition, i.e. modularization of the PSS, is indicated by <<contains>> relationships between one *Structure Element* and another. The modules of a PSS may be homogeneous (e.g. pure hardware modules) or heterogeneous modules (e.g. mechatronic modules that contain hardware and software elements).

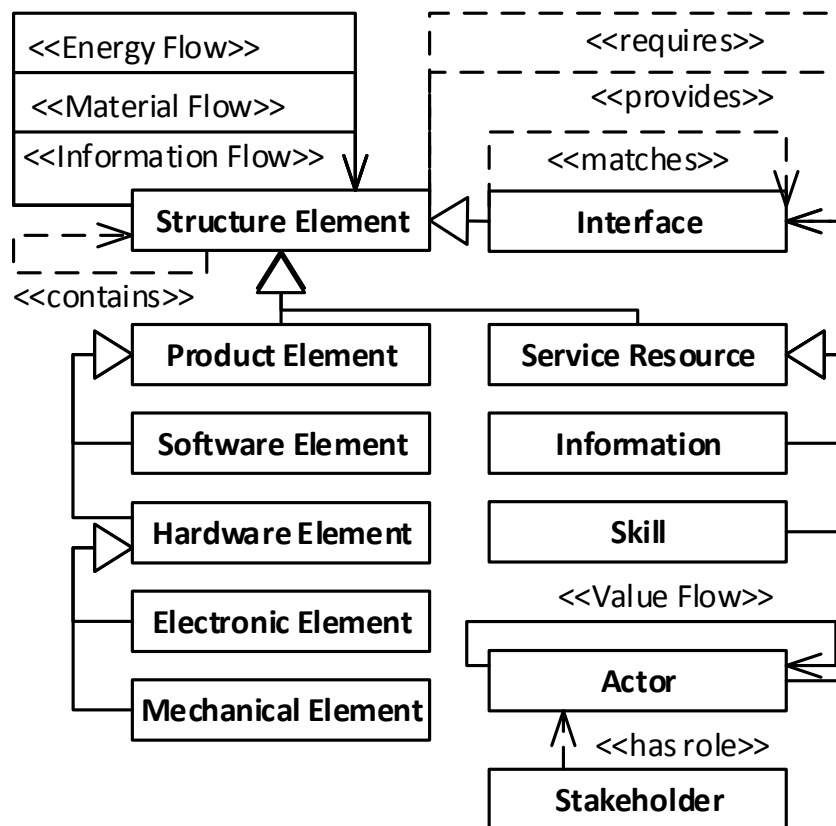


Figure 31: Structure Elements Submodel

In alignment with the disciplines involved in PSS engineering we found that *Product Elements* can be further split up into *Software Elements* and *Hardware Elements*, which are again comprised of *Electronic Elements* and *Mechanical Elements*. While in traditional product development it is sufficient to trace the fulfillment of *Requirements* to hardware and software elements that can be formally specified and evaluated, PSS providers additionally need to consider the often rather informally specified *Service Resources*. In terms of the *Service Resources* we differentiate between intangible resources like *Information* and *Skills* that are necessary prerequisites for service provision as well as tangible resources like *Actors* that perform certain *Activities* within the service process. Moreover, the aspect of *Value* co-creation which is a central concept in service engineering can be captured by <<Value

Flows>> between service provider, the customer and other *Actors* within the value creation network. <<Value Flows>> denote the assessment of the value of exchange relationships between different *Stakeholders* within the value network, such as funds, goods, information or services. The resulting network of <<Value Flows>> can be used to estimate the fairness of PSS business models and consequentially the likelihood for long-term relationships among the partners. Overall, long-term relationships are more likely to evolve if the business model is perceived as fair, meaning that each partner within the value network receives the same amount of value that it gives.

Against the background of modularization and inter-disciplinary collaboration in the development of PSS the definition of *Interfaces* plays a vital role for PSS. *Interfaces* can be seen as a definition of allowed inputs or outputs of a *Structure Element* and can thus be used for means of standardization. Consequentially, *Interfaces* may be <<provided>> or <<required>> by *Structure Elements*. If *Interfaces* <<matches>> other *Interfaces* the associated *Structure Elements* are compatible with each other.

4. Exemplary Use Cases

In order to demonstrate the practical applicability of the proposed reference model we can consider the following example of a car sharing PSS (See Figure 32 and Figure 33). A car sharing system is a typical example for a PSS that appears in different forms in various markets. In many cases the providers of car sharing systems had to go through a long period of negative return on investment as they underestimated the complexity of developing integrated solutions that sufficiently satisfy the requirements of the various stakeholders. To reduce complexity of this use case we only illustrate an extremely simplified excerpt and only use exemplarily selected constructs of our reference model.

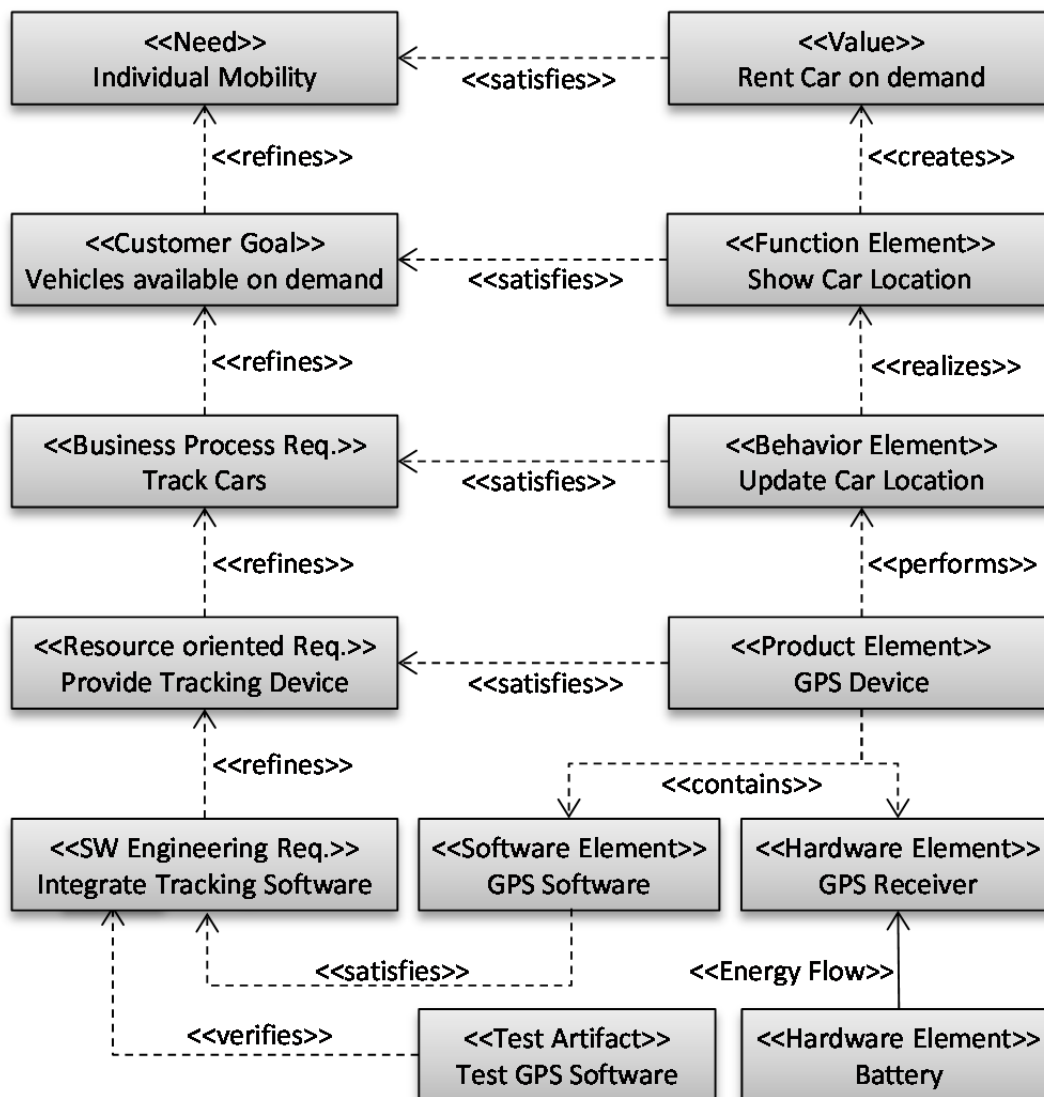


Figure 32: Exemplary Use Case: Traceability of Requirements Refinement

We consider a common need of an average citizen, namely the availability of means for individual mobility. In order to resolve this need, a potential PSS provider might want to develop a business model that offers free-floating rental cars to its customers that can be used within a certain area. One of the customer goals that can be derived from this need is that the vehicles should be available on demand without prior reservation. As the cars are floating freely within the business area of the provider a central business process requirement is that the current location of each car has to be known, meaning that the cars have to be tracked. In order to do so each car needs to be equipped with some kind of tracking device. Consequentially one of the further refining software engineering requirements is that the tracking software needs to be integrated with the tracking device which again needs to be tested. Based on these requirements the developers have decided for a GPS device which consists of hardware and software elements. Again, these solution items have relationships to other elements of the PSS, e.g. the battery which powers the GPS receiver. The GPS device as a whole continuously updates the position of the car, thus satisfying the business requirement of tracking the car. This contributes to realizing a function that allows displaying the current

location of available cars to potential customers, e.g. using a smart phone app. Altogether, the various function elements comprising the car sharing PSS create value for the customer, thus satisfying the original need.

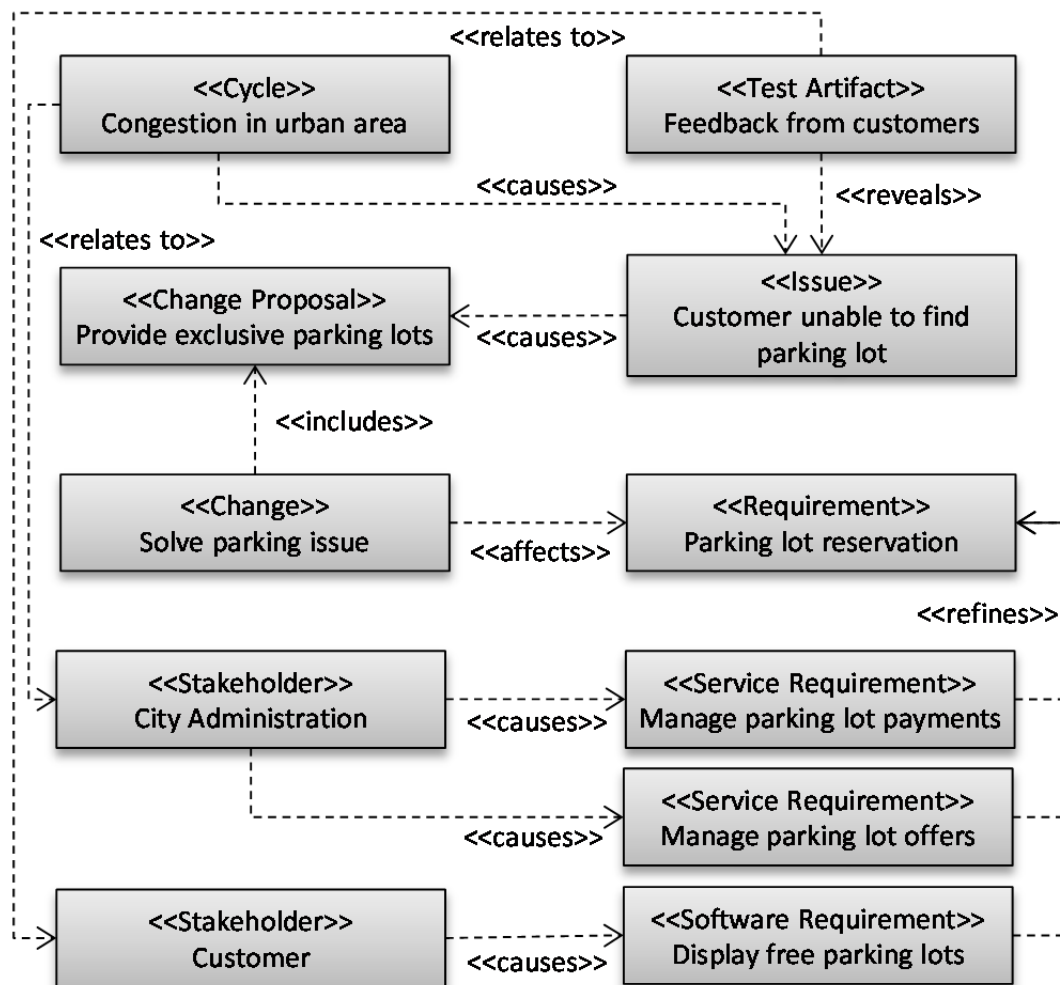


Figure 33: Exemplary Use Case: Traceability of Changes

The proposed reference model can further be used to trace changes that occur during the phase of service provision. A common issue with free-floating rental cars is that customer are unable to find adequate parking lots at their desired destination. This issue mainly appears during times of heavy congestion in urban areas, a problem that arises on a regularly basis, especially during the rush hours or on weekends. In the use case example presented in Fig. 10, this issue is revealed through customer feedback. As this issue appears repeatedly, the car sharing provider evaluates its options to improve service quality by eliminating the issue. A feasible change proposal is the provision of exclusive parking lots for its cars. Having decided on the implementation of the service change a new requirement regarding the reservation of parking lots is recorded. Further evaluation of the proposed service change shows in impact on the service as well as the software components. To be able to offer exclusive parking lots to its customers, the PSS provider needs to negotiate with a new stakeholder, the city administration. The city administration demands adequate compensation for providing reserved parking lots. Further, in order to ensure a high quality of the parking lot service, the

PSS provider needs to actively manage its offers and allocate new parking lots if necessary. However, the new parking lot service does not only impact the service components of the PSS. Also the infotainment software in the car may need to be changed so that the navigation system displays free parking lots at the destination.

5. Discussion

Our literature review shows that there has already been a significant amount of research on requirements traceability in general. The publications that were reviewed address a broad variety of aspects and propose solutions for the basic challenges that are relevant in this context. However, most publications on traceability specialize on software or systems engineering and thus address the issues from their discipline-specific perspective. In other disciplines such as mechanical engineering requirements traceability has yet not been in the focus of research. Especially in the service engineering domain, we were unable to find adequate traceability models. This way, we are among the first to explicitly model the socio-technical aspects of services resources, such as information and required skills. Some of the traceability models that we reviewed suggest that different stakeholders should be considered in order to be able to trace requirements back to their sources. However, they do not explicitly model the various types of stakeholders that need to be considered. Furthermore, integrated approaches that consider the characteristics of each discipline, relevant for PSS development have not yet been proposed. However, for seamless integration of the various discipline-specific artifacts related to PSS design a cross-disciplinary reference model for requirements traceability is necessary. This helps to reduce development costs and create additional value for the customer.

The proposed references model describes the artifacts that are relevant for achieving traceability across disciplines and defines the semantic relationships connecting those artifacts. Central issues for the practical applicability of a traceability reference model are its flexibility and adaptability towards a specific project context (Gotel et al. 2012). Our model takes this aspect into account by structuring the proposed traceability artifacts and corresponding semantic trace links into several granularity levels. Furthermore, the structure of the reference model is designed in a way that makes it extensible. The logical decomposition of artifacts and semantic relationships using hierarchical inheritance allows adding further, more detailed relationship and artifact types, if needed for special project purposes. Based on our literature review and the workshops, we found that a reference model for requirements traceability for PSS has to address four challenges: (1) pre-specification traceability, (2) inter-requirements traceability, (3) post-specification traceability and (4) traceability of changes.

Our reference model supports pre-specification traceability by identifying the relevant stakeholders in PSS development and linking them to requirements (c.f. Figure 25 and Figure 26). In doing so, we can ensure that the origin of each requirement is captured. This is important since the development of PSS is characterized by the involvement of many different stakeholders whose needs change frequently. While in many cases requirements elicitation starts with defining the desired functions and characteristics of the product, our

reference model emphasizes the value proposition of a PSS and the satisfaction of original customer needs by defining these as the true source of requirements.

Regarding inter-requirements traceability, we found that starting from abstract business goals, requirements should be refined until discipline-specific component requirements can be specified. In this process a complex network of interrelated requirements evolves iteratively. This aspect is addressed in the Requirements Submodel (c.f. Figure 27). For post-specification traceability the entire spectrum of discipline-specific artifacts along the PSS life cycle needs to be considered. Primarily, this includes tracing the satisfaction of *Requirements* through *Solution Artifacts*, namely *Function Elements*, *Behavior Elements* and *Structure Elements* (c.f. Figure 30).

Furthermore, tracing the evolution and changes of the various artifacts is essential especially regarding the continuously changing requirements that are the result of different life cycles of PSS components. Cycle orientation is a key success factor for PSS, since providers need to constantly adapt and enhance their value proposition to various cyclical changes. The history of *Changes* is captured in the Management Artifacts Submodel (c.f. Figure 29). This allows for constant evaluation of how changes in one discipline affect artifacts that are relevant for the other disciplines.

6. Conclusion and Outlook

In this work, we proposed a requirements traceability reference model for PSS. Based on a review of literature and modeling workshops with experts from various disciplines involved in PSS design and development, we identified discipline-specific artifacts and relations among them. The results of the literature review and the workshops formed the foundation for the creation of an integrated traceability reference model targeted towards cross-disciplinary development of PSS.

This paper contributes to theory by identifying characteristics of PSS and discipline-specific artifacts. By studying discipline-specific artifacts relevant for PSS development and defining the relations between those artifacts we contribute to a better understanding of requirements traceability in PSS development. From a practical perspective, the proposed model may promote cross-discipline understanding, in which the various stakeholders can access a common solution model. The reference model encompasses the relevant discipline-specific information in the form of artifacts and provides a first step to safeguard the traceability of requirements. By identifying the relationships between development and solution artifacts, the cross-checking of requirements and associated system components is possible. Using this reference model may correspondingly reduce errors, improve communication and result in a tighter integration of the disciplines involved in PSS development.

As a next step we plan to implement the proposed reference model in a software tool supporting requirements traceability in PSS engineering to evaluate its practical applicability. Possible extensions of the reference model include determining predefined attributes of the artifacts and the consideration of relations with the business models of the PSS providers.

Further research is needed regarding automated gathering of traceability data from discipline-specific engineering tools that are commonly used for creating and documenting artifacts.

Acknowledgment

We thank the German Research Foundation (DFG) for funding this work as part of the collaborative research center ‘Sonderforschungsbereich 768 – Managing cycles in innovation processes – Integrated development of product-service-systems based on technical products’ (SFB768).

Publication 8

Introducing TRAILS: A Tool supporting Traceability, Integration and Visualisation of Engineering Knowledge for Product Service Systems Development

Thomas Wolfenstetter^a, Mohammad R. Basirati^a, Markus Böhm^a, Helmut Krömer^a

^a Chair for Information Systems
Technische Universität München
Munich, Germany
{thomas.wolfenstetter, basirati, markus.boehm, krcmar}@in.tum.de

Abstract

Developing state of the art product service systems (PSS) requires the intense collaboration of different engineering domains, such as mechanical, software and service engineering. This can be a challenging task, since each engineering domain uses its own specification artefacts, software tools and data formats. However, to be able to seamlessly integrate the various components that constitute a PSS and also being able to provide comprehensive traceability throughout the entire solution life cycle it is essential to have a common representation of engineering data.

To address this issue, we present TRAILS, a novel software tool that joins the heterogeneous artefacts, such as process models, requirements specifications or diagrams of the systems structure. For this purpose, our tool uses a semantic model integration ontology onto which various source formats can be mapped. Overall, our tool provides a wide range of features that supports engineers in ensuring traceability, avoiding system inconsistencies and putting collaborative engineering into practice. Subsequently, we show the practical implementation of our approach using the case study of a bike sharing system and discuss limitations as well as possibilities for future enhancement of TRAILS.

Keywords

Model-based Systems Engineering, Traceability, Product Service Systems, Model Integration

1. Introduction

1.1. Motivation

In an increasingly digitized economy more and more companies realize that products themselves are no more the main contributors to value creation in their business. Instead, value for the customer is being created in service-oriented business models. Already today, most developed economies owe a far greater share of their national income to service provision than to manufacturing of physical products (Meier et al. 2010). Even in traditional manufacturing industries, global competition forces companies to focus on building long-term relationships with their customers by providing product-supporting services, such as maintenance, or offering the product itself as a service (Marques et al. 2013). Furthermore, environmental considerations cause enterprises to move from a product-based economy to a service-based economy which limits their susceptibility to environment issues (Maussang et al. 2009). As a consequence, the concept of product service systems (PSS), i.e. integrated systems that combine product and service components, is increasingly gaining popularity as a strategic measure to deal with these issues.

PSS development thus involves various stakeholders from different engineering domains who need to develop hardware, software and service components based on descriptions of the customers' needs and seamlessly integrate them into a comprehensive solution while at the same time reacting flexibly to changing requirements and a dynamic system environment. For example, changing legislation regarding privacy protection might impact the way customer related data is handled by the PSS provider in order to ensure compliance. This not only impacts the service processes in which this data is being collected, but also software systems that store and process the data and even might force the PSS provider to change hardware components that rely on customer data in order to provide their functions. In this example a simple requirements change entails an adaptation or possibly redevelopment of various components of the PSS, requiring engineers from different disciplines to communicate with each other, coordinate the changes made to the system as a whole and anticipate how changing one component influences other parts of the PSS. As a result, not only the degree of involvement of stakeholders from different domains increases. There is also need for tight collaboration and communication among all stakeholders involved. Therefore, a major challenge for PSS engineering is to provide integrated conceptual models and comprehensive representation techniques to support cross-domain collaboration (Vasantha et al. 2012).

Moreover, the cross-domain engineering process is not the only aspect that differentiates the development of PSS from traditional product development. By its very central idea the concept of PSS focuses on integration of business models, products and services along the entire life cycle in order to create additional value for the customer (Vasantha et al. 2012). Like in every long-term relationship, expectations and capabilities, both on the provider and on the customer side evolve over time. Consequentially, PSS providers need to deal with changing requirements to be satisfied. Therefore, they need to monitor the traceability relationships between requirements and affected parts of the PSS solution including both, tangible product components as well as intangible services (Maussang et al. 2009).

The complexity of PSS engineering also manifests itself in the heterogeneity of artefacts, which are created and used along the PSS life cycle. For instance, in the process of developing a PSS every engineering domain involved follows their own domain-specific approaches when creating the various types of development artefacts that are required along the process, such as process models, requirements specifications, design structure matrices, use case diagrams or component diagrams. As artefact we hereby understand every tangible information object that is created along the life cycle of a system to describe its e.g. design, architecture, functions as well as the processes and the organization associated with it. All of these artefacts are highly interdependent as they ultimately specify components of the PSS, which finally need to function together reliably.

Managing the relationships between PSS engineering artefacts is necessary for developers to anticipate the change impact of an artefact on others and to prevent inconsistencies as well as to trace the evolution of the individual artefacts and the PSS as a whole. For this purpose, the structural architecture of a PSS together with the dynamics of the service processes and the evolution of requirements that are linked to them needs to be captured. Moreover, all of this engineering knowledge needs to be presented in a way that allows engineers to get a comprehensive overview of the problem as well as the solution domain (Meier et al. 2010). By doing so, it is possible to dynamically adapt the solution to changing needs of customers and the evolving environment in which the PSS competes.

However, current industry practice shows that PSS development relies on a multitude of different modelling languages and tools that are largely incompatible with each other. Thus, today the analysis of dependencies within a PSS requires tremendous manual efforts and the integrability of components as well as mutual impact can only be checked relatively late in the development process.

In a nutshell, PSS development is a highly complex process with high number of dependencies between heterogeneous artefacts. However, in practice traditional engineering methods often struggle when coping with the challenges of PSS development, thus producing callow solution designs that cannot live up to their full potential. Although there exists a wide range of approaches for modelling the different components of a PSS (Meier et al. 2010; Vasantha et al. 2012), the design of integrated products and services along with the issue of traceability has not been supported sufficiently by software tools (Baines et al. 2007; Meier et al. 2010; Cavalieri and Pezzotta 2012).

Also, tools and modelling languages which are used in PSS development do not support integrated analysis of PSS artefacts and their relationships which can lead to inconsistencies or unanticipated changes even in late phases of development. Therefore, there is lack of a tool to support modelling and analysing PSS artefacts and their relationships from a holistic viewpoint.

1.2. Approach

We tackle this issue by proposing a tool that supports PSS development by providing means to integrate the various domain-specific artefacts into a comprehensive "semantic engineering" graph. This graph represents the various PSS artefacts, such as requirements, components, processes, activities, stakeholders, use cases or tests as nodes and the relationships and flows between those artefacts as edges. The tool facilitates capturing the relations between different artefacts through the whole PSS life cycle. It further visualises the semantic engineering graph or particular views (subset of nodes or edges) to the user and provides features to analyse and edit the graph.

To achieve the aforementioned goal, our research intends to establish a theoretical foundation and then present the corresponding software tool that enables cross-domain traceability and model integration among PSS elements. Based on this agenda, we name our software tool TRAILS, **TR**Aceability, model **I**ntegration and **L**ife-cycle management **S**upport.

Our proposed approach is not focused on capturing and describing every little detail of the system components that can be modelled in the respective domain-specific modelling languages (e.g. single function calls in software code, detailed geometry of hardware parts or activities of a service process that are modelled exact to the second), but it is more concentrated on a project management level, allowing requirements engineers or project managers to analyse the overall relationships between system components, requirements, stakeholders or other artefacts that are relevant in the development process. At this point, we also want to emphasize, that our approach and tool are primarily designed to support the model-based engineering (MBE) of PSS but not model-driven engineering (MDE), i.e. automatic generation of software code or service guidelines from models. In case of PSS we think that at the moment MBE is more feasible than MDE since PSS are complex socio-technical systems. Therefore models can play an important but not a dominant role in the design and development of PSS. Since PSS development requires an integrated view on the system under development, the tool features multiple integration approaches demonstrating the result as a semantic engineering graph (network). This approach is enhanced with allowing multiple views on the resulted graph for each specific purpose.

To this end, first, we define a reference ontology that specifies the conceptual entities that are used in the multiple engineering domains involved in the development of a PSS. The development of this integration ontology, is based on literature reviews within the engineering domains involved as well as expert interviews, PSS case studies and modelling workshops. In TRAILS this integration ontology is used as a framework that defines element types and their associations that are used during PSS design and development. In TRAILS model integration is being performed by transforming each type of model that is supported by the tool into the format defined by the integration ontology and then linking similar or related artefacts.

1.3. Structure of Article

The structure of this paper is as follows. In section 2, first we give an overview of available modelling methods used in PSS engineering, then existing software tools for PSS are analysed. Afterwards, in section 3, the model transformation methods are discussed and we explain the model transformation process used in TRAILS. Following, in section 4, the basic concepts of TRAILS are introduced which is followed by describing available features of our tool in section 5. In section 6, a case study demonstrates the use of TRAILS' different features in a use case. In section 7 we discuss limitations and possible future improvements of our approach. Finally, section 8 gives a summary of the research presented in this paper.

2. Related Work

Analysing the literature related to PSS design and development we find that research is more focused on methodologies and modelling techniques rather than providing tools to support the presented methods. Except two works on computer aided design tools targeted at PSS development and modelling which we discuss subsequently, we did not find any additional tools that are explicitly design for supporting the development of PSS. In this section we first summarise proposed PSS modelling methods in literature and then briefly explain both tools we found.

2.1. PSS Modelling Methods

A considerable number of modelling methods for PSS development have been proposed in literature, most of which aim at systematising the functions or value proposition of a PSS from the customer's perspective and focusing on the service aspect of the PSS (Qu et al. 2016). Here, one group of methods focuses particularly on the hierarchical configuration of a PSS from services or other components (Klingner and Becker 2012). In contrast, some works aimed at covering PSS innovation phases comprehensively, therefore they offered several modelling techniques, each focusing at a particular situation in a PSS development.

Most of the approaches for modelling a PSS presented in literature are based on service blueprinting proposed by Shostack (1982) more than 30 years ago. Geum and Park (2011) for example extend the service blueprint with new notations to capture the flow of product usage and service usage from the provider to the customer and the relationship between products and services. Lee and Kim (2012) focus on functional modelling of a PSS. They modify the service blueprint by adding a function layer to show interactions between service provider and service consumer more explicitly. Geng and Chu (2011) use a conceptual service blueprint adding a user task model (to improve process-oriented design of a PSS with requirement analysis from user perspective) and a function model (to show the relation between requirements and PSS concepts). Service blueprinting and its extensions are mainly focused on visualisation of service processes in the context of a PSS. This technique elaborates the provided service activities at every stage of the PSS life cycle and specifies the level of customer involvement or visibility of certain activities to the customer. Although adopting service blueprint gives a thorough overview of the service activities they contain, it lacks the

details for designing a PSS specially the relations between needs, services and (physical) PSS components.

Lim et al. (2012) analyse the modelling techniques used for visualisation of PSS. They divide methods based on the aspect of PSS which is modelled. According to their study, most research focuses on the service processes of a PSS which among them, service blueprint got more attention for visualisation. Another studies aim at modelling the stakeholders of a PSS focusing on the relations between them, proposing alternative presentations of the service processes using a matrix called PSS board which shows how the PSS provider and the partners works to fulfil the customers' need in different stages of the service process. Maussang et al. (2009) argue that there is a gap between product development and the need for technical specifications of physical objects and the system approach. In order to close this gap, they suggest to use the graph of inter-actors and functional block diagrams for designing a PSS. They argue that functional block diagram is a useful tool for PSS modelling and analysis as functional representation of a PSS during its conceptual design phase is necessary.

According to Van Halen et al. (2005), appropriate tools are required to deal with high complexity of a typical PSS. They propose a methodology that offers a wide range of modelling methods for strategic analysis in different phases of PSS development. With several papers published in this area, we find that model integration is a common concept in the systems engineering process. However, most work in this area is on a rather high level, analysing the suitability of certain integration strategies, i.e. vertical vs. horizontal (Frank et al. 2014). Although there are approaches that utilize ontologies for modelling the dependencies between the various components of a PSS (Hajimohammadi et al. 2017), they remain at a high level of abstraction, thus suggesting that existing products and services are bundled together in order to form a PSS. In contrast to this, our approach is able to capture PSS where product and service components are engineered from scratch or at least modified in order to be integrated seamlessly.

2.2. PSS Computer Aided Modelling Tools

Several studies which reviewed the state of art of PSS engineering, discuss that there is need for software tools to support the modelling of PSS (Baines et al. 2007; Meier et al. 2010). Beuren et al. (2013) emphasize the need for tools that provide visualisation and modelling of different components of PSS including tangible and intangible elements to improve the understanding of a PSS engineering project. Morelli (2006) highlights the importance of a graphic representation technique for modelling PSS requirements. He also claims that while there are plenty of graphical notations in information sciences, they cannot be used for representing all elements involved in a PSS, like space, time and physical outlines.

Following this view, Sakao et al. (2009b), discuss that a variety of the tools available for product development concentrate on physical and domain-specific details, but there is no particular tool that aims at designing integrated systems of services and products simultaneously. In order to close this gap, they propose a tool, called Service Explorer, which supports designing services according to value created by products' functions and user

requirements. Service Explorer provides several modelling techniques, a database for managing services and some reasoning engines to help developers.

Komoto and Tomiyama (2008) argue that Service Explorer (Sakao et al. 2009b) cannot explicitly elaborate the relations between services and products which is required for designing a PSS. They present a tool which combines service modelling with a life cycle simulator. The tool allows to analyse alternative PSS designs by quantitatively calculating economic and environmental performances from a holistic viewpoint. A relatively similar approach has been presented by Nemoto et al. (2015). They present a framework and software tool that allows to formalize design knowledge from previous engineering projects and existing PSS and use those insights for configuring a new PSS offering, but on a rather high level of abstraction.

2.3. Implications for Comprehensive PSS Engineering Tool Support

The works we discussed are mostly from the service design view and lack consideration of physical elements in modelling. However, we argue that there is no single modelling technique for development of a PSS that tackles the issues sufficiently. Since PSS are rather complex socio-technical systems, many different perspectives are required to be investigated for a comprehensive design.

The need for a tool that supports engineers in analysing the dependencies among artefacts has been recognized for a plethora of different use cases, e.g. for building the links between requirements engineering and safety analysis (Vilela et al. 2017) or integrating product life cycle management with service life cycle management of a PSS offering (Wiesner et al. 2015). Also Tang et al. (2006) argue that for complex software systems there needs to be traceability from rationale to design. The same is even more important for PSS where an even bigger picture needs to be taken into account. It is however surprising that the need for an automated traceability tool has been recognized more than 3 decades ago (Dorfman and Flynn 1984) and that still today satisfying solutions to this problem are scarce. In fact, a recent literature review on requirements engineering for PSS listed conflict detection and resolution among requirements, dynamic requirement change forecasting smart requirement management and proactive response as the main challenges to be tackled in the future (Song 2017). Therefore, we aim at enabling an inclusive view by supporting visualisation of traceability links between different components of a PSS in a model integration ontology.

3. Model Integration

As discussed in section 1, PSS development involves various stakeholders which rely on their domain-specific models, diagrams or other development artefacts. Therefore, every artefact is created using specialized software tools, is specified in a domain-specific modelling language and is serialized as one of many different persistent data formats. Besides, domain-specific models represent knowledge only from a particular perspective and just a fraction of the

system is captured. Consequently, a thorough understanding of the comprehensive system design and the cross-domain dependencies is missing.

One way to address this problem is to use a single modelling language across all involved engineering domains. However, this approach comes with a huge disadvantage. The more concepts and logics a modelling language is capable of expressing, the more complex and complicated to comprehend it gets. Thus, establishing a single comprehensive modelling method covering all different aspects of a system leads to a complex and confusing representation (Eisenbart et al. 2012).

On the other hand, analysis of a PSS using models from different perspectives lacks consideration of relations between these models since many elements are interdependent. According to Chen et al. (2008), the lack of interoperability across the different departments of an enterprise is caused by interoperability barriers on four different levels (data, service, process and business). In our approach we focus on overcoming the issue of interoperability of data, since this is the fundamental layer for the integration of PSS components across conceptual, technological and organisational barriers. However, we encourage others to advance our work in order to support interoperability on a higher level.

We address the interoperability issue on the data level by proposing an integration ontology which specifies the generic artefact types (entities) that are used by the various DSML as well as the types of semantic relationships that can exist between those artefacts. This framework enables stakeholders with separate perspectives to analyse the relationships between their models and the others'.

To integrate models in one cross-domain representation, transformations from different domain-specific models are required. Thus, first we discuss briefly what types of model transformations exist; later the approach of this work for integration of different models is presented.

3.1. Model Transformation

Since in practice, most of the models are graph-based or can be transformed into a graph (also natural language expressions can be viewed as a graph), we analyse mainly the graph-based transformations. In general, several types of graph-based model-to-model transformation methods exist. We categorize these methods based on two general criteria and explain each category separately.

Graph-based transformation methods can be direct or indirect based on whether models are transformed directly to each other or an intermediate model is applied. If the transformation mechanism requires both source and target models to be in the same technical space, we call it syntax-dependent transformation. On the contrary, syntax-independent transformations are not based on a particular language.

With regards to the introduced criteria, we classify all graph-based transformations into four categories (c.f. Table 12).

Table 12: General Types of Model Transformations

Direct Syntax-dependent	Direct Syntax-independent
Fixed Intermediate Language	Flexible Intermediate Language

Direct transformation methods define a set of rules which transfer source model to target model without use of an intermediate model. Direct syntax-dependent transformations requires both source and target models to use the same technical space (e.g. XML). In contrast, direct syntax-independent methods are not coupled to a particular technical space. This type of transformations first parses the source model into the syntax of target model's technical space, then map the parsed model to the structure conforming to the target model's meta-model (Mens and van Gorp 2006). While direct syntax-dependent methods can enclose the maximal possible transmittable detail from the source model to the target model (Varró and Pataricza 2004), such methods are very limited regarding the languages which can be supported due to their binding to a concrete syntax. Syntax-independent direct transformations resolve this issue by decoupling the transformation description from the technical space of the involved languages. For every pair of source and target models, direct transformations requires an explicit implementation. For example, Medvidovic et al. (2003) present a model integration approach that uses direct mappings from one modelling language to another. However, this approach is primarily focused on syntactical issues than on semantics (meaning) of the relationships between artefact and it requires individual mappers for each pair of modelling languages that is to be covered. This problem has been addressed by indirect transformations.

Indirect transformations rely on a well-defined intermediate language and each transformation process involves transforming from the source model to the intermediate language and afterwards, transforming from intermediate language to the target model (Czarnecki and Helsen 2006). We refer to a syntax-dependent indirect transformation as fixed intermediate language transformation and similarly syntax-independent indirect transformation as flexible intermediate language transformation. The intermediate language of a fixed method is defined in the same technical space of the source and the target models. Flexible indirect methods are not bound to the underlying syntax of a technical space, but rather the intermediate language is only expressed in its concepts. Thus, with higher level of abstraction, the intermediate language can support more models and be more flexible regarding in future improvements and changes (Bézivin et al. 2006). The great advantage here is that one does not require a separate set of transformation rules between every two domain-specific modelling language but only between each domain-specific modelling language and the intermediate language.

4. TRAILS Integration Method

In its core, TRAILS is founded on two basic concepts, model integration and model transformation with the latter being a prerequisite for the first. In a nutshell, the basic ideas behind TRAILS is to interrelate all available engineering information within a semantic graph by transforming various kinds of domain-specific models and other engineering artefacts into a format that conforms to a cross-domain model integration ontology. We explain these basic concepts in the following in order to lay a solid ground for further explaining the core features of TRAILS.

4.1. Model Integration Ontology

As presented in detail in some of our earlier publications (Kernschmidt et al. 2013; Wolfenstetter et al. 2014; Wolfenstetter et al. 2015b), we developed an ontology of PSS engineering artefacts whose evolution needs to be captured in order to ensure traceability. However, this ontology is not solely focused on ensuring traceability but also on the more generic issue of integrating engineering information which is an essential prerequisite for ensuring traceability. In this sense, it defines the fundamental ontological concepts in the context of PSS development and during service provision, such as requirements, actors, business processes or decisions made in the engineering process. Additionally it specifies which types of semantic relationships can exist between those ontological concepts. For example, a solution component *satisfies* a requirement or an actor *performs* an activity. For the purpose of merging the engineering knowledge distributed over several domain-specific models, every artefact that is imported into TRAILS is translated to comply with this model integration ontology. This means that the domain-specific ontologies that are defined by the meta-models of the domain-specific modelling languages are mapped onto a common language using the proposed ontology as a meta-model. In the following we explain the structure of the model integration ontology in detail.

The model integration ontology uses on the most abstract level three basic *Elements* to describe the artefacts of PSS engineering and their dependencies. These *Elements* are *Nodes*, *Edges* and *Attributes* with each of them being hierarchically further decomposed into more concrete types of ontological concepts.

The general purpose of *Attributes* is to capture descriptive information such as duration, weight, price or colour of ontological entities as well as meta-information (e.g. name, id, date of creation etc.); basically everything that cannot be considered as an entity itself. Naturally, attributes can have attributes themselves, for example names or units. Furthermore, it is possible to define *Relationships* between *Attributes* to e.g. define the mechanics of unit conversions.

Regarding *Edges* the TRAILS model integration ontology differentiates between *Flows* and *Relationships* (c.f. Figure 34). On the one hand, *Flows* characterise the transferral or transmission of value, material, energy or information between two entities or describe the order of activities in a process, i.e. *Control Flow*. On the other hand, there is the concept of

Relationships which are universally valid while *Flows* have a cause and are bound to a defined period of time in which they occur. In this context, the TRAILS integration ontology distinguishes *Causal Relationships* (e.g. create), *Chronologic Relationships* (e.g. evolves to), *Referential Relationships* (e.g. refers to), *Inclusion Relationships* (e.g. part of) and *Inheritance Relationships*.

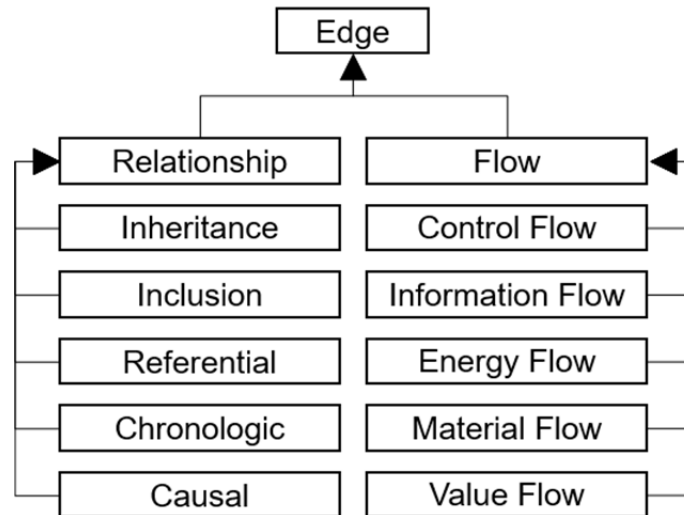


Figure 34: TRAILS Model Integration Ontology: Edge Types

As with *Nodes*, we further differentiate between two generic types (c.f. Figure 35). *Solution Artefacts* represent the solution to the original customer problem, i.e. the features, behaviour and (component) structure of the PSS itself. *Development Artefacts* specify the problem domain and the process of working on a solution to these problems, i.e. the development process. Additionally, since PSS are socio-technical systems, humans that interact with the PSS or are by other means related to the development of the PSS or to service provision are summarised as stakeholders and on more concrete levels of the ontology decomposed into the various sub-types.

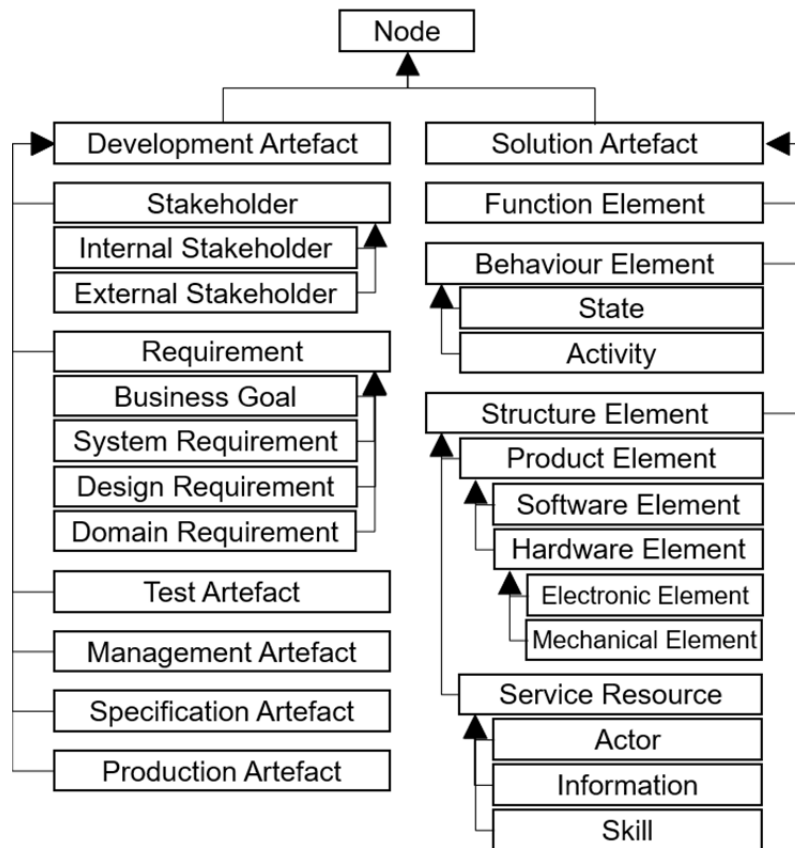


Figure 35: TRAILS Model Integration Ontology: Node Types

As discussed, *Development Artefacts* are supposed to contain information related to the development and they are not part of the resulting PSS. On a more detailed level, we distinguish between five different sub-types of *Development Artefacts* in PSS engineering. First, *Requirement Artefacts* are used to structure and define the problem for which the final PSS represents a solution.

In the context of PSS engineering, requirements can be broken down into four levels of abstraction. On the highest level, business goals of the PSS are being defined. Based on these business goals, system level requirements are derived which represent for example the needs of stakeholders, environmental considerations and business process demands. In the next step, design level requirements are elicited to address the details of system level requirements. Again, at the most detailed abstraction level the design level requirements are translated into domain-specific requirements which all different involved domains (e.g. software engineering, mechanical engineering or service engineering) can work with.

Specification Artefacts are means to describe requirements or system designs by using different techniques. *Specification Artefacts* can take various forms. Most commonly they appear as natural language texts, graph-based models or other sorts of diagrams. However, also sketch drawings, other kinds of illustrations and even videos could serve as *Specification Artefacts*.

Test Artefacts are any kind of artefact that serve in the process of checking whether the solution satisfies the requirements. Hence, the variety of potential *Test Artefacts* ranges from mathematical or logical proofs to computational simulations, experiments to informal methods, such as stakeholders' feedbacks.

Production Artefacts capture and represent knowledge that is relevant for the manufacturing process of hardware components of the PSS. This includes for example, the machinery within the assembly line, equipment used during the process or the specification of the manufacturing process itself. This way it is possible to link physical PSS components to the manufacturing process and trace whether changes to the design of hardware components impact the set-up for component manufacturing.

Management Artefacts keep track of issues initiated in development of a PSS. For example a change request or a decision information are considered as *Management Artefacts*. They are predominantly of use when tracing the evolution of other types of *Development Artefacts* over time.

Stakeholders represent different types of roles who are involved in the PSS life cycle. Handling complex network of stakeholders in a PSS is challenging due to several reasons. Since a PSS requires a new business model, the entire organization of PSS provider is affected and consequently new requirements for each department should be addressed. Besides, by adding services to a core product, new range of stakeholders will be included in development of a PSS. In the ontology we distinguish between two general types of stakeholders: *Internal Stakeholders* and *External Stakeholders*. Internal stakeholders refer to all units and collaborators to providing the PSS. In contrast, external stakeholders are not part of the PSS providing organization.

Solution Artefacts refer to components which construct the PSS including products and services. In the meta-model we classified *Solution Artefacts* into *Function Elements*, *Behaviour Elements* and *Structure Elements*. *Solution Artefacts* satisfy *Requirement Artefacts* and they are verified by *Test Artefacts*.

Structure Elements are fundamental resources constituting a PSS which are categorized to *Product Elements* and *Service Elements*. Both tangible resources, like material, and intangible resources, like information, contribute to structure elements. The system performs some workflows and processes in order to accomplish a target function. These workflows and processes are presented in the *Behaviour Element* and the functions of the system are presented in *Function Element*.

4.2. Model Transformation Process

To consolidate the engineering information that is contained in the various domain-specific models, TRAILS transforms these models into a semantic graph that conforms to the model integration ontology presented in the preceding section.

In order to support model transformations between different technical spaces and serialization formats, the model transformation process itself is divided into two independent steps. First, so-called **I/O mappers** are used to de-serialize various data formats, such as ReqIF, XMI or even CSV, and representing them as a generic graph structure. In this context, generic graph representation means that the resulting data structure consists of only untyped nodes, edges and attributes. At this point, the concrete syntax that is used by domain-specific modelling and engineering tools has been transformed into an abstract syntax that still conforms to the the meta-model of the source modelling language. As a second step, **model mappers** transform the generic graph into a semantic graph that conforms to the specifications defined by the model integration ontology. To ensure a high level of transformation accuracy, TRAILS uses customized model mappers for each domain-specific modelling language. However the general mode of operation is similar for each model mapper.

Each **model mapper** contains a set of rules that specify an equivalent for every node, edge or attribute of the respective meta-model within the model integration ontology. In the most simple case, each element can be mapped on one equivalent element of the same type (e.g. a node type being mapped to another node type). However, in other cases mapping patterns are more complex. For instance, a set consisting of two nodes that are linked by a certain type of edge could be mapped to one single node with a certain attribute. Overall, each model mapper uses a sequence of atomic transformation operators illustrated in



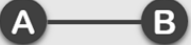











Operation	Source Model	Target Model
map		
insert		
skip		
create		
hide		
split		
merge		

Figure 36: Generic Transformation Operators

Probably the most intuitive transformation operator is to **map** one type of element from the source model to an analogical element of the target model, i.e. edge to edge, node to node and attribute to attribute. In this case, the mapping operator works bi-directionally and each

element can be considered as the equivalent of the other. Simply speaking, this operation does merely just change the name of corresponding type to the one defined in the target model. Of course, performed on nodes and edges, this operation entails analogous operations on their attributes as well.

The **insert** transformation operator is mainly performed on two nodes that are connected by an edge. For a certain pattern (node A connected by edge 1 to node B) this operator splits the edge into two and inserts a predefined node X at the junction. Amongst others, this operator is used for in cases where the meta-model of a domain-specific modelling language would not allow a direct edge between two nodes. For example, the Event-driven Process Chain (EPC) meta-model does not allow a control flow from one activity to another, but only between activities and event. So when transforming a UML activity diagram via the model integration ontology to an EPC model, additional event nodes need to be inserted. As depicted in Figure 36, there is also an analogous **skip** operator that performs the inverse of the **insert** operator. This means, when detecting a node X that is connected to A and B via an edge of type 1 it removes X and links A and B directly.

When transforming models, it is often the case that for certain elements in the source model their equivalent in the target model misses important attributes. In this context, it is often necessary to **create** an additional node to capture the information that would otherwise get lost. Also, some modelling languages allot that certain nodes only appear in connection with other nodes. For example, in UML use case diagrams, use case always need to be directly or indirectly linked to actors. Vice versa, the **hide** operator ignores for example nodes in the source model that are not intended in the target model. Wherever possible, it preserves the information contained in the discarded node by adding it to the one that is maintained.

Another inverse pair of atomic transformation operators are used to **split** or **merge** nodes. When specified for a certain type of node within the source model the **split** operator creates two separate nodes of another type than the original node connected by a pre-defined edge. This kind of operator is required usually if an element in the source model has no direct equivalent in the target model. In this case, after the transformation the information that was before carried by one node is split onto two separate nodes. An example for this approach can be found when transforming a BPMN diagram into a format compliant to the model integration ontology. While gateways in BPMN contain the decision that is being made as well as the logical consequence (i.e. which control flow is being followed after the decision) the concept specified differently in the integration ontology. Here, the activity and the logical conjunction (AND, OR, XOR) are treated as two separate nodes. Therefore, when importing a BPMN diagram TRAILS will apply the **split** operator and vice versa, the **merge** operator when exporting to BPMN.

By separating the model transformation process into I/O mapping and model mapping each I/O mapper can be used for transforming various modelling languages or meta-models respectively. Correspondingly, each model mapper is able to transform a specific type of meta-model into a graph representation that conforms to the model integration ontology for PSS that is used in TRAILS. Apart from importing various domain-specific models, TRAILS

is also capable of exporting the integrated PSS engineering information into domain-specific model languages and data formats. For this purpose, the model transformation process, as described before is reversed.

5. TRAILS Features

TRAILS pivotal mission is to support various stakeholders along the PSS life cycle in integrating, analysing and enhancing the knowledge that is often spread across and hidden in the multiple different engineering artefacts. The tool therefore provides a number of features related to import, merging, editing and analysing graph-based models from various engineering domains.

5.1. Importing Models

The core ability of TRAILS is that different types of model specification formats can be imported and transformed into the cross-disciplinary representation defined by the TRAILS Model Integration Ontology. Furthermore, the entire semantic graph that results from integrating the various types of PSS engineering artefacts can later be entirely or in parts transformed into other formats supported by the tool.

One type of specification formats supported by TRAILS is for example the rather text-oriented **Requirements Interchange Format** (ReqIF). ReqIF is a format based on XML which enables stakeholders with different modelling and requirement authoring tools to collaborate by exchanging their requirements' information.

Since PSS intend to be solutions to specific needs it is crucial for the PSS provider to fulfil the customer's requirements as complete as possible. Requirements engineering is thus one of the most important activities both during PSS development as well as service provision. Due to the dynamic business environment in which PSS compete, requirements eventually change over time and the PSS needs to be adapted accordingly. By linking the information that is contained in requirements documents to system design models of the product components or process models related to service delivery it is possible to anticipate the impact of changing requirements on the PSS design more accurately and anticipate the consequences. The information that is needed in this context can be imported from distinct requirements engineering tools using the ReqIF format.

Two other important modelling languages that are relevant in PSS engineering are the **Unified Modelling Language** (UML) and its almost twin brother, the **Systems Modelling Language** (SysML). UML, the older of both brothers, originated from software engineering and was developed to provide a common platform for system architects and software developers to communicate over system analysis and implementation.

When recognizing the advantages of a notation that allows logically decomposing complex systems, UML subsequently entered the mechanical engineering domain and was adapted to fit its characteristics. The resulting modelling language SysML extends a subset of the basic UML diagram types but also introduces new concepts, such as ports. So, while UML is a

more software-oriented modelling language, SysML aims at modelling and designing complex systems that rather stem from the mechanical engineering domain. However, both of them offer various diagrams for specifying a systems structure as well as its dynamics.

While UML and SysML can be used to specify the rather technical aspects of a PSS, namely the hardware and software components, the service engineering domain mostly relies on notations to specify business processes. Widely used modelling techniques for business processes are the **Business Process Modelling Notation** (BPMN) as well as **Event-driven Process Chains** (EPC). An EPC, for example, is an ordered graph of events and functions that enables describing alternative and parallel execution of processes and it is enhanced with logical operators like AND, OR, etc. The structure and notation of both, BPMN and EPCs, is very similar and they are often supported by the same software tools, e.g. MS Visio. We thus chose this software tool as an example to show the import of such process models.

Although they are not commonly referred to as modelling languages, TRAILS supports the import of (and export to) other important formats. For example, the **Resource Description Framework** (RDF) is a general technique for conceptual description of resources. It is widely used in the context of semantic web applications for specifying entities and their semantic relationships to each other forming an ontological graph that explains real world concepts. TRAILS uses the RDF format to define the structure of the model integration ontology it uses internally as a model representation format. Furthermore, it is used to define the model mapping rules that are applied when importing and exporting models in another language. RDF models can be serialized in various formats, the most important being probably the **RDF-XML** format and the **Terse RDF Triple Language** (TTL) which is a serialization format that is easier for humans to read than the widely used RDF-XML format. Since RDF is a technology that is used across various domains in order to structure and represent knowledge in a graph-based form, we chose to support both formats in TRAILS.

5.2. Merging Models

Since PSS aim to be customer-centric solutions in which the individual components need to be integrated seamlessly to provide the desired service and guarantee an enhanced customer experience, developers need to be able to evaluate how the design and the behaviour of the individual components impacts each other. In order to do so, it is helpful to identify and explicitly model the dependencies and overlaps among the various domain specific development artefacts involved in the PSS development.

For this purpose, TRAILS allows to merge the various models of a PSS, each describing a specific viewpoint on the system as a whole, by identifying common concepts or entities in the different models. After importing the different domain specific models into TRAILS they are merged into a semantic graph with the nodes of this graph representing entities or real world concepts, respectively.

Each time a new model is imported, TRAILS can perform model merging operation to determine the overlaps of the newly imported domain-specific model with the integrated model in the database. In order to identify model overlaps, i.e. similar nodes or sub graphs,

TRAILS uses three types of similarity calculation methods, each consisting of multiple approaches. First, model overlaps can be determined by calculating the similarity of the descriptions or captions of model elements. Examples of such approaches are String Edit Distance or Levenshtein Distance that reflect how similar the captions of two model elements are. Second, TRAILS is able to determine the similarity of model elements by evaluating their attributes. In general, model elements that have some identical attributes tend to be similar or at least closely related. And third, TRAILS uses a method we call context similarity evaluation. This method determines the similarity of two nodes based on the similarity of their neighbours. According to this method, two nodes have a high similarity if their adjacent nodes in the graph appear to have the same type, name or other attributes. In model-based engineering it is reasonable to expect that model elements with similar adjacencies are closely related or identical.

The different model similarity indicators can be combined flexibly to allow for optimal merging results. TRAILS then presents the results of the similarity calculation to the user ordered by a combined similarity measure. For each pair of likely similar elements the user can select to merge two nodes into one, link the nodes using an edge that describes their relationship (e.g. new version of) or ignore the similarity. By offering comparison algorithms that can be combined flexibly, TRAILS provides the means to implement automated procedures for traceability maintenance as proposed by Maeder and Gotel (2012) when it comes to changes along the engineering life cycle.

5.3. Adaptable Cross-domain Model Integration Ontology

As stated before, when importing models from external software tools that are described in a certain domain-specific modelling language or format, TRAILS maps those imported models onto a cross-domain ontology (cf. Section 4.1) that has the expressive power to integrate the viewpoints of multiple engineering domains.

Although the TRAILS model integration ontology incorporates concepts from various domain-specific modelling languages, it is not feasible to consider every modelling language or format ever invented. In fact, more or less every company uses some self-developed legacy software tools or data formats that they customized to their needs. They vary from customized and extended off-the-shelf engineering tools to simple spreadsheets enhanced by using macros.

In order to deal with the issue of having to interoperate with non-standard software tools and data formats, the TRAILS integration ontology can be modified and extended by the user according to individual, context-dependent needs. This feature allows the ontology to be adapted to specific needs of the application environment, such as specific industry or project characteristics. This way, it is also possible to define additional ontological concepts that are needed in order to import artefacts which are specified in further modelling languages or data formats that are not part of TRAILS standard implementation. However, if the integration ontology is altered, in some cases rules for model mapping have to be adapted as well. In TRAILS, the integration ontology as well as mapping rules can be accessed, managed and

modified directly within the graphical user interface. Furthermore, the files containing this information can be exchanged with other users.

5.4. Editing Models

Although TRAILS supports the user in producing an integrated model of the PSS from a number of different sources by offering smart merging algorithms, in some cases there is need for manual editing. This way, the user is able to add missing links or clean up duplicate information, both of which are very likely to cause inconsistencies within the product design and specification. Again, in some cases such inconsistencies ultimately lead to product failures and costly callback if they remain undetected.

Furthermore, it is possible to create new nodes, add them to the integrated PSS model and complement or adjust attributes. Hence, the user can add details to the system specification that could not have been expressed in the source modelling tools. By doing so, the user is able to enrich the system model by adding additional information that is required for e.g. change management or requirements tracing. Thus, instead of limiting itself to just importing, processing and interpreting data that has been created using other software tools TRAILS allows the user to edit or delete the nodes and edges that the imported models consist of.

5.5. Customizable Appearance and Standard Graph Layouts

The fundamental idea of TRAILS is to structure and illustrate the entire knowledge about the product or solution being developed and the development process itself as a semantic graph. This graph consists of nodes which represent any kind of artefact created in the engineering process as well as edges which represent different types of relationships among these artefacts.

As a consequence, the visual appearance of this graph determines the comprehensibility of the model and consequentially the usability of the TRAILS software tool. PSS engineers not only need to understand the increasingly complex technical products that are part of the PSS solution but also the dynamics of the business processes in which the service is provided to the customer. Therefore, intuitive presentation of engineering information and the possibility to interact with the semantic graph are crucial features. This way, TRAILS supports the user in revealing hidden information through rearranging the graph or highlighting certain nodes or edges.

Specifically, TRAILS allows automatically arranging the displayed graph in one of several pre-defined standard graph drawing strategies, such as circular or force-based layouts. Furthermore, the user can rearrange nodes manually and expand or compress nodes that contain sub-graphs.

In addition to that, the tool offers the possibility to customize the appearance of nodes and edges. The user can select the standard colours for each type of node and edge in order to facilitate visual differentiation. Moreover, it is possible to choose between multiple node shapes or embed individual icons or images for each type of node.

5.6. Customized Filtering and Viewpoint Creation

The analysis of integrated engineering information as it can be performed using TRAILS is in some ways a double-edged sword. On the one hand it is desirable to collect and integrate as much information as possible from various domain-specific models, documents and other sources. On the other hand, one quickly obtains a tremendously complex network of interrelated artefacts that in all its details is hardly comprehensible for the human analyst at first sight.

However, most analysis tasks only concern a minor fraction of the nodes and edges that form the integrated PSS model. In addition to various graph layout options, TRAILS provides a customized filtering feature. This feature allows the user to filter the graph according to node types, edge types and even attribute values in order to decrease the amount of information visualized to what is actually needed for performing the analysis. With the possibility to only display a certain fraction of the nodes and edges that form the graph the user is better able to e.g. follow the evolution of a particular requirement over time or oversee just the information flows within the PSS.

In TRAILS these customized filters are referred to as viewpoints on the semantic graph. These viewpoints can be defined manually by the user or loaded from pre-defined templates that serve specific purposes or reflect a certain role in the engineering process, e.g. the requirements analyst. Once defined, a viewpoint can be saved to the viewpoint selection for future use. In addition to simplifying the visual appearance of the integrated PSS model, viewpoints allow restricting access for certain roles, e.g. when engineering information needs to be shared with external stakeholders.

5.7. Matrix View and Spreadsheet Integration

In some engineering disciplines like mechanical or industrial engineering matrix-based engineering tools such as design structure matrices (DSM) or domain mapping matrices (DMM) are continuously popular. Although these tools are often naturally grown legacy systems, they are widely used across various industries and most companies use at least one tool of this kind in their engineering process.

For some use cases, like generating supplementary nodes or capturing a larger number of traceability relationships, matrices constitute a more convenient form of visualisation. Using matrices a larger number of nodes can be arranged in a space-saving manner allowing for a more compact overview of the overall system structure. Hence, besides the default graph view, TRAILS also provides the possibility to visualize the semantic traceability graph as a matrix. Similar to the graph view, TRAILS allows customized filtering and viewpoints in the matrix view as well. Moreover, all graph editing operation can be performed the same way using the matrix view. This way, the user is able to switch between both forms of visualisation flexibly using the perspective the fits best for the respective task (Tilstra et al. 2010).

Another reason why many companies still use such matrix-based engineering tools is that they can be implemented using standard office software for spreadsheet calculation. This way, matrix-based engineering tools can be flexibly used and easily adapted. In order to ease integration with such tools, TRAILS allows exporting the semantic traceability graph or parts thereof to common spreadsheet formats so that the data can be analysed using existing matrix-based analysis tools.

5.8. Multi-user Capabilities and Database Server

As technical products become more and more complex, so have the processes of their development. Today, the development of technical products usually involves a team of specialists from multiple engineering domains to design and integrate the various components, i.e. hardware, software and in many cases services. Hence, the typical engineering process and with it engineering software tools are increasingly shaped by the need for collaborative and concurrent team activities.

The need for supporting collaborative engineering through adequate tools is even more prominent when multiple companies are involved in solution design and service delivery. The various stakeholders from different companies sometimes distributed globally need to be able to work concurrently on a central instance that serves as a single point of truth for the integrated PSS model.

For this purpose, TRAILS provides the possibility to store all engineering data on a central graph database server. In our current implementation we use the open-source framework Apache Jena as database to store RDF data together with Fuseki Server for serving RDF data over standard internet protocols, such as HTTP. This way, multiple installations of TRAILS can be used concurrently and synchronize updates with the central database.

6. Case Study: Bike Sharing System

In this section, we explain a bike sharing system situation as a PSS example to clarify how TRAILS can support PSS development. The bike sharing system is not a hypothetical example but was implemented as a functioning prototype at our university with multiple departments collaborating extensively. Overall, service processes were designed, software components (such as central database systems and a mobile application) were implemented and the necessary modifications to a bike were engineered and built in order to set up a functional prototype of a bike that operates within a free floating bike sharing system. This way, it was possible to have control and unlimited access to the whole PSS engineering process which would not have been the case in a real industry example. Even though TRAILS was evaluated in a much more detailed case study than can be presented here, we admit that several industry case studies are desirable in order to evaluate the true potential of our software tool. Nonetheless, we think that a simplified version of our case study gives the reader a better idea of what TRAILS is and how it functions.

In order to understandably present our case we only describe the high level architecture of the system. In this conjunction, our goal is not to present every feature of TRAILS in detail, but

to give the reader a general overview of the idea behind our tool using an intuitive application scenario. Bike sharing systems are a typical example of a PSS with the aim of providing mobility as a service to customers. In our case study the PSS provider offers bikes on an on-demand basis to customers at multiple sharing stations within the city limits. The bikes can be rented by registered customers simply by entering their customer ID and PIN at one of the bike sharing stations and one of the bikes would be released. The customer can then use it on a pay-per-minute basis and then return it at the same or any other of the bike sharing stations. After returning the bike, the amount due is charged to the customer's credit card. From an architectural perspective, the bike sharing system is composed of stations that feature a keyboard and screen as a user interface. Besides, stations are equipped with an external power supply and they communicate with the back-office fleet management system using a mobile telecommunications module (UMTS module). Bikes can be locked to the stations via an electric lock. This lock is released when the customer rents one of the bikes. The back-office fleet management system, which operates in the background, is further linked to a central database as well as a payment system (c.f. Figure 37).

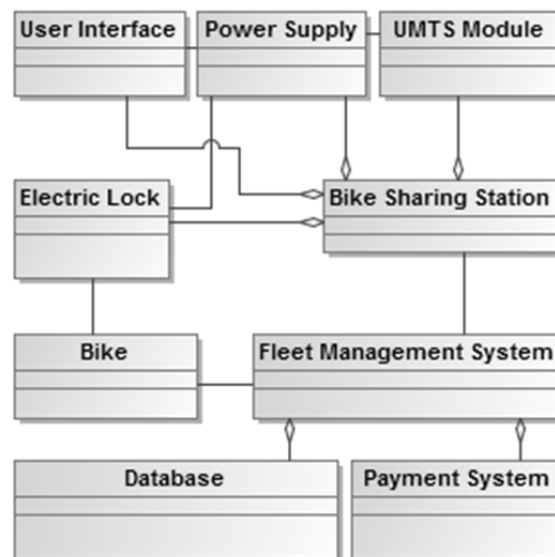


Figure 37: SysML block diagram of the stationary bike sharing system

As in a PSS we are not only dealing with products, but also services, to achieve a full understanding of the bike sharing system, it is necessary to consider the dynamic service processes (c.f. Figure 38). When a bike is needed, the customer walks to the next sharing station and enters his user ID into the user interface. The sharing then checks the user ID for validity by communicating with the central database. If the user ID is valid, the system will change to the next screen, where the customer can enter his PIN, which again will be checked for validity in the database. If this PIN is also correct, the sharing station then releases one of the bikes by opening its electric lock.

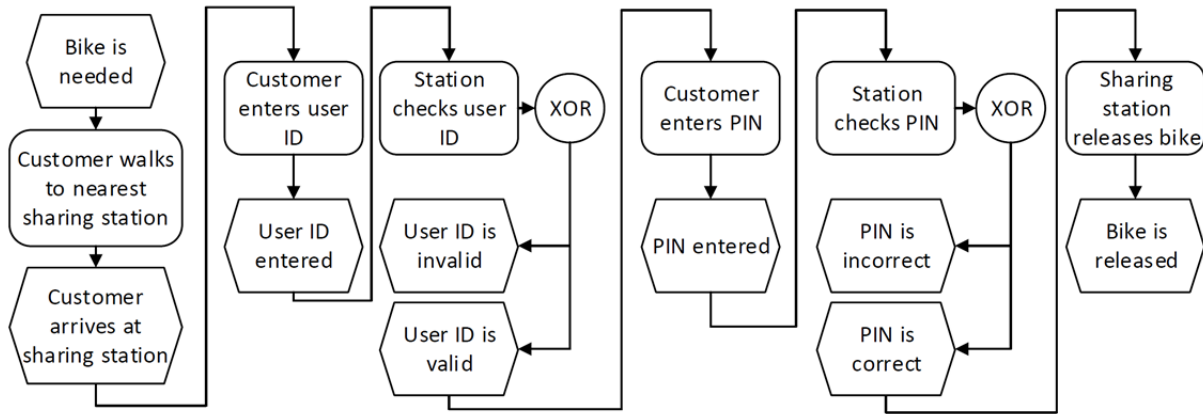


Figure 38: EPC of the rental process in the stationary bike sharing system

We consider the case that the bike sharing provider re-investigates market trends as well as technological possibilities in order to improve its customer service and reach out to new market segments. The bike sharing provider reaches the conclusion that it should change its business model from offering stationary bike sharing to offering a free-floating system where customers can pick up or leave the bike anywhere in the city area. Figure 39 depicts a requirements document containing selected requirements for the free-floating system that influence some of the existing structural components and services processes which need to be adapted according to the new business model.

Requirements Document		Free Floating Bike Sharing System.reqif
Description		
1	Ⓡ	Free floating bike sharing system
1.1	Ⓡ	Functional Requirements
1.1.1	Ⓡ	Fleet Management System
1.1.1.1	Ⓡ	Tracking bikes
1.1.1.2	Ⓡ	Manage users
1.1.1.3	Ⓡ	Manage payments
1.1.1.4	Ⓡ	Manage reservations
1.1.2	Ⓡ	Bikes
1.1.2.1	Ⓡ	Continuously send location to Fleet Management System
1.1.2.2	Ⓡ	Check user credentials
1.1.2.3	Ⓡ	Open electric lock
1.1.2.4	Ⓡ	Close electric lock
1.1.2.5	Ⓡ	Enter reserved mode
1.1.2.6	Ⓡ	Exit reserved mode
1.1.3	Ⓡ	Smartphone App
1.1.3.1	Ⓡ	Get bike locations
1.1.3.2	Ⓡ	Update bike locations
1.1.3.3	Ⓡ	Display bike locations on map
1.1.3.4	Ⓡ	Show account status

Figure 39: Excerpt of requirements document for free floating bike sharing system

In order to satisfy these requirements, architecture of the bike sharing system needs to be adapted. As the sharing system no longer depends on fixed stations, there has to be an

alternative way for customers to locate the nearest available bike. In the new system architecture a smart-phone application that customers can install on their device will display the location of bikes on a map. This smartphone application needs to be able to communicate with the central fleet management system, which will store and continuously update the position and availability of nearby bikes. Furthermore, the user interface and electric lock that used to be part of the fixed sharing stations now need to be integrated into the bikes themselves. In order to be able to communicate with the back-office fleet management system for updating the bikes location or checking the users' credentials the bikes need to be equipped with a UMTS module as well as a GPS receiver. All of the electric equipment furthermore needs to be powered by a battery attached to the bike.

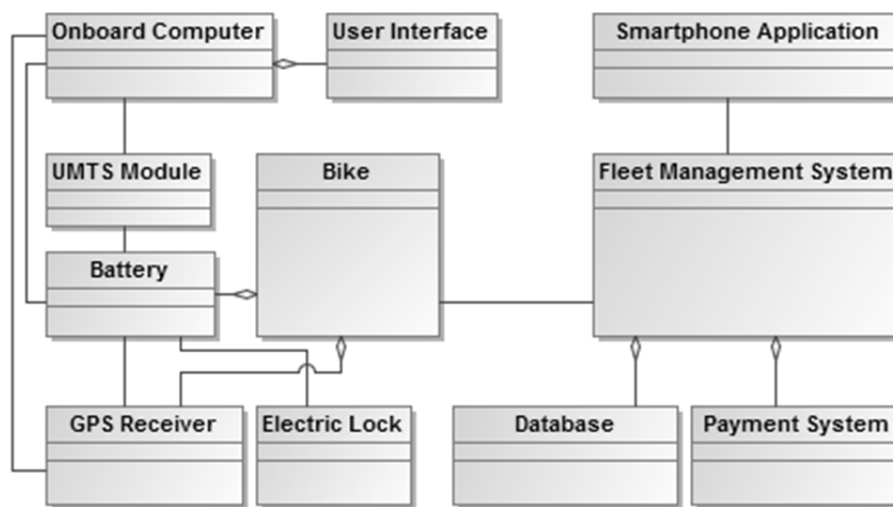


Figure 40: SysML block diagram of the free floating bike sharing system

Apart from the system structure, the service processes need to be adapted as well. As an example, we again take the rental process (c.f. Figure 41). This process has been adapted in order to fulfil the requirements of the new business model. If a bike is needed the customer first opens the bike sharing application on his smart-phone, which then checks for nearby bikes and displays them on a map. The customer can then select one of those bikes and reserve it for the next few minutes, until he reaches the bike. A reservation request is directly sent to the fleet management system (FMS), which then updates the bike's status in the database and forwards the reservation request to the bike. When the customer then reaches the bike, he can proceed as normal by entering his user ID and PIN into the user interface attached to the bike. The credentials are sent to the fleet management system and if correct, finally the electric lock opens.

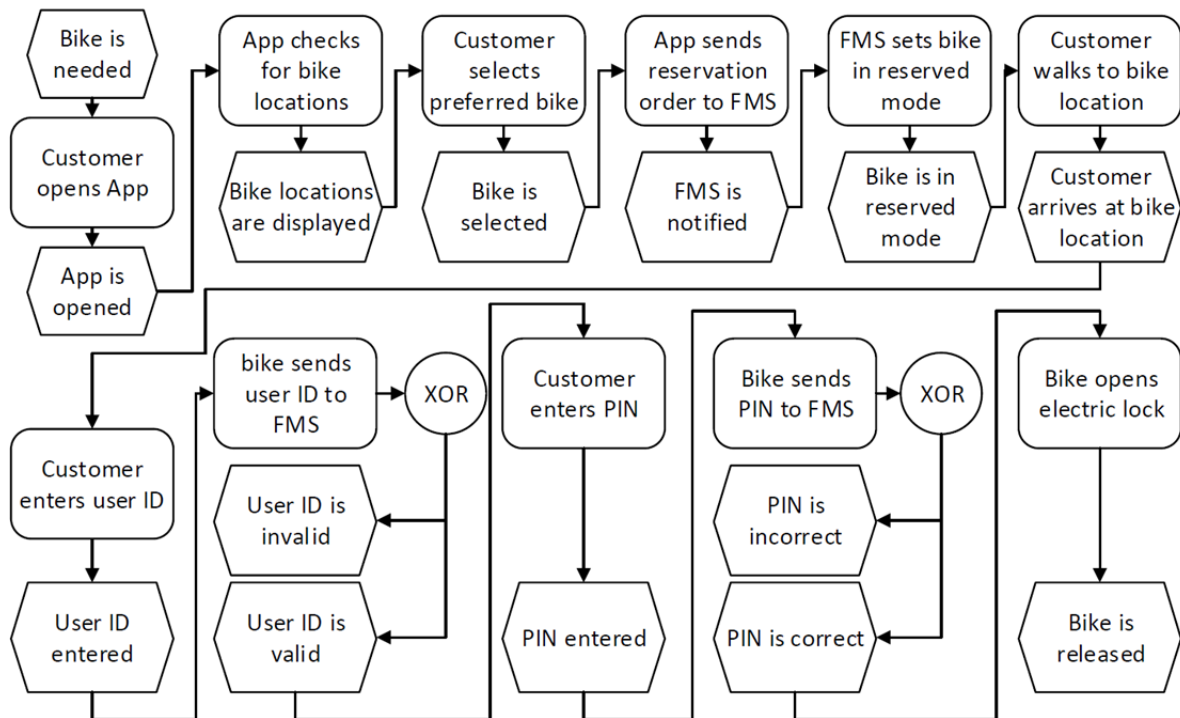


Figure 41: EPC of the rental process in the free floating bike sharing system

In order to record the changes of the system architecture as well as the rental process in TRAILS we need to capture the changes in all affected development artefacts, i.e. the SysML block diagrams, the EPC process models and the ReqIF requirements document. As we map the different versions onto each other and also link the altered requirements to the solution artefacts that fulfil those requirements, we can easily understand which parts of the system need to be adapted and how. In order to do so, the different versions of the SysML block diagram need to be imported into TRAILS. When merging several source models into one comprehensive description of the system and its evolution, TRAILS offers several comparison algorithms that support the user in identifying model overlaps and common elements. Figure 42 shows this step in the process of merging the block diagram of the stationary bike sharing system being merged with the altered requirements document.

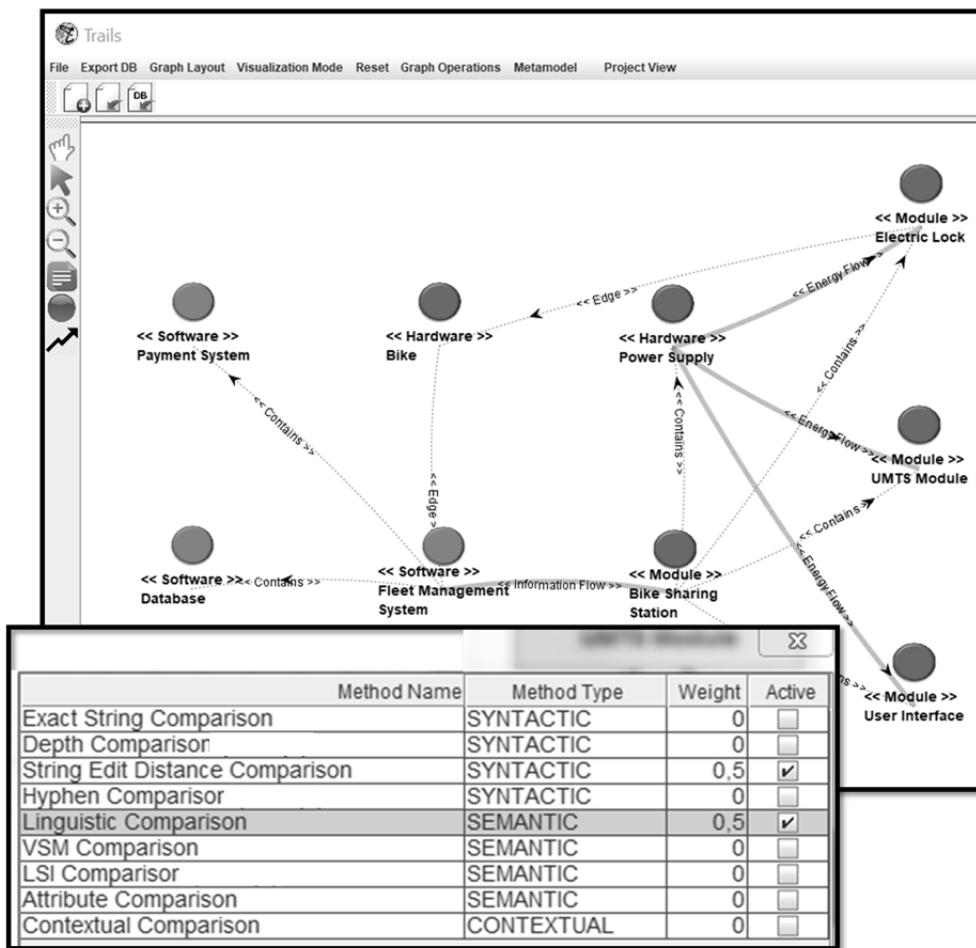


Figure 42: Configuration of comparison algorithms for merging models

At this stage, the user can select which of the comparison calculations (as introduced in Section 5.2) should be performed. Here, it is also possible to select multiple comparison algorithms and calculate a combined weighted similarity score. TRAILS then evaluates the similarity of elements within the two models that are being merged and displays them as a sortable list. The user may then decide which of the pairs of nodes should be linked or merged into one. The result of comparing the block diagram of the stationary bike sharing system with the requirements document can be seen in Figure 43. For this merging process an equally weighted combination of vector space model comparison and string edit distance was chosen in order to identify similarly named entities. In this example we see that even though the algorithm has to operate on a very simplified requirements document that contains only captions and not the requirement descriptions themselves, we can easily identify solution artefacts and requirements that are related.

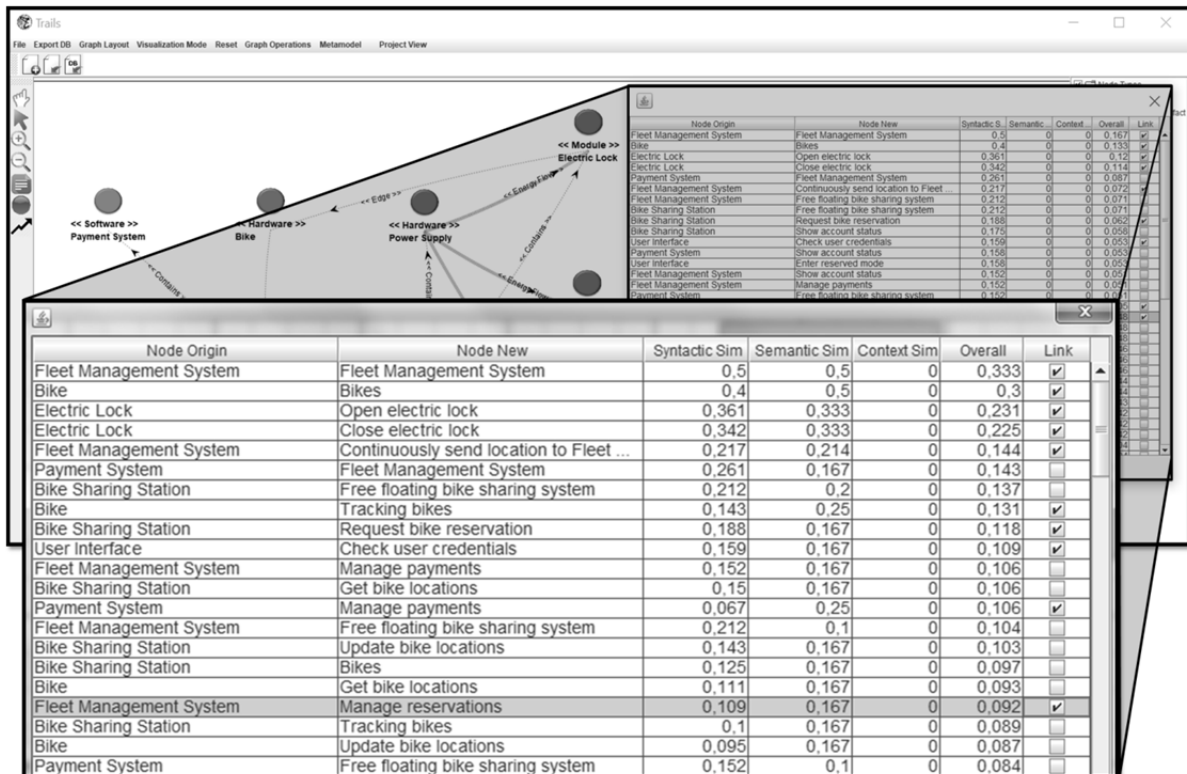


Figure 43: Merging Results of Requirements with Block Diagram for a Stationary Bike Sharing System

Having merged the specification of the system architecture, TRAILS provides a visualisation of the combined model. This way, the user can explore the model visually in order to get a clearer understanding of the inherent system structure and complement missing links between related elements of the models. As illustrated in Figure 44, TRAILS not only shows which of the requirements impact a certain solution component but also shows the internal break down structure of the requirements document. Consequently, the user is able to deduce the dependencies between individual requirements. In this context the user can furthermore chosen which types of semantic relationships to highlight or which to fade out, thus increasing clarity of the visualised dependencies.

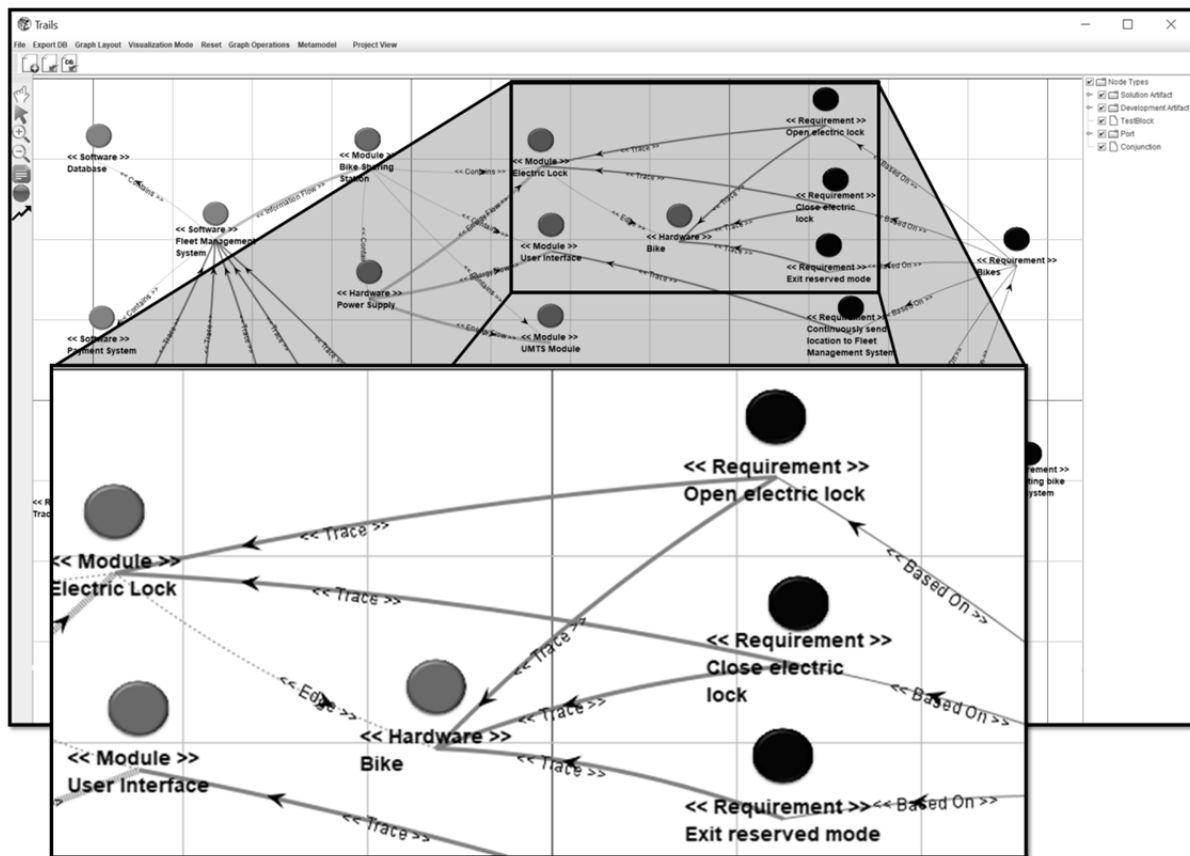


Figure 44: Linking Requirements to Solution Components

To trace the changes regarding the system components that result from evolving the bike sharing business model from stationary to free float, a next step would be to merge the current system architecture as depicted in the SysML block diagram with the future architecture that was specified according to the new requirements. By doing so, engineers can see at first glance which components have been changed. However, changes not only manifest themselves in the architectural setup of the system, they also incur in the system behaviour, i.e. the business processes. Again, changes in the business processes may impact the system architecture. Thus, it is advantageous to visualise which components are involved in performing certain service processes. Figure 45 shows the result of merging the SysML block diagram that reflects the structure of the PSS with the EPC diagram illustrating the bike rental process. Through visualisations like this, engineers can easily identify which components are likely to be impacted by a change in the business processes.

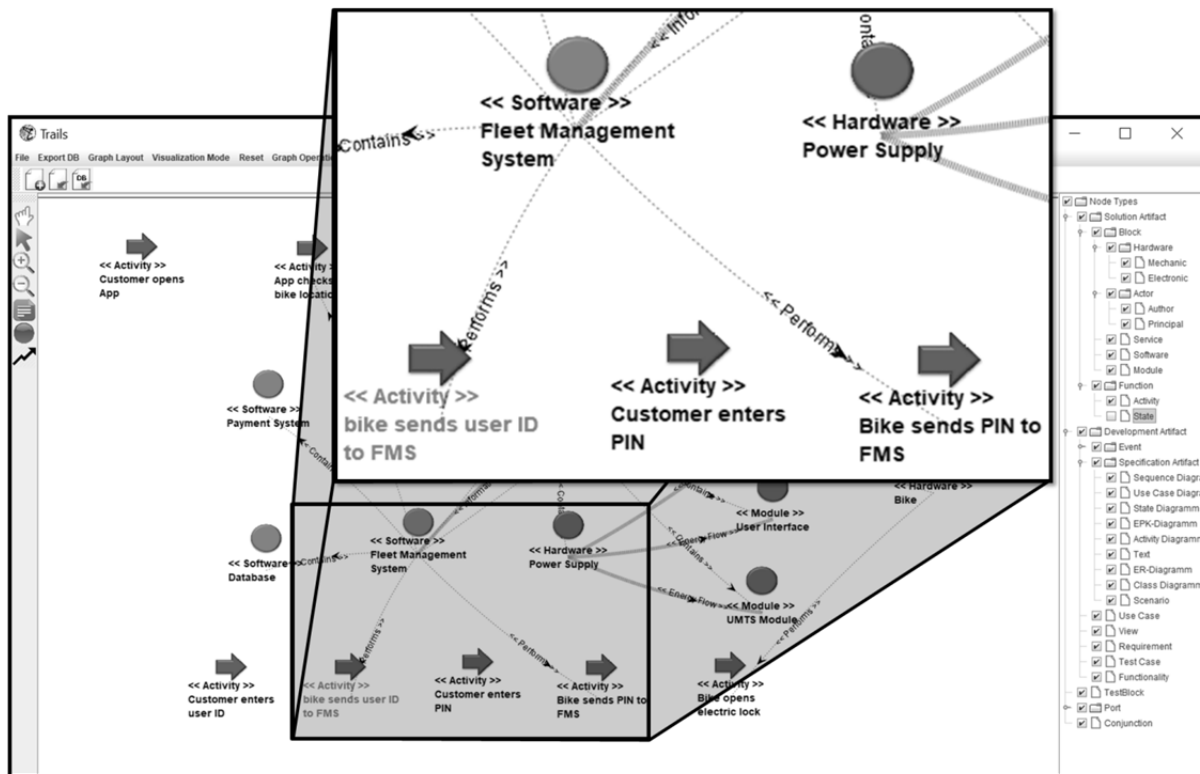


Figure 45: Linking activities to system components

Although the models in our case study are only a very rudimentary description of a bike sharing system, they permit some insights into the traceability and model integration support that is offered by TRAILS. Once imported and merged into a comprehensive system model, TRAILS stores the resulting semantic graph in a central server-based database that can be accessed by multiple clients. Throughout the development process new versions of the solution or development artefacts can be continuously merged with the existing model on the database. TRAILS will then constantly update the evolution of these artefacts and enable users to query the semantic graph in order to get new insights.

7. Discussion

As discussed in the first section of this paper, there is no tool that supports integration and holistic analysis of heterogeneous PSS engineering artefacts along with the dependencies between those. To address this issue, we developed a tool prototype that enables integrating models from different domains of PSS engineering, the visualisation of the relationships among the merged elements and managing the changes through a version management mechanism. In this section, we discuss the strengths and weaknesses of this tool prototype including its features and the methods and technologies employed.

Capturing of PSS Artefacts in a Semantic Engineering Graph

As its main functionality, TRAILS allows capturing the relationships between all types of artefacts like actors, use cases, decisions, process activities, product components and so on. The graph-based presentation enables easy understanding of the dependencies among an within artefacts. While most modelling tools only allow the user to view the relationships of

element within a specific model and others like document management systems focus on the evolution of as well as the relationships between certain documents, TRAILS is able to do both, using a graph representation the captures the semantics of engineering knowledge. However, this advantage can only be realized if TRAILS is capable of processing various domain-specific meta-models and tool-specific data formats.

Traceability and Change Management

TRAILS keeps the history and evolution of the artefacts by recording the reasons for changes, the stakeholders involved in realizing a change and all versions of the artefacts. Furthermore, TRAILS does not only track which models have changed but also which of the entities within a model have changed. Therefore, engineers are not only supported in understanding the current structure of the PSS but also they are able to follow its evolution. Consequently, our tool may increase the awareness of stakeholders during development, which leads to a higher acceptance of the design decisions and supports making better decisions in further development. Although its fundamental technical architecture would allow a further in-depth analysis based on e.g. logical reasoning, TRAILS is however currently dependent on the user to estimate the further consequences of changes in requirements or solution artefacts and to document the reason for a change.

Extendible Ontology

Model integration and transformation in order to identify the semantic relationships between entities of heterogeneous models are a central concept of TRAILS. To this end, the integration ontology (see Section 4.1), forms the backbone of model integration by playing the role of a middle language. Moreover, the integration ontology is designed to be adaptable to specific organisational requirements. Therefore, it essentially defines generic types of artefacts, which are common in the development of PSS. Along with this, the ontological entities that these artefacts contain as well as the types of semantic relationships that can exist between them are defined. In addition, hierarchical inheritance structures of artefacts within the ontology ensure the compatibility of TRAILS with many modelling languages. However, at the current stage there are some limitations. To this end, undefined semantic relationships in the integration ontology are resolved through abstraction during the integration process. This means that if a certain type of node or edge is unknown in the integration ontology and no transformation rules have been defined for this situation, the rules for the parent node or edge type are applied. In this case, TRAILS will still be capable of importing the artefact but some of the semantic information that is contained in the original model can be lost, e.g. a very specific type of relationship is replaced with a more generic type.

Visualisation

Another key idea behind TRAILS is that visualisation of the integrated engineering models in a way that the structure and dynamics of a PSS can be understood intuitively. By presenting the integrated engineering knowledge in an appealing and convenient form to the user, they can perform visual analysis of the models making use of the fact that humans are capable of visually detecting complex patterns than machines can't. Accordingly, the main features actualizing this goal are: graph layouts and customizable shapes for graph elements. Besides, TRAILS provides a matrix presentation, which automatically is derived from the graph-based

presentation. In addition, user can create custom views that highlight some types of nodes and edges while hiding others. However in this context, visual analytics should not be seen as a substitute for rule-based analysis or reasoning but more as a complementing instrument.

Simulation

In many cases, there are various alternative PSS designs combining different features of products and services. As every design decision leads to a different performance and costs for the final PSS, it is of high importance to evaluate and prioritise PSS design alternatives (Alfian et al. 2014). However, service components of a PSS impose highly stochastic behaviours to the system, which makes evaluation of a PSS design challenging (Kimita et al. 2012). PSS literature widely proposes simulation as the method to assess PSS designs. To this end, numerous interdependent aspects and measurements need to be addressed in a PSS simulation, such as product and service usage factors e.g. usage frequency and duration, life cycle-related factors, e.g. reusability and maintenance, and environmental impacts (Kimura and Kato 2002; Garetti et al. 2012). PSS simulation thus needs to consider various of these factors in order to deliver realistic results.

As argued by Zacharewicz et al. (2017) tight model alignment is an essential prerequisite to analyse dynamic dependencies through simulation. In its current version, TRAILS aims at enabling a rather loose coupling between domain-specific modelling avoiding complex and hard to maintain interfaces (or connectors) between different domain-specific modelling tools.

As TRAILS enables integrating different models such as life cycle models as well as structural models, it establishes the basis for such complex simulations. To this point however, the simulation has not been the focus of our work as we primarily aimed at supporting the model-based engineering approach before implementing functions that support model-driven engineering.

However, we believe that TRAILS is perfectly suited to host a simulation engine (e.g. based on system dynamics) that allows analysing dynamic dependencies, such as the impact of resource availability on the result of business processes.

Comparison with other Tools

To better clarify TRAILS scope of abilities, it is necessary to compare it with other types of existing tools. Therefore, in the following we compare TRAILS with two major similar types of tools.

There are a plethora of **Requirements Engineering Tools** available such as DOORS¹², Rational Requisite¹³, Integrity¹⁴, etc. However, TRAILS does not aim at general management of requirements, but enabling traceability of requirements through models of development artefacts and processes. In addition, as Trails has been designed to support managing the life

¹² <http://www-03.ibm.com/software/products/en/ratidoor>

¹³ <https://www-01.ibm.com/software/in/awdtools/reqpro/>

¹⁴ <http://www.ptc.com/application-lifecycle-management/integrity>

cycle of a PSS with a focus on the development stage, it gives a high level understanding of how each requirement is being satisfied by the PSS.

The other category of tools which TRAILS can be compared is general **Modelling Tools** like Visual Paradigm¹⁵, Microsoft Visio¹⁶, etc. General modelling tools usually illustrate distinct viewpoints on a specific real world concept, i.e. the component structure of a hardware product or the logical order of activities within a business process. If the intention is to understand a certain part of the PSS from a specific viewpoint, one should use such modelling tools. However, TRAILS takes a different approach by integrating multiple models or perspectives into a system under development as well as the development process itself. Therefore, TRAILS enables stakeholders to comprehend the inter-dependencies within the PSS components.

In summary, the current version of TRAILS mainly offers the architectural foundations to implement advanced engineering intelligence features. Today, its core functionality is to import various models specified in DSML and data formats into a comprehensive PSS model that comes as a semantic graph. In doing so, TRAILS relies on the resource description framework as a format for representing the engineering information within a semantic graph. It is therefore of a great importance to draw the limitations of our approach, not only to better reflect the scope of this study, but also to expose limitations which we want to encounter in future work, as we discuss subsequently.

8. Conclusion and Future Work

PSS are complex socio-technical systems that containing multiple physical, software and service components that need to be seamlessly integrated in order to deliver the desired value-in-use to the customer. Consequentially, the development and life cycle management of PSS demands stakeholders from various engineering domains to work together with each of them using special development methods, modelling languages and engineering software tools. As repeatedly stated in literature, visualisation and analysis of relationships between the different engineering artefacts is essential for stakeholders to understand the interdependencies among components of the system. To close this gap, we presented the our software tool, TRAILS, which enables integrating models from different engineering domains, capturing and visualising the semantic relationships among and within merged engineering artefacts.

To this end, first we focused on developing an integration ontology, which acts as a meta-model to enable model transformation and integration. Besides, the proposed ontology can act as a traceability reference model capturing trace links between various engineering artefacts. However, in order to make the tool more flexible in terms of compatibility with third party engineering tools a desirable feature for the further development of TRAILS is the possibility to specify model transformation rules through drag and drop combination of atomic transformation operators. This way, domain experts can add new DSMLs or data formats to

¹⁵ <https://www.visual-paradigm.com>

¹⁶ <https://products.office.com/visio/>

TRAILS without having to touch the software code by just configuring the transformation process.

In a case study example we presented a comprehensive PSS model that consists of only five sub-models (two of them being modified versions of already existing models). However, in a real industrial case, such a model comprises a much higher number of different sub-models, each of them featuring a higher level of detail. Thus, in a realistic setting, the semantic graph would accumulate to several thousand nodes. As a consequence, TRAILS features like role-based and intuitive filtering need to receive increased efforts. In this context we also see many promising prospects in further developing TRAILS to realise the potentials of visual analytics. If engineers are capable of intuitively comprehending the dependencies within a system, they will most likely be able to provide better solutions.

As next steps, we also will add enhanced team work features to TRAILS. As development of PSS involves high number of people, features to support collaborative work is necessary. Collaborative engineering features are critical for an engineering tool like TRAILS. Currently, the TRAILS back-end database server has limited support for several concurrent users and managing different versions of a PSS structure caused by editing of different users. Regarding better multi-user support, the TRAILS back-end offers various potentials for enhancement including secure user management and data transfer (currently data is served via HTTP), a revision control system, change management and update broadcasts in order to better suit the needs of concurrent engineering.

Since TRAILS uses RDF to represent engineering knowledge within a semantic graph, there is the possibility to enhance the tool by exploiting other standard semantic web technologies, such as SPARQL, OWL or RIF. By doing so, TRAILS can be equipped with advanced semantic search functions. Besides, an inconsistency management mechanism can be developed. For example, based on a user-defined set of rules, inconsistencies among PSS elements, which are imported and merged from domain-specific models, can be detected. Right now there is no automatic inconsistency detection and resolution. Hence, manual inspections are needed to identify conflicts between different models. However, manual inspection is often an exhausting and complicated process suffering from human faults that lead to inaccuracy and incompleteness.

In future, TRAILS should be able to automatically check or support the manual inspection in order to identify the semantic traceability graph inconsistencies. Such inconsistencies could be violation of fundamental physical or logical laws (deadlock in a work-flow because of cyclic control flows; self-containment; negative component weight; dimensions of a component are bigger than its containment; incorrect conversion of measurement units), a mismatch between a requirement and the solution artefact that is supposed to fulfil this requirement, contradicting requirements that refer to the same solution artefacts, an unbound requirement (requirement that is not fulfilled by any solution artefacts), a solution artefact that does not fulfil any requirement (over engineering), a mismatch between the attributes of a subsystem and its components (the aggregate weight of components is higher than the specified weight of the compound). Also, we will implement functionality that aims at using

the information captured by TRAILS to simulate service provision in order to optimize the PSS overall architecture.

To sum up, we argued that complex engineering projects, such as the development of PSS that involve a variety of engineering domains have need for a tool enabling the integration of heterogeneous engineering artefacts into a comprehensive model for purposes like traceability, change management or various analysis tasks. Having introduced our overall concept to tackle this issue, we introduced our prototypical tool TRAILS and presented its core features. We then demonstrated the possible application of TRAILS in an academic PSS engineering project using the example of a bike sharing system. Discussing the current development state, the basic concepts and comparing TRAILS to other types of engineering tools, we concluded that TRAILS is subject to a number of limitations and showed potentials for further enhancing the tool. Nevertheless, in summary the holistic concept of TRAILS, i.e. enabling the integration of various heterogeneous engineering artefacts through abstraction and model transformations provides a fundamental value-added for various stakeholders within the life cycle of a PSS.

Acknowledgements

We thank the German Research Foundation (DFG) for funding this work as part of the collaborative research centre Sonderforschungsbereich 768 - Managing cycles in innovation processes - Integrated development of product-service-systems based on technical products' (SFB768).

PART C: DISCUSSION

1 Discussion

As digitalization initiatives gain pace in many companies, more and more engineering knowledge is being codified in various forms of digital documents, models or in data bases. This knowledge can be automatically interpreted by machines if it is represented in a properly structured format. By doing so, the large potentials of proper knowledge management can be realized in engineering and companies that are making use of this knowledge are awarded with competitive advantage (Probs et al. 2010). If the knowledge of the individual engineer is being preserved and organized, it can be turned into organizational knowledge that can be reused by others. In general, the reuse of knowledge leads to improved processes and better final engineering designs (Hicks et al. 2002, Rezayat 1999). A key pre-requisite for proper knowledge capturing, management and consequentially knowledge reuse is comprehensive traceability of engineering artifacts. It allows to join and connect information from various sources and hence allows for knowledge to become contextualized (Ramesh 2002). The concepts developed in this thesis aim at exactly this target. In the following we summarize and discuss the results contained in the publications of this thesis. We further explain possible implications for research and practice, we identify some major limitations of our research and finally we highlight some promising starting points for future research.

1.1 Summary of Findings

In Publication 1 we identify and explain nine types of features that differentiate the development of PSS from traditional engineering. First of all, the development of a PSS is coined by the need for seamless (1) integration of the product and service components. Additionally, these components are characterized by (2) different life cycles and furthermore many other cycles shape development and service provision. Thus, the development of a PSS as well as service provision require an (3) intense collaboration of different engineering domains. As PSS impose (4) solutions to individual customer needs they often need to be designed as modular system architectures. The individual combination of PSS modules, sometimes individually for each customer, further results in a (5) high variability of service provision. Consequentially, a (6) high degree of customer integration is necessary in both, development and service provision. Moreover, PSS development as well as service provision imposes (7) organizational challenges and it drives the importance of (8) value network integration and in many cases, PSS are associated with a certain (9) sustainability goal.

These characteristics of PSS engineering also lead to a similar number of design recommendations that PSS providers should follow. In particular, PSS providers should focus on offering solutions that are both, economically and environmentally sustainable, by focusing on the essential needs of the customer. In order to do so, they need to identify the essential solution independent requirements at the early stages of the development process and consider various alternatives of service provision models. This strategy also implies to actively integrate the customer along all phases of the PSS lifecycle. Since the development of a PSS requires an intense collaboration of different engineering domains, PSS providers need to foster the collaboration between stakeholders of all those domains. One important

factor to facilitate this collaboration is to implement a traceability strategy and manage the interdependencies between the domain-specific development artifacts.

Having identified major features, that differentiate the development of PSS from traditional products or services, we used this fundament to determine, whether existing Traceability approaches were suited for PSS engineering (cf. Publication 2). Our analysis showed, that none of the existing approaches was recommendable for PSS without restrictions. At the same time, each approach has a certain characteristic that is advantageous for a certain task of traceability in PSS engineering. Hence, we concluded that systematic combination and enhancement of those approaches offers great potential to develop an approach tailored to the features of PSS development.

From a process perspective, traceability is in general concerned with three fundamental tasks. First, trace links and other kinds of traceability information need to be captured. Second, they must be kept up to date continuously and third, one needs to ensure that traceability information is being used properly. Since these tasks are largely independent from the specifics of the development project under consideration, this thesis primarily focuses on viewing traceability from a conceptual perspective and building conceptual models and tools that can be used along the traceability process as a whole. A first major building block in this context is a common model or vocabulary for PSS engineering that depicts development and solution artifacts that are essential to ensuring traceability as well as their relationships among each other. Such a model can be realized in form of an ontology that specifies the semantic relationships between the various ontological concepts that appear in the context of PSS development and service provision. A major difficulty in designing such a model is to make it universal and at the same time adaptable to the specifics of an organization or a development project if needed. Hence, to derive such a comprehensive yet customizable traceability reference model for the development of PSS, we also resorted to the expert interviews and case studies we conducted in multiple industries in order to identify the various needs of engineers, software developers project managers and other types of stakeholders. The single models, that resulted from this step were later to be integrated into a single model.

The results of one of these case studies are illustrated in Publication 3 showing the relationships between the central artifacts of requirements engineering and engineering change management. In this context we found for example, that an engineering change can be viewed as an abstract concept that manifests itself generally in three types of documents. A change proposal evolves to a change request and after a final decision turns into a change order that is to be executed. Furthermore, changes can relate requirements as well as solution artifacts or even production artifacts. In another industry case study (cf. Publication 4) we analyzed what kind of requirements traceability data structures are needed when following an agile approach in the engineering process. During this case study we developed a data model that allows traceability from requirements to solution artifacts and tests within an agile project setting.

The case studies we conducted show, that there is a further challenge to overcome when developing a reference model for traceability and providing adequate tools help support

practitioners in implementing it. As discussed in publication 1, the development of PSS requires an interdisciplinary system understanding and development process. However, in practice each engineering domain commonly uses specific modeling languages and tools, which focus on certain aspects of the system.

There are two ways of solving this issue. First, one could develop one new modeling language for each and every purpose that fulfills the requirements of all engineering domains involved. However, such a true universal modeling language would not only be hardly possible to develop, it would also be impossible to learn for the developers due its inevitable complexity. Such a language would be over-engineered for almost every single purpose. The second way of solving this "integration" issue is much more feasible and desirable at the same time. Its basic idea to abstract from the various specialties each modeling language has to offer and only extract those parts of the artifact descriptions that are required for traceability.

In Publication 5 we therefore introduce a concept for a cross-disciplinary model integration ontology that enables the various domains involved in the development of a PSS to map the elements of their specific modeling approaches to a joint representation. By doing so, we abstract from the detailed grammar (meta-model) that is inherent to any domain-specific modeling language and focus on building a categorization system that structures the various artifacts and relationships that are being used in these languages.

As a next logical step, we develop a conceptual methodology for model transformation using this integration ontology (cf. Publication 6). By using various kinds of development artifacts, such as use case diagrams, class diagrams, business process models or other domains specific models that are generated during the engineering process anyway, since they are used for other purposes, it is possible to keep the overhead efforts for ensuring traceability down to a minimum. Furthermore, by transforming multiple models of a PSS that are created by the different involved domains (e.g. mechanics, electrics/ electronics, software and services) the information and modeling artifacts can be (re-)used by the different domains. The presented transformation approach is based on the integration ontology introduced in Publication 5. It can handle both, structural as well as behavior models. Besides supporting the capturing of trace links in PSS engineering as well as keeping them up to date, our semi-automated approach for the integration of PSS models allows for a more sophisticated and comprehensive system view for all stakeholders that can be of aid in many areas of PSS development. Especially for the successful integration of the different components that constitute a PSS, a joint system model that allows understanding interdependencies is essential.

With the integration ontology presented in Publication 5 and the model transformation approach introduced in Publication 6 it is now possible to derive a reference model for traceability in PSS engineering whose information requirements can be fulfilled with relatively limited effort since existing data sources (especially models) can in many cases be tapped automatically. The reference model we propose in Publication 7 specifies the artifacts relevant for traceability in a manner that is abstracted from the various domain-specific representations. Furthermore, our reference model defines the semantic relationships

connecting those artifacts. In this context, the hierarchy of the types of artifacts and semantic relationships is aligned with the integration framework presented in Publication 5.

Since the practical applicability of a traceability reference model is largely determined by its flexibility and adaptability towards a specific engineering context, structures the proposed traceability artifacts and corresponding semantic trace links into several granularity levels. If needed, additional artifact types and types of semantic relationships can be added at each level or their required or optional attributes can be adjusted. This way, the structure of the reference model remains extensible.

Finally, in Publication 8 we introduce our prototypical software tool TRAILS, which aims at supporting PSS engineers in ensuring traceability in complex engineering projects which involve stakeholders from different engineering domains. TRAILS allows to import various types of domain-specific specification artifacts from different third-party software tools and join them into a common representation that conforms with our PSS model integration ontology introduced before and display the result as a graph or table. Furthermore, TRAILS allows to edit imported models (or specification formats, respectively), it offers a customizable appearance with standard graph layouts and it offers customized filtering of the ontological entities and semantic relationships.

1.2 Implications for Research

We believe that the analyses, concepts and solutions presented in this thesis contributes to several fields of research, most prominently (1) requirements engineering, (2) model-based systems engineering and (3) product service systems research. Overall, perhaps the most important contribution of this thesis is that our approach involves insights and concepts from all of these three research fields. We strongly believe, that digging into the results of this thesis, might serve as an example for researchers from all of those fields to think outside the box and look for potential use cases for the concepts and solutions they develop or, if facing a problem search for inspirations in any other field. In the following we discuss the contributions of this thesis to each of the three fields of research:

Contribution to Requirements Traceability Research

Requirements engineering is in many cases still seen as a rather document driven domain. In this context, requirements are often specified using natural language text documents. However, since solutions are increasingly designed and engineered according to the principles of model-based engineering, the research on requirements engineering more and more focuses on how requirements can be specified using other kinds of artifacts, such as (semi-)formal diagrams or even mock-up sketches and videos.

Still, today the quality of a requirements document is believed to manifest itself in the degree to which requirements are formulated unambiguously or not to say legally unassailable. By writing bullet proof requirements documents that can contain thousands of pages, principals strive for legally binding documents to guard themselves against contractors falling short of what they promise. In such documents, every little detail is defined to a hair, making it difficult for developers to even comprehend what kind of system is desired. Requirements

engineers therefore need to ask themselves, whether this situation is exactly what makes development projects miss their goals.

In this context, we believe that requirements traceability should be about more than simply being able to check whether all requirements as specified in documents are satisfied by the solution design and that there are test cases to prove this. In fact, we view semantic trace links as a multi-faceted tool whose application area ranges from change management and knowledge management to project monitoring. Looking into research on requirements traceability however, we see that today the domain is mostly focused on software engineering. As a consequence, we are among the first, to extend the perspective of requirements traceability to a cross-domain setting, such as the development of PSS.

Researchers can build upon our analysis of traceability approaches in various domains to develop methods that fit the need of complex cross-disciplinary engineering projects. Also, we present a traceability approach that is not only knowledge-oriented instead of document oriented but also helps to pave the way to using artificial intelligence in requirements engineering. As a beneficial by-product, using ontologies and semantic web technologies for ensuring requirements traceability, opens a whole new world to requirements analysis an automated reasoning regarding the effects of changes.

Contribution to Research on Model-based Systems Engineering

Not only in industry practice, but also within the research community, the model-based systems engineering paradigm increases its popularity continuously. Especially in academia however, the model-based approach is often associated with the engineering of technical systems, especially the development of cyber physical systems, only.

A pivotal element in this context is SysML. Accordingly, the Systems Modeling Language is used in more and more industry sectors and application areas in general. Within the course of this development, SysML is being confronted with an increasing amount of requirements regarding its expressive power. As engineers shift from other modeling techniques to SysML, the complexity of the standard grows. As a counter measure to that, researchers are already working only SysML dialects with limited expressive power which are easier to learn. Anyway, the need for integration of various types of model artifacts and formats is continuously high as no graphical modeling notation, such as SysML will ever have absolute expressive power and if it attempts to, it will be useless because of complexity.

Perhaps our core contribution is that our approach avoids this dilemma by abstracting the specification details for single components that are captured in domain-specific models from the the more generic information about an engineering artifact captured by an integrated comprehensive PSS model that is needed for collaboration among the engineering domains involved. This way, our approach facilitates the integration of domain-specific models by abstracting from specific details of the modeling language or data format, that are not relevant in the integrated model perspective. In this sense, researchers can use our model integration ontology as a blueprint approach for linking knowledge in model-based systems engineering.

Based on the example of PSS engineering, our research shows how popular concepts from the model-based systems engineering field can be applied to cross-disciplinary engineering and we provide a prototypical tool for this purpose. Moreover, we show how traceability can be realized on a much more detailed level in the context of model-based systems engineering than through just SysML Requirements Diagrams or textual references which are still currently used in requirements specification documents.

We further contribute to model-based systems engineering by widening the scope of the domain. The overall goal of model-based systems engineering, as it is commonly described in literature, is to provide plans for a physical object which is then produced. From our perspective however, this limited scope falls short of representing dynamically changing systems such as PSS. In such systems the physical and software part are only means to an end and the service that is provided to the customer and with it, the business model plays the dominating role.

Finally, our approach shows how to use ontologies and semantic analysis in a model-based systems engineering context. In this regard, it is important to find the right balance between comprehensiveness and level of detail of the model representation. While higher fidelity in the specification allows for deterministic automated model analysis to provide more precise analysis results, more generic model representations need to resort to heuristic stochastic analysis techniques, which are in general less exact. However, comprehensiveness of the model allows for a much broader analysis, revealing hidden and indirect dependencies that remain unrecognized if only domain-specific models are regarded independently from each other. Overall, we believe that the combination of these two research fields: model-based systems engineering and semantic technologies can lead to promising applications of artificial intelligence in engineering.

Contribution to PSS Research

As argued before, PSS models are mostly rather high level and used more for ideation in early development stages. After the essential PSS components have been identified, development mostly takes place in the individual domains, separated from each other. As a consequence, when it comes to finally integrating the different PSS components, it becomes obvious that the overall systems does not fit together seamlessly. At this stage, time to market is often the dominating factor and the PSS is literally “taped” together, forming a dirty solution attempt, rather than an integrated solution that is thoughtfully tailored towards customer needs.

With our approach we try to bring a certain degree of formalism into the development of PSS, thus justifying the term “PSS engineering”. Using model-based approaches to specify and document PSS engineering artifacts, it is possible to detect conflicts between different solution components of a PSS much easier and often earlier in the development process. However, our approach for model integration together with our prototypical software tool TRAILS cannot relieve developers from making deliberate decisions based on their know-how and experience. Our approach can rather be considered as an instrument that supports the human engineer in performing analysis and to get a general overview of the PSS in order to make decisions on complex engineering issues.

The approach presented in this thesis bridges the gap between the high level of details that is usually found in domain-specific modeling approaches and the rather generic descriptions of customer needs, solution components and business models which is inherent to most modeling approaches found in PSS literature.

From our perspective on PSS engineering (c.f. Figure 5), hardware and software components form the fundament for the service delivered by the PSS and the overall business model. Therefore, all components of the PSS, namely business model, service processes, software and hardware components need to be carefully aligned and harmonized in order to mutually highlight each other's strengths and cancel out deficits. However, in literature on PSS we have often found that many PSS researchers exaggerate the role of service in PSS so much, that one could even think other parts were not necessary. With this work, we therefore want to deliver the message that all domains: hardware, software and service should be viewed as equally important.

With regard to PSS engineering researchers often highlight the lack of conceptual models that describe the problem domain (Becker et al. 2010). However, with the far-ranging landscape of domain-specific modeling approaches that are already employed by the various engineering domains, it is in our eyes contra-productive to create a self-contained and independent conceptual model for the PSS domain as a whole. With our approach we are among the first to present an integrative conceptual model that builds upon existing domain-specific modeling approaches, rather than presenting yet another perspective on the environment of PSS engineering that is built up from scratch.

1.3 Implications for Practice

Using our approach, engineers can use their domain-specific modeling tools and do not have to care about integrateability of the specifications they produce. In practice we often realize that the dependencies between individual engineering artefacts need to be determined and documented manually, at least to some degree. Hence, establishing traceability entails an amount of manual effort that is not to be neglected. Especially in large engineering projects documenting artifacts, trace links and their evolution produces a tremendous amount of data. In this large data pool, inaccurate, obsolete or inconsistent information likely get lost in the shuffle (Ramesh & Edwards, 1993). In traditional service engineering for example, requirements engineering is often seen as irrelevant, since service are believed to need to evolve over time. However, we believe the requirements engineering for services and especially traceability between the requirements and the service process help service providers to train employees, monitor service reception by the customer or reuse certain elements when designing new services.

The manual generation of trace links is often subjective and consequentially error-prone. Also, trace links and associated traceability information are subject to heterogeneous granularity and relevance. This means that even if in general information is documented, answering specific questions might still be impossible due to data gaps and variations in the data quality (Gotel & Finkelstein, 1994). Moreover, the manual effort for establishing traceability, as discussed by Lee et al. (2003), is one of the major reasons why traceability is

not implemented in practice. In many cases, the various stakeholders are not motivated to document any traceability information as they neither see themselves as the ones who profit from this information for their work, nor is it part of their usual job description.

The consistency of the trace link network also depends on frequent updates whenever changes occur. If traceability information is not kept up to date reliably, managers make misinformed decisions or engineers implement flawed designs (Lee et al., 2003).

As comprehensive traceability also involves documenting which stakeholder is accountable for a certain artefact or an activity within the development process of the PSS, this information can be misused as an input for employee's performance evaluations. In practice engineers see traceability often critical as they fear to be under constant performance pressure (Ghazarian, 2008).

When setting the traceability strategy, one has to decide on the level of granularity at which traceability information is being captured and trace links are being recorded. Too coarse grained traceability misses out on important details while capturing information on a rather fine grained level makes the effort surpass the value added (Ramesh & Jarke, 2001).

The human interpretation of trace links can lead to heterogeneous results as each stakeholder might interpret the semantics of a particular trace link differently (Ramesh & Jarke, 2001). The varying perspectives often need to be aligned by direct communication among the stakeholders. Yet again, knowledge that is exchanged or created during these conversations is often not captured explicitly in the traceability database thus hiding important information from others to whom it might be relevant as well (Gotel & Finkelstein, 1994).

Overall, we believe that our approach for ensuring traceability can be used in various areas of engineering. Thus, the specialization on requirements traceability should only be seen as a major use case for this dissertation.

1.4 Limitations

Although the previous sections have shown the many advantages that the suggested approach in this thesis offers for the engineering and managing the lifecycle of PSS, there are also some issues connected to establishing traceability in general and using this approach in particular. These issues are discussed in the following.

General limitations regarding the research approach

As presented in part A section 3, the research presented in this thesis was performed following the design science research (DSR) paradigm. Is special characteristic of DSR is its focus on creating, evaluating and enhancing artifacts. Moreover, DSR encourages the researcher in repeating these activities iteratively, resulting in an ever better design of the artifacts under consideration after each cycle. As a consequence, this research strategy avoids getting lost in a never ending problem analysis phase and along with that procrastinating the development of an initial prototypical artifact design. However, it also makes it hard to strike new paths except when finding oneself trapped in a dead end. This way, our research aimed

more at building a functional model integration ontology along with a working prototype of our software tool TRAILS, rather than trying to find the optimal structure of the ontology or the implementing the tool with the optimal technology and runtime execution efficiency.

The evaluation of our model integration approach along with our software tool TRAILS is largely based on a single case study, namely the development of a bike sharing systems (c.f. Publications 5, 7 and 8). By doing so, it was possible to perform an in-depth evaluation of our concepts and tool for this particular PSS. However, the informative value of the evaluation regarding the various industries in which PSS business models are applicable, would undoubtedly be higher if multiple case studies from different areas could have been analyzed. Also, although in the course of developing our model integration ontology, we conducted qualitative interviews with experts from different engineering domains (c.f. Publication 7) and project managers from various industries, we could not interview a company that sees itself as a pure PSS provider. However, most of the companies interviewed combined at least two engineering domain (e.g. mechanical and software engineering) in their product development processes.

Furthermore, although the consideration of many different aspects from theory as well as from practice brings many advantages, it also brings in conflicting requirements and views regarding solution development. Hence, it is necessary to set a clear and limited scope of which engineering domains, fields of research and potential use cases of the aspired IT artifact are at the focus. Although the approach and tool presented in this thesis can find their application in many areas of engineering, during their development we explicitly focused on PSS and the engineering challenges associated with them. This means, that special characteristics of a particular engineering domain were not considered in detail if they had no significant impact on the overall PSS and the integration of its components.

Limitations regarding the model integration ontology

As argued before, the focus of our ontology rather lies on the integration of existing engineering artifacts that are documented using domain-specific modeling languages and data formats than on specifying or modeling these artifacts in the first place. First and foremost the resulting ontology acts as a meta-model to enable model transformation and integration. In this context we explicitly do not aim at capturing the PSS and the corresponding engineering process in every detail. Consequentially, the specification of details is something we deliberately leave to domain-specific modeling approaches and artifacts.

Although during the development of the model integration ontology we analyzed existing ontologies in various application areas, our primary goal was not interoperability or conformity with these ontologies, but rather compatibility with domain-specific modeling approaches. Hence, the structure of our ontology was derived from what we identified as the common conceptual core of different domain-specific modeling approaches instead of building the ontology from scratch based on the definition of concepts in each engineering domain that can be found in respective scientific literature.

Within the scope of this thesis we focused on a limited number of domain-specific modeling approaches. For modeling the system environment we considered just one approach, namely e3-value modeling. In the area of modeling the system results we focused on Requirements Diagrams (SysML), I*, Use Case Diagrams (UML, SysML) and Function Trees. Regarding the system behavior, we particularly took BPMN, EPC, Activity Diagrams (UML, SysML), Sequence Diagrams (UML, SysML), Service Blueprints and Petri Nets into consideration. And for modeling of the system structure our analysis most notably involved Block Diagrams (SysML), Class Diagrams (UML) and Design Structure Matrices. Other modeling approaches were only marginally considered without spending much time on any in-depth analysis. This way, it is very likely that the integration ontology may need to be extended or adapted if certain specifics of other modeling approaches are to be incorporated in the future.

Limitations regarding the prototypical software tool

In general TRAILS should not be seen as a software tool that is ready to be rolled out to the market but rather as a academic prototype for research purposes. In fact, our goal with TRAILS is to show the general feasibility of a tool that works according to the concepts and principles described in this thesis regarding traceability and model integration.

Following the Design Science Research basic guidelines, our primary objective when developing of our software tool TRAILS, was to present a working prototype rather than search for the ideal solution. For this reason, we decided for a modular software architecture that allows flexibly adding or replacing the implementation of software features by separating the model management core functions from e.g. the presentation layer or the synchronization with the central graph database. Consequentially, while iterating through the cycles of DSR, we evaluated the suitability of certain technologies and revealed obstacles along this way.

In fact, the basket of technologies that are utilized by the tool evolved over time as during our research we learned that many of the challenges we faced were similar to what the semantic web community is facing (a research area we originally had not planned to consider). For example, the serialization format that was originally used in the course of model integration was GraphML before we switched to RDF and OWL as they are more flexible, more popular and allow us to make use of further features of semantic web technologies, such as logical reasoning.

Although TRAILS is capable of importing some standard data formats, such as ReqIF or XMI that are being used several tools, the model import feature has only been tested with a few major tools and only with the current version available. Thus, import from other tools might still cause some issues that need to be fixed. Furthermore, in the current version of TRAILS all mapping rules between domain-specific models or data formats respectively are hard-coded. However, in order to improve compatibility with third-party software (especially with tools that have not been tested yet in connection with TRAILS), we believe that a model mapping engine that can be flexibly adjusted at runtime would constitute a great improvement.

During evaluation of the model import features of TRAILS we created the various models in the context of our bike sharing system case using popular commercial-off-the-shelf software tools, most of which are industry standard (e.g. MagicDraw for UML/SysML modeling or Microsoft Visio for EPCs). We tested TRAILS using this academic case study of developing a bike sharing system. Although we are certain, that this case study is a realistic PSS example, we have to admit that a further evaluation using more extensive models and also a broader scope of models from different engineering domains and even case studies from different industries would be desirable. Until now, the performance of TRAILS in large engineering projects has yet not been tested and we expect that some improvements regarding the computing resource consumption have to be carried out in order to deliver acceptable response times and with it a smooth user experience.

1.5 Future Research

Reflecting on our results we want to highlight three major starting points for future research to advance from. A first promising starting point is the **(1) extension of the integration ontology** in order to cover additional artifact types and augment its applicability for additional use cases. Second we think that there is great potential in the **(2) enhancement of TRAILS**, our prototypical traceability and model integration software tool by adding additional features and improving the existing ones. Finally, the third point, as already mentioned in the limitations section, is a detailed **(3) empirical evaluation** of our results in terms of their performance in real industry cases. Following the structure of the subsections before, we explain each of these three starting points subsequently:

Extension of the Integration Ontology

In its current state, the integration ontology considers aspects from all life cycle phases of a PSS, but all in all, the clear focus is the development stage of the PSS components. For some use cases, such as analyzing the manufacturability or estimation of service delivery costs however, information from other PSS life cycle management domains is required. This comprises for example the production of the physical components, deployment of the software or maintenance related information during the service provision phase. In order to reflect more details from these life cycle phases of a PSS, the integration ontology might therefore need to be extended. In this context it also seems desirable to consider further modeling approaches and types of specification documents.

Looking back to the expert interviews and case studies we performed in different companies and industries in the context of requirements traceability we observe that modeling approaches along with the artifacts used do not only vary between different engineering domains but also between different industries and even companies. Bearing this fact in mind, we can easily realize that considering all modeling approaches and artifacts currently existing on the planet within the integration ontology is a Sisyphean challenge that could never be performed by an individual. Moreover, it would contradict the original intention behind the integration ontology, namely reducing the level of detail and therefore the complexity of artifacts for the purpose of providing a comprehensive overview of the system. So, instead of extending the expressive power of the model integration ontology to the infinite, a better

strategy is to define fixed extension points and position the model integration ontology under the umbrella of an upper ontology, thus making it compatible with other ontologies. This would open up whole new use cases for the integration ontology.

Publication 7 argues how our integration ontology can be used as a reference model for traceability among engineering artifacts. However, we think that it can be turned into much more, namely a reference model of PSS in general. So far, our integration ontology defines for example a hierarchical decomposition of requirements into different levels (Business Goals, System Requirements, Design Requirements and Domain Requirements). Taking this decomposition some steps further, this could result in a comprehensive library of requirements types that are either relevant or not for certain types of PSS (or PSS components, respectively). For example, the requirement type “IT-security requirement” would generally be relevant for software components whereas the requirement type “safety requirement” would be relevant for hardware components only. This way, PSS engineers would be provided with a template for the requirements specification that is to be enriched with content at an instance level.

Taking this idea even further, one wouldn't need to stop with the requirements specification. When looking at different existing PSS we couldn't miss to recognize that some types of value propositions, service processes or even types of generic software or hardware components were very similar from one PSS to another and they kept repeating. Making use of this fact, our vision is a PSS construction kit that offers templates for PSS components. E.g. a PSS that involves renting out physical goods to customers will mostly likely require a maintenance process. This maintenance process again would most likely require maintenance staff. For maintenance staff there would be some requirements regarding worker safety, privacy laws, working hours, training requirements and so on.

A result would be some kind of PSS library containing generic best practice examples for certain types of PSS. This in turn would enable the definition of reference architectures for certain PSS components or PSS types as a whole by defining templates and building blocks for PSS development based on existing PSS artifacts. Based on such a PSS design catalogue, engineers could be to some degree guided through the development process by best practice templates. A PSS provider would then just have to work out details and adapt those processes to his environment. As an analogy for this application scenario, we can look at larger vendors of enterprise resource planning software. In order to create a software system that can serve in various industries, the developers of enterprise resource planning tools took best practice examples of business processes that could serve as a reference for all sorts of enterprises as a blueprint for their tool's workflows. As these enterprise resource planning tools matured, more and more companies adapted to the business process standard as defined by the tools thus strengthening their reference character.

In order to provide a fundamental collection of PSS best practice artifacts, we propose conducting case studies of existing PSS in various industries that have gained some market acceptance. Based on such case studies one would be able to identify similar PSS components

and accordingly specify generalized engineering artifacts that can be re-used in other PSS engineering projects.

Enhancement of TRAILS

Although our prototypical software tool TRAILS already covers the in our eyes most fundamental features for the application area it was designed for, namely traceability and model integration, we think that it is worth to continue its further advancement through thorough evaluation, the enhancement of existing features as well as the development of new features.

For example, the import of models from third-party software tools occasionally produces errors when special features of the external tool are used or if the data formats used are proprietary. TRAILS would thus benefit from a smart and robust import mechanism that is able to import even files that do not conform to standard data exchange formats, such as ReqIF or XMI. Such a mechanism would then present the imported and interpreted information (e.g. as unstructured plain text) to the user and provide functions that allow the user to turn this information into the desired format by defining the mappings between element (entities, relationships and attributes) identified in the source format and elements of the target format used by TRAILS. Ideally this would work using an intuitive graphical user interfaces that allows such mapping definitions via drag-and-drop combination of atomic transformation operators into a transformation sequence. This transformation sequence would need to be defined once for each type of element in the source format and a import rule engine would then learn from such alterations and deliver a better import result for the future automatically. By implementing such a feature in TRAILS we believe that domain experts could easily add new modelling languages or data formats easily to TRAILS without having to touch the software code by just configuring the transformation process.

Another aspect that would need to be regarded when further improving TRAILS is collaboration. As discussed before, PSS engineering is a task that involves stakeholders from various domains and often a larger team that is directly concerned with development and service provision. As always, the productive collaboration of a larger engineering team requires intensive coordination, communication and cooperation, especially if the team is distributed over different locations or even organizations (Fuks et al. 2007). As a consequence collaborative engineering features are critical for an engineering tool like TRAILS. Although it is already possible with TRAILS to store the data in a central graph data base using a client-server architecture and retrieve it from there, larger models and concurrent access and manipulation of the integrated model call for more sophisticated features regarding secure user and access management, revision controls as well as anticipatory synchronization of locally-stored and server-side data, so that a user only works on a currently required sub-graph of the comprehensive PSS model.

In this context, with many people working on the same data, it is also desirable to be able to automatically detect, manage and dissolve inconsistencies in the model. Apart from conflicting versions of the same element inconsistencies can also manifest themselves as violations of fundamental physical or logical laws. This can be for example, a deadlock in a

work-flow because of cyclicity, the self-containment of components or an incorrect conversion of measurement units (e.g. between the metric and the US customary system). Furthermore, with regard to requirements engineering there are often mismatches between a requirement and the solution artifact that is supposed to fulfil this requirement or contradicting requirements that refer to the same solution artifacts. Also an requirement that is not fulfilled by any solution artifacts or a solution artefact that does not fulfil any requirement may be viewed as an inconsistency in the comprehensive PSS model. We thus aspire to augment TRAILS with an inconsistency management feature that allows to automatically detect inconsistencies in the model through logical reasoning and continuously checking the model against pre-defined rules, ranging from basic laws of physics to generally accepted good engineering practices.

In our vision, TRAILS will evolve to something we call “Semantic Engineering Network”. Similar to what for example Facebook does with people; it connects PSS solution components and other engineering artifacts. One can review the history (i.e. the evolution) of an artifact like one can review somebody’s wall on social media. Moreover, it is possible to query to semantic engineering graph like the social graph on Facebook in order to find answers to specific questions (including finding inconsistencies in the model). For this purpose, the architecture of TRAILS is already based on RDF. As a next step, one can now make use of other semantic web technologies, such as SPARQL, OWL or RIF in order to equip TRAILS with advanced semantic search, logical reasoning and rule checking functions.

Empirical Evaluation

As stated before, one limitation of the research presented in this dissertation is that due to the research strategy chosen, we focused on the development and iterative improvement of our model integration ontology and our software tool TRAILS. In this process we evaluated our results against the recommendations found in literature on PSS, requirements traceability and model-based engineering on the one hand and conducted expert interviews as well as a PSS engineering case study (developing a bike sharing system) on the other hand. At this stage, where we have a working prototype of TRAILS and the proof of concept of the model integration ontology completed, we believe it is time to take the evaluation to the next level.

In order to demonstrate the applicability of our approach, both, the model integration ontology and TRAILS should be evaluated using real industry use cases. As a first step in this direction we propose to test model transformation and integration using data from model-based engineering projects in multiple industries. These do not necessarily have to be PSS engineering projects since the focus should lie on testing whether TRAILS is able to work with different sets of real industry data and determine its performance with more extensive models. Subsequently, TRAILS should of course be also tested with data from PSS engineering projects or at least engineering projects that involve a multitude of different engineering domains.

In addition to merely evaluating TRAILS and the model integration ontology by feeding them with industry data, it is also advisable to demonstrate their application to industry experts, analyze their feedback and observe them working with the tool in order to improve usability.

Due to the complexity of the application area and the software landscape in engineering, it might at this stage be too soon to conduct piloting studies within a real engineering project. Instead we think that expert evaluation of our approach could be initially done in modeling workshops in a controlled setting. Here, each expert would receive a detailed introduction to TRAILS and then would be asked to perform specific tasks using the tool, such as finding certain piece of information within the model or importing and integrating multiple domain-specific models. Afterwards during an interview or focus group session, expert could share their experiences working with the tool, argue on advantages and disadvantages of TRAILS when compared to other engineering tools of the application area and illustrate potentials of further improvement.

2 Conclusion

Iterating through the cycles of design science research, our goal was to create a novel approach for ensuring traceability throughout the entire life cycle of a PSS. For this purpose, we analyzed the concept of traceability in PSS engineering from multiple perspectives, in theory as well as in practice. Our research has started with an analysis of the special characteristics of PSS engineering and their implications on traceability along the entire life cycle of a PSS. In this study we identified nine major characteristics that differentiate the development of PSS from traditional engineering. Based on this result, we determined whether existing traceability approaches were suitable for PSS engineering. We came to the conclusion, that out of the existing approaches we analyzed, none was without restrictions recommendable for PSS but each approach was targeted at solving a specific challenge associated with traceability and if combined and enhanced they could together serve as a valuable starting basis for the development of a traceability approach for PSS. To approach traceability from a practical viewpoint, we furthermore conducted multiple case studies involving experts from different industries and engineering domains.

From our analyses we concluded, that the most important prerequisite for traceability in a cross-disciplinary setting, such as the development of PSS, is being able to integrate all information that is required in order to ensure traceability in a common representation that allows for an evaluation of the artifacts and their semantic relationships. Consequently, we introduced our PSS integration ontology which allows joining artifacts from the various domains involved in PSS engineering as well as an approach to automatically transform the traceability information contained in domain-specific models into a generic format. On the basis of all preceding results, we specified a traceability reference model that is explicitly tailored towards PSS engineering. Altogether, our model integration ontology for domain-specific PSS models form an optimal fundament for the development of software tools that appropriately support traceability in the life cycle of a PSS.

Our prototypical software tool TRAILS allows to integrate graph-based models and other types of specification artifacts that are commonly used in the different engineering domains involved in the development of a typical PSS. We evaluated TRAILS together with the underlying ontology-based approach for model integration in a case study that is concerned with the development of a free-floating bike sharing system, where users can search for available bikes using a smartphone app, rent them out and return them at any desired location.

Although we have to admit, that our research is subject to some limitations due to the research strategy that was chosen as well as regarding the design of our model integration ontology and the prototypical implementation of our software tool TRAILS, we are certain that the results presented in this dissertation provide valuable input for others to advance from here and build upon our results. Taking a look at possible future research we especially want to highlight three starting points that in our eyes seem promising, namely extension of our model integration ontology, enhancement of TRAILS as well as an empirical evaluation of our results during industry case studies.

References

- Adzic, G. (2011): *Specification by Example: How Successful Teams Deliver the Right Software*. 1st ed.: Manning.
- Ahn, S.; Chong, K. (2006): A Feature-oriented Requirements Tracing Method. A Study of Cost-Benefit Analysis. In: *International Conference on Hybrid Information Technology*. Washington DC, pp. 611–616.
- AIS, Institute for Automation and Information Systems (2014): The Pick and Place Unit. Available online at <http://www.ais.mw.tum.de/forschung/demonstratoren/ppu/>, checked on 12/1/2016.
- Aizenbud-Reshef, N.; Nolan, B. T.; Rubin, J.; Shaham-Gafni, Y. (2006): Model traceability. In *IBM Systems Journal* 45 (3), pp. 515–526.
- Akao, Yōji (1990): *Quality function deployment. Integrating customer requirements into product design*. Cambridge, Mass: Productivity Press.
- Alfian, Ganjar; Rhee, Jongtae; Yoon, Byungun (2014): A simulation tool for prioritizing product-service system (PSS) models in a carsharing service. In *Computers and Industrial Engineering* 70, pp. 59–73.
- Alonso-Rasgado, Teresa; Thompson, Graham; Elfström, Bengt-Olof (2004): The design of functional (total care) products. In *Journal of Engineering Design* 15 (6), pp. 515–540.
- Anacker, Harald; Dorociak, Rafal; Dumitrescu, Roman; Gausemeier, Jürgen (2011): Integrated tool-based approach for the conceptual design of advanced mechatronic systems. In: *Systems Conference (SysCon), 2011 IEEE International*. IEEE, pp. 506–511.
- Anderl, Reiner; Eigner, Martin; Sandler, Ulrich; Stark, Rainer (2012): *Smart engineering: interdisziplinäre Produktentstehung*: Springer-Verlag.
- Andersson, Fredrik; Sutinen, Krister; Malmqvist, Johan (2003): Product Model for Requirements and Design Concept Management: Representing Design Alternatives and Rationale. In: *3th Annual International Symposium INCOSE 2003*.
- Antoniol, G.; Canfora, G.; Casazza, G.; Lucia, A. de; Merlo, E. (2002): Recovering traceability links between code and documentation. In *IEEE Transactions on Software Engineering* 10 (28), pp. 970–983.
- Arkley, P.; Riddle, S. (2006): Tailoring Traceability Information to Business Needs. In IEEE (Ed.): *Proceedings of the 14th IEEE International Requirements Engineering Conference. International Requirements Engineering Conference*. Minneapolis/St. Paul, MN, 11.-15. September 2006, pp. 239–244.
- Aurich, Jan C.; Fuchs, Christian; Wagenknecht, Christian (2006): Life cycle oriented design of technical Product-Service Systems. In *Journal of Cleaner Production* 14 (17), pp. 1480–1494.
- Baines, T. S.; Lightfoot, H. W.; Evans, S.; Neely, A.; Greenough, R.; Peppard, J. et al. (2007): State-of-the-art in product-service systems. In *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 221 (10), pp. 1543–1552. DOI: 10.1243/09544054JEM858.
- Barbieri, Giacomo; Kernschmidt, Konstantin; Fantuzzi, Cesare; Vogel-Heuser, Birgit (2014): A SysML based design pattern for the high-level development of mechatronic systems to enhance re-usability. In: *IFAC World Congress*, pp. 3431–3437.

- Bartolomeo, Matteo; dal Maso, Davide; Jong, Paulien de; Eder, Peter; Groenewegen, Peter; Hopkinson, Peter et al. (2003): Eco-efficient producer services—what are they, how do they benefit customers and the environment and how likely are they to develop and be extensively utilised? In *Journal of Cleaner Production* 11 (8), pp. 829–837.
- Baskerville, Richard L. (1997): Distinguishing action research from participative case studies. In *Journal of systems and information technology* 1 (1), pp. 24–43.
- Bauer, S.; Paetzold, K. (2006): Influence of DfX criteria on the design of the product development process. In S. Vajna (Ed.): Proceedings of the 6th Workshop on Integrated Product Development. 6th Workshop on Integrated Product Development. Magdeburg, 18.-20.09.2006.
- Baxter, David; Roy, Rajkumar; Doultsinou, Athanasia; Gao, James; Kalta, Mohamad (2009): A knowledge management framework to support product-service systems design. In *International journal of computer integrated manufacturing* 22 (12), pp. 1073–1088.
- Becker, Dipl-Inf Michael; Klingner, Dipl-Inf FH Stephan (2013): Formale Modellierung von Komponenten und Abhängigkeiten zur Konfiguration von Product-Service Systems. In: Dienstleistungsmodellierung 2012: Springer, pp. 114–140.
- Becker, Jörg; Beverungen, Daniel; Knackstedt, Ralf; Glauner, Christoph; Stypmann, Marco; Rosenkranz, Christoph et al. (2009): Ordnungsrahmen für die hybride Wertschöpfung. In: Dienstleistungsmodellierung: Springer, pp. 109–128.
- Becker, Jörg; Beverungen, Daniel F.; Knackstedt, Ralf (2010): The challenge of conceptual modeling for product-service systems: status-quo and perspectives for reference models and modeling languages. In *Information Systems and e-Business Management* 8 (1), pp. 33–66.
- Becker, Jörg; Krcmar, Helmut (2008): Integration von Produktion und Dienstleistung. Hybride Wertschöpfung. In *Wirtschaftsinformatik* 50 (3), pp. 169–171.
- Becker, Jörg; Pfeiffer, Daniel (2008): Solving the Conflicts of Distributed Process Modelling: Towards an Integrated Approach. In: ECIS, pp. 1555–1568.
- Berente, Nicholas; Lyytinen, Kalle (2007): What is being iterated? Reflections on iteration in information system engineering processes. In *Conceptual Modelling in Information Systems Engineering*, pp. 261–278.
- Bergenthal, Jeff (2011): Final Report of the Model Based Engineering (MBE) Subcommittee. Available online at http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Committees/M_S%20Committee/2011/February/NDIA-SE-MS_2011-02-15_Bergenthal.pdf, checked on 6/1/2017.
- Berkovich, M.; Esch, S.; Leimeister, J. M.; Krcmar, H. (2010a): Towards Requirements Engineering for Software as a Service. In: Proceedings of Multikonferenz Wirtschaftsinformatik (MKWI 2010). Göttingen, Germany.
- Berkovich, M.; Leimeister, J. M.; Krcmar, H. (2010b): Ein Bezugsrahmen für Requirements Engineering hybrider Produkte. In: Proceedings of Multikonferenz Wirtschaftsinformatik (MKWI 2010). Göttingen, Germany.
- Berkovich, Marina; Esch, Sebastian; Leimeister, Jan Marco; Krcmar, Helmut (2009): Requirements engineering for hybrid products as bundles of hardware, software and service elements – a literature review. In: Proceedings of the 9th International Conference on Wirtschaftsinformatik.

- Berkovich, Marina; Esch, Sebastian; Mauro, Christian; Leimeister, Jan Marco; Krcmar, Helmut (2011a): Towards an Artifact Model for Requirements to IT-enabled Product Service Systems. In *Wirtschaftsinformatik* 2011, p. 16.
- Berkovich, Marina; Leimeister, Jan Marco; Hoffmann, Axel; Krcmar, Helmut (2012): A requirements data model for product service systems. In *Requirements Eng.* DOI: 10.1007/s00766-012-0164-1.
- Berkovich, Marina; Leimeister, Jan Marco; Krcmar, Helmut (2011b): Requirements Engineering for Product Service Systems - A State of the Art Analysis. In *Bus Inf Syst Eng* 3 (6), pp. 369–380. DOI: 10.1007/s12599-011-0192-2.
- Berkovich, Marina; Mauro, Christian; Leimeister, Jan Marco; Weyde, Felix; Krcmar, Helmut (2011c): Towards Cycle-Oriented Requirements Engineering. In Abraham Bernstein, Gerhard Schwabe (Eds.): Proceedings of the 10th International Conference on Wirtschaftsinformatik 2011, vol. 2. International Conference on Wirtschaftsinformatik. Zurich, Switzerland. Internationale Tagung Wirtschaftsinformatik. Raleigh: Lulu, pp. 963–973.
- Bernard, Y.; Burkhart, R. M.; Koning, H. P. de; Friedenthal, S.; Fritzson, P.; Paredis, C. et al. (2010): An Overview of the SysML-Modelica Transformation Specification. In: Proc. of INCOSE, pp. 11–15.
- Beuren, F. H.; Gomes, M. G.; Miguel, P. A. (2013): Product-Service Systems. A Literature Review on integrated Products and Services. In *Journal of Cleaner Production* 47, pp. 222–231.
- Beverungen, D.; Knackstedt, R.; Hatfield, S.; Biege, S.; Bollhöfer, E.; Krug, C. et al. (2009): Hybride Wertschöpfung - Integration von Produktion und Dienstleistung. Berlin: Beuth Verlag.
- Bézivin, Jean; Büttner, Fabian; Gogolla, Martin; Jouault, Frédéric; Kurtev, Ivan; Lindow, Arne (2006): Model transformations? transformation models! In: Model driven engineering languages and systems: Springer, pp. 440–453.
- Bochnig, Holger (2012): Assistenzsystem zur Ausgestaltung hybrider Leistungsbündel. In: Integrierte Industrielle Sach- und Dienstleistungen: Springer, pp. 89–111.
- Boehm, B. (2000): Requirements that handle IKIWISI, COTS, and rapid change. In *Computer* 33 (7), pp. 99–102.
- Boehm, M.; Thomas, O. (2013): Looking beyond the Rim of one's Teacup. A multidisciplinary Literature Review of Product-Service Systems in Information Systems, Business Management, and Engineering & Design. In *Journal of Cleaner Production* 51, pp. 245–260.
- Böhmman, Tilo; Krcmar, Helmut (2007): Hybride Produkte. Merkmale und Herausforderungen. In Manfred Bruhn, Bernd Stauss (Eds.): Wertschöpfungsprozesse bei Dienstleistungen: Forum Dienstleistungsmanagement: Gabler, pp. 240–255.
- Böhmman, Tilo; Langer, Philipp; Schermann, Michael (2008): Systematische Überführung von kundenspezifischen IT-Lösungen in integrierte Produkt-Dienstleistungsbausteine mit der SCORE-Methode. In *Wirtschaftsinformatik* 50 (3).
- Bonnemeier, Sebastian; Ihl, Christoph; Reichwald, Ralf (2007): Wertschaffung und Wertaneignung bei hybriden Produkten - Eine prozessorientierte Betrachtung. München (Arbeitsbericht, Nr. 03 / 2007).
- Borchers, J. O. (2000): A Pattern Approach to Interaction Design. New York: ACM.

- Botta, Christian (2007): Rahmenkonzept zur Entwicklung von Product-Service Systems. Product-Service Systems Engineering. Lohmar: Josef Eul Verlag.
- Bouillon, Elke; Güldali, Baris; Herrmann, Andrea; Keuler, Thorsten; Moldt, Daniel; Riebisch, Matthias (2013a): Leichtgewichtige Traceability im agilen Entwicklungsprozess am Beispiel von Scrum. In *Softwaretechnik-Trends* 33 (1), pp. 29–30.
- Bouillon, Elke; Mäder, Patrick; Philippow, Ilka (2013b): A survey on usage scenarios for requirements traceability in practice. In: Requirements Engineering: Foundation for Software Quality: Springer, pp. 158–173.
- Brambilla, Marco; Cabot, Jordi; Wimmer, Manuel (2012): Model-driven software engineering in practice. In *Synthesis Lectures on Software Engineering* 1 (1), pp. 1–182.
- Brickley, Dan; Miller, Libby (2014): FOAF Vocabulary Specification 0.99. Available online at <http://xmlns.com/foaf/spec/>, checked on 7/30/2017.
- Bullinger, H.-J; Fähnrich, K.-P; Meiren, T.: Service Engineering - Methodical Development of New Service Products. *Fraunhofer Institute of Industrial Engineering*.
- Bullinger, Hans-Jörg; Scheer, August-Wilhelm (Eds.) (2003): Service Engineering: Springer Berlin Heidelberg.
- Bullinger, Hans-Jörg; Schreiner, Peter (2006): Service engineering: Ein Rahmenkonzept für die systematische Entwicklung von Dienstleistungen. In: Service Engineering: Springer, pp. 53–84.
- Burianek, Ferdinand; Bonnemeier, Sebastian; Ihl, Christoph; Reichwald, Ralf (2009): Grundlegende Betrachtung hybrider Produkte. In *Hybride Wertschöpfung-Konzepte, Methoden und Kompetenzen für die Preis und Vertragsgestaltung*. Lohmar: EUL, pp. 13–31.
- Burianek, Ferdinand; Ihl, Christoph; Bonnemeier, Sebastian; Reichwald, Ralf (2007): Typologisierung hybrider Produkte. Ein Ansatz basierend auf der Komplexität der Leistungserbringung. Edited by Organisation und Management der Technischen Universität München Lehrstuhl für Betriebswirtschaftslehre – Information (Arbeitsbericht), checked on 4/5/2012.
- Cavaliere, Sergio; Pezzotta, Giuditta (2012): Product-Service Systems Engineering. State of the art and research challenges. In *Computers in Industry* 63 (4), pp. 278–288.
- Chandrasekaran, Balakrishnan; Josephson, John R.; Benjamins, V. Richard (1999): What are ontologies, and why do we need them? In *IEEE Intelligent Systems and their applications* 14 (1), pp. 20–26.
- Chen, David; Doumeingts, Guy; Vernadat, François (2008): Architectures for enterprise integration and interoperability. Past, present and future. In *Computers in Industry* 59 (7), pp. 647–659.
- Cheng, Betty H. C.; Atlee, Joanne M. (2007): Research Directions in Requirements Engineering. In IEEE (Ed.): Proceedings of the 15th IEEE International Requirements Engineering Conference. 15th IEEE International Requirements Engineering Conference.
- Chucholowski, N.; Wolfenstetter, T.; Wickel, M. C.; Krcmar, H.; Lindemann, U.; others (2014): Towards Cycle-oriented Traceability in Engineering Change Management. In *DS* 77, pp. 1491–1500.
- Cleland-Huang, J.; Settini, R.; BenKhadra, O.; Berezanskaya, E.; Christina, S. (2005): Goal-centric Traceability for Managing non-functional Requirements. In. 10th European

Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Lissabon, pp. 362–371.

Cleland-Huang, Jane (2005): Toward Improved Traceability of Non-Functional Requirements. In *TEFSE '05 Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, pp. 14–19.

Cleland-Huang, Jane; Chang, Carl K.; Sethi, Gaurav; Javvaji, Kumar; Hu, Haijian; Xia, Jinchun (2002): Automating speculative queries through event-based requirements traceability. In: *IEEE Joint International Conference on Requirements Engineering, 2002. Proceedings. IEEE*, pp. 289–296.

Cleland-Huang, Jane; David Schmelzer (2003): Dynamically Tracing Non-Functional Requirements through Design Pattern Invariants. In *Workshop on Traceability in Emerging Forms of Software Engineering, in conjunction with IEEE International Conference on Automated Software Engineering*.

Cleland-Huang, Jane; Gotel, Orlena C. Z.; Huffman Hayes, Jane; Mäder, Patrick; Zisman, Andrea (2014): Software traceability: trends and future directions. In: *Proceedings of the on Future of Software Engineering. ACM*, pp. 55–69.

Cockburn, Alistair; Highsmith, Jim (2001): Agile software development: The people factor. In *Computer* (11), pp. 131–133.

Cook, M. B.; Bhamra, T. A.; Lemon, M. (2006): The Transfer and Application of Product Service Systems. from Academia to UK Manufacturing Firms. In *Journal of Cleaner Production* 14 (17), pp. 1455–1465.

Cooper, Harris M. (1988): Organizing knowledge syntheses: A taxonomy of literature reviews. In *Knowledge in Society* 1 (1), pp. 104–126.

Creusen, Mariëlle E. H. (2011): Research Opportunities Related to Consumer Response to Product Design*. In *Journal of Product Innovation Management* 28 (3), pp. 405–408.

Curtis, Bill; Krasner, Herb; Iscoe, Neil (1988): A field study of the software design process for large systems. In *Communications of the ACM* 31 (11), pp. 1268–1287.

Cycorp (2016): Cyclic Development Platforms. Available online at <http://www.cyc.com/platform/>, checked on 7/30/2017.

Czarnecki, Krzysztof; Helsen, Simon (2003): Classification of model transformation approaches. In: *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, vol. 45. USA, pp. 1–17.

Czarnecki, Krzysztof; Helsen, Simon (2006): Feature-based survey of model transformation approaches. In *IBM Systems Journal* 45 (3), pp. 621–645.

Davies, Andrew; Brady, Tim; Hobday, Michael (2006a): Charting a path toward integrated solutions. In *MIT Sloan Management Review* 47 (3), p. 39.

Davies, John; Studer, Rudi; Warren, Paul (2006b): *Semantic Web technologies: trends and research in ontology-based systems*: John Wiley & Sons.

DCMI (2017): DCMI Specifications. Dublin Core Metadata Initiative. Available online at <http://dublincore.org/specifications/>, checked on 7/30/2017.

Diestel, Reinhard (2005): *Graduate texts in mathematics: Graph theory*. Berlin, Heidelberg: Springer.

- Doppler, Klaus; Lauterburg, Christoph (2008): Change management: den Unternehmenswandel gestalten: Campus Verlag.
- Dorfman, Merlin; Flynn, Richard F. (1984): Arts—an automated requirements traceability system. In *Journal of Systems and Software* 4 (1), pp. 63–74.
- Drath, Rainer; Lüder, Arndt; Peschke, Jörn; Hundt, Lorenz (2008): AutomationML—the glue for seamless automation engineering. In: IEEE International Conference on Emerging Technologies and Factory Automation, 2008. ETFA 2008, pp. 616–623.
- Dubois, Hubert; Peraldi-Frati, Marie-Agnes; Lakhali, Fadi (2010): A Model for Requirements Traceability in a Heterogeneous Model-Based Design Process: Application to Automotive Embedded Systems. In IEEE (Ed.): Proceedings of the 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS). Oxford, United Kingdom, 22.-26. März 2010, pp. 233–242.
- Durugbo, Christopher (2011): Collaborative networks for product-service systems delivery. In: Concurrent Enterprising (ICE), 2011 17th International Conference on. IEEE, pp. 1–8.
- Durugbo, Christopher (2013): Integrated product-service analysis using SysML requirement diagrams. In *Syst. Engin.* 16 (1), pp. 111–123. DOI: 10.1002/sys.21229.
- Ebert, C.; Man, J. de: Requirements Uncertainty. Influencing Factors and concrete Improvements. In: 27th International Conference on Software Engineering, pp. 553–560.
- Ebert, Christof (2008): Systematisches Requirements-Engineering und Management. Anforderungen ermitteln, spezifizieren, analysieren und verwalten. 2. Aufl. Heidelberg: dpunkt-Verlag.
- Ebert, Christof (2012): Systematisches Requirements Engineering. Anforderungen ermitteln, spezifizieren, analysieren und verwalten. 4., überarb. Aufl. Heidelberg: dpunkt.
- Ebert, Christof (2014): Systematisches Requirements Engineering: Anforderungen ermitteln, dokumentieren, analysieren und verwalten: dpunkt. verlag.
- Egyed, Alexander (2001): A scenario-driven approach to traceability. In IEEE Computer Society (Ed.): Proceedings of the 23rd international conference on software engineering: IEEE Computer Society, pp. 123–132.
- Egyed, Alexander; Biffel, Stefan; Heindl, Matthias; Grünbacher, Paul (2005): A value-based approach for understanding cost-benefit trade-offs during automated software traceability. In: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering. ACM, pp. 2–7.
- Egyed, Alexander; Grünbacher, Paul (2005): Supporting Software Understanding with Automated Requirements Traceability. In *International Journal of Software Engineering & Knowledge Engineering* 15 (5), pp. 783–810.
- Ehrlenspiel, K. (2009): Integrierte Produktentwicklung: Denkabläufe, Methodeneinsatz, Zusammenarbeit: Hanser.
- Eigner, Martin (2012): Interdisziplinäre Produktentwicklung-Modellbasiertes Systems Engineering-korrigiert.
- Eisenbart, Boris; Blessing, Lucienne; Gericke, Kilian; others (2012): Functional modelling perspectives across disciplines: a literature review. In: Proceedings of 12th international design conference, design.

- Eisenbart, Boris; Qureshi, Ahmed; Gericke, Kilian; Blessing, Luciënne (2013): Integrating different functional modeling perspectives. In: ICoRD'13: Springer, pp. 85–97.
- El Maraghy, Waguih; El Maraghy, Hoda; Tomiyama, Tetsuo; Monostori, Laszlo (2012): Complexity in engineering design and manufacturing. In *CIRP Annals-Manufacturing Technology* 61 (2), pp. 793–814.
- Espinoza, Angelina; Garbajosa, Juan (2011): A study to support agile methods more effectively through traceability. In *Innovations in Systems and Software Engineering* 7 (1), pp. 53–69.
- Eversheim, Walter; Kuster, Johannes; Liestmann, Volker (2006): Anwendungspotenziale ingenieurwissenschaftlicher Methoden für das Service Engineering. In H.-J Bullinger, A.-W Scheer (Eds.): *Service Engineering: Entwicklung und Gestaltung innovativer Dienstleistungen*, vol. 3. 3rd ed. Berlin: Springer, pp. 418–441.
- Feiler, Peter H.; Gluch, David P. (2012): *Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language*: Addison-Wesley.
- Fernández, Daniel Méndez; Penzenstadler, Birgit; Kuhrmann, Marco; Broy, Manfred (2010): A meta model for artefact-orientation: fundamentals and lessons learned in requirements engineering. In: *Model driven engineering languages and systems*: Springer, pp. 183–197.
- Fernández, Daniel Méndez; Wieringa, Roel (2013): Improving requirements engineering by artefact orientation. In: *Product-Focused Software Process Improvement*: Springer, pp. 108–122.
- Fernández-López, Mariano; Gómez-Pérez, Asunción; Juristo, Natalia (1997): *Methontology. From ontological art towards ontological engineering*.
- Ferrarini, L.; Mantovani, G.; Allevi, M.; Dede, A. (2011): An integrated approach for specification and design of automated production systems. In: *System Integration (SII), 2011 IEEE/SICE International Symposium on*. IEEE, pp. 1406–1411.
- Fettke, Dipl.-Wirt.-Inf Peter (2006): State-of-the-Art des State-of-the-Art. In *Wirtschaftsinformatik* 48 (4), pp. 257–266.
- Floerecke, S.; Herzfeldt, A.; Krcmar, H. (2012): Risiken bei IT-Lösungen – Ein Risikokatalog für Praktiker aus Anbietersicht. In *IM – Die Fachzeitschrift für Informationsmanagement und Consulting* 27 (4), pp. 22–30.
- Floerecke, Sebastian; Wolfenstetter, Thomas; Krcmar, Helmut (2015): Hybride Produkte - Stand der Literatur und Umsetzung in der Praxis. In *IM+IO - Die Fachzeitschrift für Innovation, Organisation und Management* 30 (2), pp. 61–66.
- Foote, Nathaniel W.; Galbraith, Jay; Hope, Quentin; Miller, Danny (2001): Making solutions the answer. In *The McKinsey Quarterly*, p. 84.
- Fowler, Martin; Foemmel, Matthew (2006): Continuous integration. In *Thought-Works* [http://www.thoughtworks.com/Continuous Integration.pdf](http://www.thoughtworks.com/Continuous%20Integration.pdf), p. 122.
- France, Robert; Rumpe, Bernhard (2007): Model-driven development of complex software: A research roadmap. In: *2007 Future of Software Engineering*. IEEE Computer Society, pp. 37–54.
- Frank, Moti; Harel, Amir; Orion, Uzi (2014): Choosing the Appropriate Integration Approach in Systems Projects. In *Systems Engineering* 17 (2), pp. 213–224.

- Frese, Erich; Lehnen, Marc; Valcárcel, Sylvia (1998): Dienstleistungswettbewerb und regionale Reichweite. In Hans-Jörg Bullinger, Erich Zahn (Eds.): Dienstleistungs Offensive. Wachstumschancen intelligent nutzen. Stuttgart: Schäffer-Poeschel, pp. 35–64.
- Fricke, Ernst; Gebhard, Bernd; Negele, Herbert; Igenbergs, Eduard (2000): Coping with changes: causes, findings, and strategies. In *Systems Engineering* 3 (4), pp. 169–179.
- Fritzsche, Peter (2007): Innovationsmanagement für Dienstleistungen durch Service Engineering. Bedeutung und Ablauf der systematischen Dienstleistungsentwicklung. Saarbrücken: VDM, Müller.
- Fuks, Hugo; Raposo, Alberto; Gerosa, Marco A.; Pimentel, Mariano; Lucena, Carlos J. P. (2007): The 3c collaboration model. In *The Encyclopedia of E-Collaboration, Ned Kock (org)*, pp. 637–644.
- Galbraith, Jay R. (2002): Organizing to deliver solutions. In *organizational Dynamics* 31 (2), pp. 194–207.
- Garetti, Marco; Rosa, Paolo; Terzi, Sergio (2012): Life cycle simulation for the design of product-service systems. In *Computers in Industry* 63 (4), pp. 361–369.
- Gausemeier, Jürgen; Dorociak, Rafał; Kaiser, Lydia (2010): Computer-aided modeling of the principle solution of mechatronic systems: A domain-spanning methodology for the conceptual design of mechatronic systems. In: ASME 2010 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. American Society of Mechanical Engineers, pp. 1109–1118.
- Gebauer, Heiko; Paiola, Marco; Saccani, Nicola (2013): Characterizing service networks for moving from products to solutions. In *Industrial Marketing Management* 42 (1), pp. 31–46.
- Geisberger, Eva (2005): Requirements Engineering eingebetteter Systeme. Ein interdisziplinärer Modellierungsansatz. Aachen: Shaker (Berichte aus der Softwaretechnik).
- Geng, Xiuli; Chu, Xuening (2011): A new PSS conceptual design approach driven by user task model. In: *Functional Thinking for Value Creation*: Springer, pp. 123–128.
- Gero, John S. (1990): Design prototypes: a knowledge representation schema for design. In *AI magazine* 11 (4), p. 26.
- Geum, Youngjung; Park, Yongtae (2011): Designing the sustainable product-service integration: a product-service blueprint approach. In *Journal of Cleaner Production* 19 (14), pp. 1601–1614.
- Ghazarian, Arbi (2008): Traceability patterns: an approach to requirement-component traceability in agile software development. In: Proceedings of the 8th conference on Applied computer science. World Scientific and Engineering Academy and Society (WSEAS), pp. 236–241.
- Gläser, Jochen; Laudel, Grit (2010): Experteninterviews und qualitative Inhaltsanalyse: Springer-Verlag.
- Goedkoop, M.; van Halen, C.; Riele, Te H.; Rommens, P. (1999): Product Service Systems, Ecological and Economic Basics. Arbeitspapier. Dutch Ministries of Environment (VROM) and Economic Affairs (EZ). Niederlande.
- Gordijn, Jaap; Akkermans, J. M. (2003): Value-based requirements engineering: exploring innovative e-commerce ideas. In *Requirements engineering* 8 (2), pp. 114–134.

- Gotel, Orlena; Cleland-Huang, Jane; Hayes, J. Huffman; Zisman, Andrea; Egyed, Alexander; Grünbacher, Paul; Antoniol, Giuliano (2012): The quest for Ubiquity: A roadmap for software and systems traceability research. In: Requirements Engineering Conference (RE), 2012 20th IEEE International. IEEE, pp. 71–80.
- Gotel, Orlena; Finkelstein, Anthony (1994): An Analysis of the Requirements Traceability Problem. In IEEE Computer Society (Ed.): Proceedings of the First International Conference on Requirements Engineering. First International Conference on Requirements Engineering. Colorado Springs, pp. 94–101.
- Gotel, Orlena; Finkelstein, Anthony (1995): Contribution Structures. In IEEE Computer Society (Ed.): Proceedings of the Second IEEE International Symposium on Requirements Engineering. Second IEEE International Symposium on Requirements Engineering. York, UK, 27.-29. March. Los Alamitos, Calif: IEEE Computer Society Press, pp. 100–127.
- Grande, M. (2013): 100 Minuten für Konfigurationsmanagement – Kompaktes Wissen nicht nur für Projektleiter und Entwickler. 1. Auflage. Wiesbaden: Vieweg und Teubner Verlag.
- Gräßle, B. Sc Marc; Thomas, Oliver; Dollmann, Dipl-Inform Thorsten (2010): Vorgehensmodelle des Product-Service Systems Engineering. In: Hybride Wertschöpfung: Springer, pp. 82–129.
- Gruninger, Michael; Fox, Mark S. (1994): An activity ontology for enterprise modelling. In *Submitted to AAAI-94, Dept. of Industrial Engineering, University of Toronto* 321.
- Guarino, Nicola (1998): Formal ontology and information systems. In: Proceedings of FOIS, vol. 98, pp. 81–97.
- Gürtler, Matthias; Kortler, Sebastian; Helms, Bergen; Berkovich, Marina; Leimeister, Jan Marco; Kremer, Helmut et al. (2013): Von Anforderungslisten zum konzeptionellen Design-Funktionsbasierte Analyse von Anforderungen an Product-Service Systems. In: Dienstleistungsmodellierung 2012: Springer, pp. 96–113.
- Hailpern, Brent; Tarr, Peri (2006): Model-driven development: The good, the bad, and the ugly. In *IBM Systems Journal* 45 (3), p. 451.
- Hajimohammadi, Azadeh; Cavalcante, Juliana; Gzara, Lilia (2017): Ontology for the PSS Lifecycle Management. In *Procedia CIRP* 64, pp. 151–156.
- Hamraz, Bahram; Caldwell, Nicholas H. M.; Clarkson, P. John (2013): A holistic categorization framework for literature on engineering change management. In *Systems Engineering* 16 (4), pp. 473–505.
- Hao, Jing (2012): The evolution of product-service system business model: A case study. In: Management Science and Engineering (ICMSE), 2012 International Conference on. IEEE, pp. 790–795.
- Hayes, Jane Huffman; Dekhtyar, Alex; Janzen, David S. (2009): Towards traceable test-driven development. In: Traceability in Emerging Forms of Software Engineering, 2009. TEFSE'09. ICSE Workshop on. IEEE, pp. 26–30.
- Hazaël-Massieux, Dominique (2003): The Semantic Web and its applications at W3C. World Wide Web Consortium. Available online at <http://www.w3.org/2003/Talks/simo-semwebapp/>.
- Heindl, Matthias; Biffel, Stefan (2005): A case study on value-based requirements tracing. In ACM (Ed.): Proceedings of the 10th European software engineering conference held jointly

with 13th ACM SIGSOFT international symposium on Foundations of software engineering. Lisbon, Portugal: ACM, pp. 60–69.

Hepp, Martin (2011): GoodRelations Language Reference V 1.0. Available online at <http://purl.org/goodrelations/v1>, checked on 7/30/2017.

Hepperle, C.; Orawski, R.; Nolte, B.D; Mörtl, M.; Lindemann, U. (2010): An integrated lifecycle model of product-service-systems. In *CIRP IPS2 Conference, Linköping*.

Herberg, Arne; Langer, Stefan; Netter, F.; Lindemann, U. (2010): Characterizing triggers of reactive cycles within design processes based on process observation. In: *Industrial Engineering and Engineering Management (IEEM), 2010 IEEE International Conference on*. IEEE, pp. 972–976.

Herzfeldt, A.; Schermann, M.; Krcmar, H. (2012): A Product Service System Lifecycle Model for the IT Service Industry. In. *Multikonferenz der Wirtschaftsinformatik (MKWI 2012)*. Braunschweig, pp. 53–64.

Herzfeldt, Alexander; Briggs, Robert O.; Read, Aaron; Krcmar, Helmut (2011): Towards a Taxonomy of Requirements for Hybrid Products. In: *System Sciences (HICSS), 2011 44th Hawaii International Conference on*. IEEE, pp. 1–10.

Herzfeldt, Alexander; Schermann, Michael; Krcmar, H. (2010): Towards a set of requirements for a holistic IT solution engineering approach. In: *Australasian Conference on Information Systems*, pp. 1–10.

Hevner, Alan R. (2007): A three cycle view of design science research. In *Scandinavian journal of information systems* 19 (2), p. 4.

Hevner, Alan R.; March, Salvatore T.; Park, Jinsoo; Ram, Sudha (2004): Design science in information systems research. In *MIS Q* 28 (1), pp. 75–105.

Hicks, Ben J.; Culley, Steve J.; Allen, R. D.; Mullineux, Glen (2002): A framework for the requirements of capturing, storing and reusing information and knowledge in engineering design. In *International journal of information management* 22 (4), pp. 263–280.

Highsmith, J. (2002): What is Agile Software Development? STSC Crosstalk. In *Journal of Defense Software Engineering*.

Hildenbrand, Tobias (2008): Improving traceability in distributed collaborative software development. A design science approach. Frankfurt, M. u.a.: Lang.

Hitzler, Pascal; Krötzsch, Markus; Rudolph, Sebastian; Sure, York (2007): *Semantic Web: Grundlagen*: Springer-Verlag.

Holtzblatt, Karen; Beyer, Hugh R. (2013): Contextual Design. In Rikke Friis Dam, Mads Soegaard (Eds.): *The Encyclopedia of Human-Computer Interaction*, vol. 2. Aarhus, Denmark: The Interaction Design Foundation, p. 8. Available online at <http://www.interaction-design.org/books/hci.html>, checked on 6/14/2013.

Huang, G. Q.; Mak, K. L. (1999): Current practices of engineering change management in UK manufacturing industries. In *International Journal of Operations & Production Management* 19 (1), pp. 21–37.

Huang, G. Q.; Yee, W. Y.; Mak, K. L. (2001): Development of a web-based system for engineering change management. In *Robotics and Computer-Integrated Manufacturing* 17 (3), pp. 255–267.

- Isaksson, Ola; Larsson, Tobias C.; Rönnbäck, Anna Öhrwall (2009): Development of product-service systems: challenges and opportunities for the manufacturing firm. In *Journal of Engineering Design* 20 (4), pp. 329–348.
- Jarke, Matthias (1998): Requirements Tracing. In *Communications of the ACM* 41 (12), pp. 32–36.
- Jarratt, T.; Clarkson, J. (2005): Engineering change. In J. Clarkson, C. M. Eckert (Eds.): Design process improvement. London: Springer, pp. 262–285.
- Jarratt, T. A.W.; Eckert, Claudia M.; Caldwell, N. H.M.; Clarkson, P. John (2011): Engineering change: an overview and perspective on the literature. In *Research in engineering design* 22 (2), pp. 103–124.
- Johansson, Juliet E.; Krishnamurthy, Chandru; Schliessberg, Henry E. (2003): Solving the solutions problem. In *McKinsey Quarterly* (3), pp. 116–125.
- Jones, Dean; Bench-Capon, Trevor; Visser, Pepijn (1998): Methodologies for ontology development. In: IFIP world computer congress, pp. 62–75.
- Kannenbergh, Andrew; Saiedian, Hossein (2010): Why Software Requirements Traceability Remains a Challenge. In *Journal of the Quality Assurance Institute* 24 (2), pp. 4–7.
- Kernschmidt, Konstantin; Vogel-Heuser, Birgit (2013): An interdisciplinary SysML based modeling approach for analyzing change influences in production plants to support the engineering. In: Automation Science and Engineering (CASE), 2013 IEEE International Conference on. IEEE, pp. 1113–1118.
- Kernschmidt, Konstantin; Wolfenstetter, Thomas; Münzberg, Christopher; Kammerl, Daniel; Goswami, Suparna; Lindemann, U. et al. (2013): Concept for an Integration-framework to enable the crossdisciplinary development of product-service-systems. In IEEE (Ed.): Proceedings of the International Conference on Industrial Engineering and Engineering Management (IEEM) 2013.
- Kimita, K.; Tateyama, T.; Shimomura, Y. (2012): Process Simulation Method for Product-Service Systems Design. In *Procedia CIRP* 3, pp. 489–494.
- Kimura, Fumihiko; Kato, Satoru (2002): Life cycle management for improving product service quality. In: Proceedings of the 9th International Seminar on Life Cycle Engineering, Erlangen (Germany), pp. 25–31.
- Kirova, V.; Kirby, N.; Kothari, D.; Childress, G. (2008): Effective requirements traceability: Models, tools, and practices. In *Bell Labs Technical Journal* 12 (4), pp. 143–157.
- Klingner, Stephan; Becker, Michael (2012): Formal Modelling of Components and Dependencies for Configuring Product-Service-Systems. In *Enterprise Modelling and Information Systems Architectures* 7 (1), pp. 44–66.
- Knackstedt, Ralf; Pöppelbuß, Dipl.-Wirt-Inf. Jens; Winkelmann, Axel (2008): Integration von Sach- und Dienstleistungen-Ausgewählte Internetquellen zur hybriden Wertschöpfung. In *Wirtschaftsinformatik* 50 (3), pp. 235–247.
- Knethen, Antje von; Paech, B.; Kiedaisch, F.; Houdek, F. (2002): Systematic requirements recycling through abstraction and traceability. In IEEE (Ed.): Proceedings of the IEEE Joint International Conference on Requirements Engineering. Joint International Conference on Requirements Engineering. Essen, Germany, 9-13 Sept. 2002, pp. 273–281.

- Koh, E. C. Y.; Keller, R.; Eckert, C. M.; Clarkson, P. J.; others (2008): Influence of feature change propagation on product attributes in concept selection. In: DS 48: Proceedings DESIGN 2008, the 10th International Design Conference, Dubrovnik, Croatia.
- Koh, Edwin C. Y.; Caldwell, Nicholas H. M.; Clarkson, P. John (2012): A method to assess the effects of engineering change propagation. In *Research in engineering design* 23 (4), pp. 329–351.
- Komoto, H.; Tomiyama, T. (2008): Integration of a service CAD and a life cycle simulator. In *CIRP Annals-Manufacturing Technology* 57 (1), pp. 9–12.
- Komoto, Hitoshi; Tomiyama, Tetsuo; Nagel, Menno; Silvester, Sacha; Brezet, Han (2005): Life cycle simulation for analyzing product service systems. In: Environmentally Conscious Design and Inverse Manufacturing, 2005. Eco Design 2005. Fourth International Symposium on. IEEE, pp. 386–393.
- Kotonya, G.; Sommerville, I. (1998): Requirements Engineering. Processes and Techniques. 1. Auflage. West Sussex: John Wiley & Sons, Inc.
- Krcmar, Helmut (2009): Informationsmanagement. 5th edition. Heidelberg: Springer.
- Krueger, Charles W. (1992): Software reuse. In *ACM Computing Surveys (CSUR)* 24 (2), pp. 131–183.
- Langer, Philipp (2013): Angebotsmanagement für hybride IT-Produkte: Prozess- und Datenmodelle für den Vertrieb kundenindividueller IT-Lösungen: Springer-Verlag.
- Langer, Stefan; Herberg, Arne; Körber, Klaus; Lindemann, Udo (2011): Integrated system and context modeling of iterations and changes in development processes. In: DS 68-1: Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society through Engineering Design, Vol. 1: Design Processes, Lyngby/Copenhagen, Denmark, 15.-19.08. 2011.
- Langer, Stefan; Kreimeyer, Matthias; Müller, Patrick; Lindemann, Udo; Blessing, Lucienne (2009): Entwicklungsprozesse hybrider Leistungsbündel-Evaluierung von Modellierungsmethoden unter Berücksichtigung zyklischer Einflussfaktoren. In: Dienstleistungsmodellierung: Springer, pp. 71–87.
- Langer, Stefan Frederik; Lindemann, Udo (2009): Managing cycles in development processes-analysis and classification of external context factors. In: DS 58-1: Proceedings of ICED 09, the 17th International Conference on Engineering Design, Vol. 1, Design Processes, Palo Alto, CA, USA, 24.-27.08. 2009.
- Laperche, Blandine; Picard, Fabienne (2013): Environmental constraints, Product-Service Systems development and impacts on innovation management: learning from manufacturing firms in the French context. In *Journal of Cleaner Production* 53, pp. 118–128.
- Laurel, Brenda; Mountford, S. Joy (1990): The art of human-computer interface design: Addison-Wesley Longman Publishing Co., Inc.
- Laurischkat, Katja (2013): Wandel des traditionellen Dienstleistungsverständnisses im Kontext von Product-Service Systems. In: Dienstleistungsmodellierung 2012: Springer, pp. 74–95.
- Lee, Christopher; Guadagno, Luigi; Jia, Xiaoping (2003): An agile approach to capturing requirements and traceability. In: Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE 2003). Citeseer.

- Lee, Ji Hwan; Shin, Dong Ik; Hong, Yoo S.; Kim, Yong Se (2011): Business model design methodology for innovative product-service systems: a strategic and structured approach. In: SRII Global Conference (SRII), 2011 Annual. IEEE, pp. 663–673.
- Lee, S. W.; Kim, Y. S. (2012): Product-Service Systems Design Method Integrating Service Function and Service Activity and Case Studies. In: Proceedings of the 2nd CIRP IPS2 Conference 2010; 14-15 April; Linköping; Sweden. Linköping University Electronic Press, pp. 275–282.
- Lee, Sora; Park, Yongtae (2010): Evaluation of PSS concepts for successful shift from product to PSS: an approach based on AHP and niche theory. In: Industrial Engineering and Engineering Management (IEEM), 2010 IEEE International Conference on. IEEE, pp. 453–457.
- Leffingwell, Dean; Widrig, Don (2003): Managing Software Requirements. A Unified Approach. Amsterdam: Addison-Wesley Longman.
- Leimeister, Jan Marco (2012): Dienstleistungsengineering und-management: Springer-Verlag.
- Leimeister, Jan Marco; Glauner, Christoph (2008): Hybride Produkte – Einordnung und Herausforderungen für die Wirtschaftsinformatik. In *Wirtschaftsinformatik* 50 (3), pp. 248–251. DOI: 10.1365/s11576-008-0051-z.
- Li, Fang; Bayrak, GAL’lden; Kernschmidt, Konstantin; Vogel-Heuser, Birgit (2012): Specification of the requirements to support information technology-cycles in the machine and plant manufacturing industry. In: Information Control Problems in Manufacturing, vol. 14, pp. 1077–1082.
- Li, Xiao; Liu, Zheng Gang (2010): An evolution framework of product service system for firms across service supply chains with integrated lifecycle perspective. In: Logistics Systems and Intelligent Management, 2010 International Conference on, vol. 1. IEEE, pp. 430–434.
- Li, Yang; Maalej, Walid (2012): Which Traceability Visualization Is Suitable in This Context? A Comparative Study. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell et al. (Eds.): Requirements Engineering: Foundation for Software Quality, vol. 7195. Berlin, Heidelberg: Springer Verlag (Lecture Notes in Computer Science), pp. 194–210.
- Liebowitz, Jay (1999): Knowledge management handbook: CRC press.
- Lim, Chie-Hyeon; Kim, Kwang-Jae; Hong, Yoo-Suk; Park, Kwangtae (2012): PSS Board. A structured tool for product-service system process visualization. In *Journal of Cleaner Production* 37, pp. 42–53.
- Lin, Jinxin; Fox, Mark S.; Bilgic, T. (2006): A Requirement Ontology for Engineering Design. In *Concurrent Engineering: Research and Applications* (4), pp. 279–291.
- Lindemann, Udo (2009): Methodische Entwicklung Technischer Produkte: Methoden Flexibel und Situationsgerecht Anwenden: Springer.
- Lindemann, Udo; Reichwald, Ralf (2013): Integriertes Änderungsmanagement: Springer-Verlag.
- Liu, Xing; Qian, Xiaobo; Zhou, Xiaojiang (2010): A new approach to realize sustainability—Integrated design under the product and service system framework. In IEEE (Ed.): 11th International Conference on Computer-Aided Industrial Design & Conceptual Design.

- Lucas, Francisco J.; Molina, Fernando; Toval, Ambrosio (2009): A systematic review of UML model consistency management. In *Information and Software Technology* 51 (12), pp. 1631–1645.
- Lucia, Andrea De; Qusef, Abdallah (2010): Requirements engineering in agile software development. In *Journal of Emerging Technologies in Web Intelligence* 2 (3), pp. 212–220.
- Luiten, Helma; Knot, Marjolijn; van der Horst, Tom (2001): Sustainable product-service-systems: the Kathalys method. In: *Environmentally Conscious Design and Inverse Manufacturing, 2001. Proceedings EcoDesign 2001: Second International Symposium on*. IEEE, pp. 190–197.
- Mäder, Patrick; Gotel, Orlena (2012): Towards automated traceability maintenance. In *Journal of Systems and Software* 85 (10), pp. 2205–2227.
- Maeder, Patrick; Egyed, Alexander (2012): Assessing the effect of requirements traceability for software maintenance. In IEEE (Ed.): *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pp. 171–180.
- Maeder, Patrick; Riebisch, Matthias; Philippow, Ilka (2006): Traceability for Managing Evolutionary Change. In: *SEDE*, pp. 1–8.
- Maletic, Jonathan I.; Munson, Ethan V.; Marcus, Andrian; Nguyen, Tien N. (2003): Using a Hypertext Model for Traceability Link Conformance Analysis. In *Proc. of the Int. Workshop on Traceability in Emerging Forms of Software Engineering*.
- Mannweiler, Carsten; Möhrer, Jürgen; Fiekers, Christoph (2010): Planung investiver Produkt-Service Systeme. In: *Produkt-Service Systeme: Springer*, pp. 15–30.
- Manzini, Ezio; Vezzoli, Carlo (2003): Product-service Systems and Sustainability: Opportunities for Solutions: UNEP, Division of Technology Industry and Economics, Production and Consumption Branch.
- Marques, Pedro; Cunha, Pedro F.; Valente, Fernando; Leitão, Ana (2013): A methodology for product-service systems development. In *Procedia CIRP* 7, pp. 371–376.
- Maussang, Nicolas; Zwolinski, Peggy; Brissaud, Daniel (2009): Product-service system design methodology: from the PSS architecture design to the products specifications. In *Journal of Engineering Design* 20 (4), pp. 349–366.
- Medvidovic, Nenad; Grünbacher, Paul; Egyed, Alexander; Boehm, Barry W. (2003): Bridging models across the software lifecycle. In *Journal of Systems and Software* 68 (3), pp. 199–215.
- Meier, Horst; Roy, Raj; Seliger, Günther (2010): Industrial product-service systems—IPS 2. In *CIRP Annals-Manufacturing Technology* 59 (2), pp. 607–627.
- Mens, Tom; van Gorp, Pieter (2006): A taxonomy of model transformation. In *Electronic Notes in Theoretical Computer Science* 152, pp. 125–142.
- Mien, Lee Hui; Feng, Lu Wen; Gay, R. (2005): An integrated manufacturing and product services system (IMPSS) concept for sustainable product development. In: *Fourth International Symposium on Environmentally Conscious Design and Inverse Manufacturing, 2005. Eco Design 2005*. IEEE, pp. 656–662.
- Mohan, Kannan; Ramesh, Balasubramaniam (2006): Change management patterns in software product lines. In *Communications of the ACM* 49 (12), pp. 68–72.
- Mont, Oksana (2000): Product-Service Systems. Final Report.

- Mont, Oksana (2002): Clarifying the concept of product-service systems. In *Journal of Cleaner Production* 10 (3), pp. 237–245.
- Mont, Oksana; Tukker, Arnold (2006): Product-Service Systems: reviewing achievements and refining the research agenda. In *Journal of Cleaner Production* 14 (17), pp. 1451–1454.
- Morcos, M. S.; Henshaw, M. J.D. (2009): A systems approach for balancing internal company capability and external client demand for integrated product-service solutions. In: Service Operations, Logistics and Informatics, 2009. SOLI'09. IEEE/INFORMS International Conference on. IEEE, pp. 32–36.
- Morelli, Nicola (2002): Designing Product/Service Systems: A Methodological Exploration. In *Design issues* 18 (3), pp. 3–17.
- Morelli, Nicola (2006): Developing new product service systems (PSS): methodologies and operational tools. In *Journal of Cleaner Production* 14 (17), pp. 1495–1501.
- Morkos, Beshoy; Shankar, Prabhu; Summers, Joshua D. (2012): Predicting requirement change propagation, using higher order design structure matrices: an industry case study. In *Journal of Engineering Design* 23 (12), pp. 905–926.
- Nemoto, Yutaro; Akasaka, Fumiya; Shimomura, Yoshiki (2015): A framework for managing and utilizing product-service system design knowledge. In *Production Planning & Control* 26 (14-15), pp. 1278–1289.
- Nerur, Sridhar; Mahapatra, RadhaKanta; Mangalaraj, George (2005): Challenges of migrating to agile methodologies. In *Communications of the ACM* 48 (5), pp. 72–78.
- Nordin, F.; Kowalkowski, C. (2010): Solutions Offerings. A critical Review and Reconceptualisation. In *Journal of Service Management* 21 (4), pp. 441–459.
- Noy, Natalya F. (2004): Semantic integration: a survey of ontology-based approaches. In *ACM Sigmod Record* 33 (4), pp. 65–70.
- Noy, Natalya F.; McGuinness, Deborah L.; others (2001): Ontology development 101: A guide to creating your first ontology: Stanford knowledge systems laboratory technical report KSL-01-05 and Stanford medical informatics technical report SMI-2001-0880, Stanford, CA.
- Object Management Group (2011a): Meta Object Facility (MOF). Version 2.4.1.
- Object Management Group (2011b): “OMG Unified Modeling Language (OMG UML™). Version 2.4.1.
- Object Management Group (2012): OMG Systems Modeling Language (OMG SysML™). Version 1.3.
- Orloff, M. A. (2006): *Inventive Thinking through TRIZ: A Practical Guide*. Heidelberg: Springer.
- Osterwalder, Alexander; Pigneur, Yves (2010): *Business model generation: a handbook for visionaries, game changers, and challengers*: John Wiley & Sons.
- Pahl, Gerhard; Beitz, Wolfgang; Feldhusen, Jörg (2006): *Konstruktionslehre. Grundlagen Erfolgreicher Produktentwicklung. Methoden Und Anwendung*. 7th ed. Berlin: Springer.
- Patel, Manish; Nagl, Sylvia (2010): Coping with Complexity: Modelling of Complex Systems. In *The Role of Model Integration in Complex Systems Modelling*, pp. 33–55.

- Pavković, Neven; Štorga, Mario; Bojčetić, Nenad; Marjanović, Dorian (2013): Facilitating design communication through engineering information traceability. In *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 27 (02), pp. 105–119.
- Peppers, Ken; Tuunanen, Tuure; Rothenberger, Marcus A.; Chatterjee, Samir (2007): A design science research methodology for information systems research. In *Journal of Management Information Systems* 24 (3), pp. 45–77.
- Phumbua, Sarocha; Tjahjono, Benny (2012): Towards product-service systems modelling: a quest for dynamic behaviour and model parameters. In *International Journal of Production Research* 50 (2), pp. 425–442.
- Ping, Wang Li; Jia, Fu (2010): Analysis on supply chain of manufacturing enterprise product service system. In: Emergency Management and Management Sciences (ICEMMS), 2010 IEEE International Conference on. IEEE, pp. 126–129.
- Pinheiro, Francisco A.C. (2004): Requirements Traceability. In JulioCesarSampaio Prado Leite, JorgeHoracio Doorn (Eds.): Perspectives on Software Requirements, vol. 753: Springer US (The Springer International Series in Engineering and Computer Science), pp. 91–113.
- Pohl, K. (2008): Requirements Engineering. Grundlagen, Prinzipien, Techniken. 2. Auflage. Heidelberg: dpunkt.verlag.
- Pohl, Klaus (1996a): PRO-ART: Enabling Requirements Pre-Traceability. In IEEE Computer Society (Ed.): Proceedings of the 2nd International Conference on Requirements Engineering. International Conference on Requirements Engineering. Colorado Springs, 15-18. April 1996. Washington, DC: IEEE Computer Society, pp. 76–84.
- Pohl, Klaus (1996b): Process-centered requirements engineering. Taunton, Somerset, England, New York: Research Studies Press; Wiley.
- Pohl, Klaus (2010): Requirements engineering: fundamentals, principles, and techniques: Springer Publishing Company, Incorporated.
- Pohl, Klaus; Rupp, Chris (2010): Basiswissen Requirements Engineering. Aus- und Weiterbildung zum "Certified Professional for Requirements Engineering" ; Foundation Level nach IREB-Standard. 2nd ed. Heidelberg: dpunkt-Verl.
- Ponn, Josef; Lindemann, Udo (2008): Konzeptentwicklung und Gestaltung technischer Produkte: optimierte Produkte-systematisch von Anforderungen zu Konzepten: Springer-Verlag.
- Qian, Lena; Gero, John S. (1996): Function–behavior–structure paths and their role in analogy-based design. In *AIEDAM* 10 (04), p. 289.
- Qu, Min; Yu, Suihuai; Chen, Dengkai; Chu, Jianjie; Tian, Baozhen (2016): State-of-the-art of design, evaluation, and operation methodologies in product service systems. In *Computers in Industry* 77, pp. 1–14.
- Qusef, Abdallah (2013): Test-to-code traceability: Why and how? In: Applied Electrical Engineering and Computing Technologies (AEECT), 2013 IEEE Jordan Conference on. IEEE, pp. 1–8.
- Radatz, Jane; Geraci, Anne; Katki, Freny (1990): IEEE standard glossary of software engineering terminology. In *IEEE Std 610121990* (121990), p. 3.
- Ramesh, Balasubramaniam (1998): Factors influencing requirements traceability practice. In *Commun. ACM* 41 (12), pp. 37–44.

- Ramesh, Balasubramaniam (2002): Process knowledge management with traceability. In *Software, IEEE* 19 (3), pp. 50–52.
- Ramesh, Balasubramaniam; Edwards, M. (1993): Issues in the development of a requirements traceability model. In IEEE (Ed.): Proceedings of the IEEE International Symposium on Requirements Engineering. International Symposium on Requirements Engineering. San Diego, CA, USA, 4-6 Jan. 1993, pp. 256–259.
- Ramesh, Balasubramaniam; Jarke, Matthias (2001): Towards Reference Models for Requirements Traceability. In *IEEE Transactions on Software Engineering* 27 (1), pp. 58–93.
- Ramesh, Balasubramaniam; Stubbs, Curtis; Powers, Timothy; Edwards, Michael (1997): Requirements traceability: Theory and practice. In *Annals of Software Engineering* 3 (1), pp. 397–415.
- Ravichandar, Ramya; Arthur, James D.; Pérez-Quñones, Manuel (2007): Pre-requirement specification traceability: bridging the complexity gap through capabilities. In *arXiv preprint cs/0703012*.
- Reichwald, R.; Mayer, D.; Bonnemeier, S. (2009): Risikomanagement bei hybrider Wertschöpfung. In *Risikomanagement und kapitalmarktorientierte Finanzierung* 1, pp. 209–228.
- Rezayat, Mohsen (2000): Knowledge-based product development using XML and KCs. In *Computer-aided design* 32 (5), pp. 299–309.
- Riebisch, M. (2004): Supporting evolutionary development by feature models and traceability links. In IEEE Computer Society (Ed.): Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems. Brno, Czech Republic, 24-27 May 2004, pp. 370–377.
- Robertson, Suzanne; Robertson, James (2006): Mastering the requirements process. 2nd ed. Upper Saddle River, NJ: Addison-Wesley.
- Roht, Olga; Engel, Tobias; Wolfenstetter, Thomas; Goswami, Suparna; Krcmar, Helmut (2014): An Analysis of Synergy Effects between Closed-Loop Supply Chains and Product-Service Systems. In *POMS International Meeting, Atlanta*.
- Rosemann, Michael (2013): Komplexitätsmanagement in Prozessmodellen: methodenspezifische Gestaltungsempfehlungen für die Informationsmodellierung: Springer-Verlag.
- Rouibah, Kamel; Caskey, Kevin R. (2003): Change management in concurrent engineering from a parameter perspective. In *Computers in Industry* 50 (1), pp. 15–34.
- Roy, Rajkumar; Shehab, Essam; Tiwari, Ashutosh; Sakao, Tomohiko; Ölundh Sandström, Gunilla; Matzen, Detlef (2009a): Framing research for service orientation of manufacturers through PSS approaches. In *Journal of Manufacturing Technology Management* 20 (5), pp. 754–778.
- Roy, Rajkumar; Shehab, Essam; Tiwari, Ashutosh; Sundin, Erik; Lindahl, Mattias; Ijomah, Winifred (2009b): Product design for product/service systems: design experiences from Swedish industry. In *Journal of Manufacturing Technology Management* 20 (5), pp. 723–753.
- Sahraoui, A. (2005): Requirements Traceability Issues: Generic Model, Methodology and Formal Basis. In *International Journal of Information Technology & Decision Making* 4 (1), pp. 59–80.

- Sakao, Tomohiko; Panshef, Veselin; Dörsam, Edgar (2009a): Addressing uncertainty of PSS for value-chain oriented service development: Springer.
- Sakao, Tomohiko; Shimomura, Yoshiki; Sundin, Erik; Comstock, Mica (2009b): Modeling design objects in CAD system for service/product engineering. In *Computer-aided design* 41 (3), pp. 197–213.
- Sauerwein, Elmar; Bailom, Franz; Matzler, Kurt; Hinterhuber, Hans H. (1996): The Kano Model: How to delight your customers. In: Preprints of the International Working Seminar on Production Economics, vol. 1. International Working Seminar on Production Economics. Innsbruck, 19. - 23. Februar 1996, pp. 313–327.
- Sawhney, Mohanbir (2006): Going beyond the product. Defining, Designing and Delivering Customer Solutions. In *The service-dominant logic of marketing: Dialogue, debate, and directions*, pp. 365–380.
- Sawhney, Mohanbir; Wolcott, Robert C.; Arroniz, Inigo (2006): The 12 different ways for companies to innovate. In *MIT Sloan Management Review* 47 (3), p. 75.
- Schaffry, A. (2008): Manufacturing Execution Systeme. Available online at <http://www.cio.de/strategien/methoden/847558/index2.html>, checked on 8/18/2014.
- Scheer, August-Wilhelm (1992): Architektur integrierter Informationssysteme. Grundlagen der Unternehmensmodellierung. 2nd ed. Berlin u.a: Springer.
- Schenkl, Sebastian A.; Behncke, Florian G. H.; Hepperle, Clemens; Langer, Steven; Lindemann, Udo (2013): Managing cycles of innovation processes of product-service systems. In: Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on. IEEE, pp. 918–923.
- Schienmann, Bruno (2001): Kontinuierliches Anforderungsmanagement. Prozesse - Techniken - Werkzeuge. 1st ed.: Addison-Wesley.
- Schmitz, G. (2008): Der wahrgenommene Wert hybrider Produkte. Konzeptionelle Grundlagen und Komponenten. In M. Bichler, T. Hess, H. Krcmar, U. Lechner, F. Matthes, A. Picot et al. (Eds.). Multikonferenz der Wirtschaftsinformatik (MKWI 2008). Berlin, pp. 665–683.
- Schütte, Reinhard (2013): Grundsätze ordnungsmäßiger Referenzmodellierung: Konstruktion konfigurations- und anpassungsorientierter Modelle: Springer-Verlag (233).
- Schweitzer, Eric (2010): Lebenszyklusmanagement investiver Produkt-Service Systeme. In: Produkt-Service Systeme: Springer, pp. 7–13.
- Sevcenko, M.; Mann, H. (2002): Intelligent user-support system for modeling and simulation. In: Computer Aided Control System Design, 2002. Proceedings. 2002 IEEE International Symposium on. IEEE, pp. 205–206.
- Shah, Aditya A.; Kerzhner, Aleksandr A.; Schaefer, Dirk; Paredis, Christiaan J. J. (2010): Multi-view modeling to support embedded systems engineering in SysML. In: Graph transformations and model-driven engineering: Springer, pp. 580–601.
- Shah, Aditya A.; Schaefer, Dirk; Paredis, Christiaan (2009): Enabling multi-view modeling with sysml profiles and model transformations. In: The 6th International Conference on Product Lifecycle Management. University of Bath, pp. 527–538.
- Sharafi, Armin; Wolf, Petra; Krcmar, Helmut (2010a): Knowledge Discovery in Databases on the Example of Engineering Change Management. In: Industrial Conference on Data Mining-Poster and Industry Proceedings, pp. 9–16.

- Sharafi, Armin; Wolfenstetter, Thomas; Wolf, Petra; Krcmar, Helmut (2010b): Comparing Product Development Models to Identify Process Coverage and Current Gaps. A Literature Review. In Lian Zhaotong, Wu Zhang, Xie Min, Jiao Roger (Eds.): Proceedings of the International Conference on Industrial Engineering and Engineering Management. Macau, 2010: Springer, pp. 1732–1737.
- Shostack, G. Lynn (1977): Breaking free from product marketing. In *The Journal of Marketing*, pp. 73–80.
- Shostack, G. Lynn (1982): How to Design a Service. In *European Journal of Marketing* 16 (1), pp. 49–63. DOI: 10.1108/EUM0000000004799.
- Sommerville, Ian (2011): Software Engineering. 9th ed.: Addison-Wesley.
- Sommerville, Ian; Kotonya, Gerald (1998): Requirements engineering: processes and techniques: John Wiley & Sons, Inc.
- Song, Wenyan (2017): Requirement management for product-service systems. Status review and future trends. In *Computers in Industry* 85, pp. 11–22.
- Spanoudakis, George; Zisman, Andrea (2005): Software Traceability: A Roadmap. In Shi Kuo Chang (Ed.): Handbook of software engineering & knowledge engineering. New Jersey [etc.]: World Scientific, pp. 395–428.
- Spath, Dieter; Demuß, Lutz (2006): Entwicklung hybrider Produkte—Gestaltung materieller und immaterieller Leistungsbündel. In H. J. Bullinger, A.-W Scheer (Eds.): Service Engineering. Entwicklung und Gestaltung innovativer Dienstleistungen. 3rd ed. Berlin: Springer, pp. 463–502.
- Stachowiak, Herbert (1973): Allgemeine Modelltheorie. Wien [etc.]: Springer.
- Storga, M. (2004): Traceability in product development. In: DS 32: Proceedings of DESIGN 2004, the 8th International Design Conference, Dubrovnik, Croatia.
- Štorga, Mario; Marjanovic, Dorian; Savšek, Tomaz; others (2011): Reference model for traceability records implementation in engineering design environment. In: DS 68-6: Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society through Engineering Design, Vol. 6: Design Information and Knowledge, Lyngby/Copenhagen, Denmark, 15.-19.08. 2011.
- Strens, M.R.; Sugden, R.C. (1996): Change analysis: a step towards meeting the challenge of changing requirements. In IEEE (Ed.): Proceedings of the IEEE Symposium and Workshop on Engineering of Computer-Based Systems. Symposium and Workshop on Engineering of Computer-Based Systems. Friedrichshafen, Germany, 11-15 March 1996, pp. 278–283.
- Sturm, F.; Bading, A. (2008): Investitionsgüterhersteller als Anbieter industrieller Lösungen – Bestandsaufnahme des Wandels anhand einer Umfrage. In *Wirtschaftsinformatik* 50 (3), pp. 174–186.
- Sun, Huibin; Zhang, Guohai (2012): Study on collaborative design methodologies of product service systems. In: Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on. IEEE, pp. 882–884.
- Tan, Adrian Ronald; Matzen, Detlef; McAloone, Tim C.; Evans, Stephen (2010): Strategies for designing and developing services for manufacturing firms. In *CIRP Journal of Manufacturing Science and Technology* 3 (2), pp. 90–97.

- Tan, Adrian Ronald; McAloone, Tim Charles; Gall, Catherine (2007): Product/Service-System Development. An Explorative Case Study in a Manufacturing Company. In *Proceedings of the International Conference on Engineering Design (ICED2007)*.
- Tang, Antony; Jin, Yan; Han, Jun (2006): A rationale-based architecture model for design traceability and reasoning. In *Journal of Systems and Software* 80 (6), pp. 918–934. DOI: 10.1016/j.jss.2006.08.040.
- Teng, Sheng-Hsien; Ho, Shin-Yann (1996): Failure mode and effects analysis: an integrated approach for product design and process control. In *International journal of quality & reliability management* 13 (5), pp. 8–26.
- Thramboulidis, Kleanthis (2010): The 3+ 1 SysML view-model in model integrated mechatronics. In *Journal of Software Engineering and Applications* 3 (02), p. 109.
- Tilstra, Andrew H.; Campbell, Matthew I.; Wood, Kristin L.; Seepersad, Carolyn C. (2010): Comparing Matrix-Based and Graph-Based Representations for Product Design. In: DSM 2010: Proceedings of the 12th International DSM Conference, Cambridge, UK, 22.-23.07. 2010.
- Torkar, Richard; Gorschek, Tony; Feldt, Robert; Svahnberg, Mikael; Raja, Uzair Akbar; Kamran, Kashif (2012): Requirements traceability: a systematic review and industry case study. In *International Journal of Software Engineering and Knowledge Engineering* 22 (03), pp. 385–433.
- Tratt, Laurence (2005): Model transformations and tool integration. In *Software and Systems Modeling* 4 (2), pp. 112–122.
- Tukker, A. (2004): Eight Types of Product-Service System. Eight Ways to Sustainability? Experiences from SusProNet. In *Business strategy and the environment* 13 (4), pp. 246–260.
- Tukker, A.; Tischner, U. (2006): Product-Services as a Research Field. Past, Present and Future. Reflections from a Decade of Research. In *Journal of Cleaner Production* 14 (17), pp. 1552–1556.
- Tuli, K. R.; Kohli, A. K.; Bharadwaj, S. G. (2007): Rethinking Customer Solutions. From Product Bundles to Relational Processes. In *Journal of Marketing* 71 (3), pp. 1–17.
- Ulrich, K. T.; Eppinger, S. D. (2012): *Product Design and Development*: McGraw-Hill Education.
- van Aken, Joan Ernst (2005): Management research as a design science. Articulating the research products of mode 2 knowledge production in management. In *British journal of management* 16 (1), pp. 19–36.
- van Halen, Cees; Vezzoli, Carlo; Wimmer, Robert (2005): *Methodology for product service system innovation. How to develop clean, clever and competitive strategies in companies*: Uitgeverij Van Gorcum.
- van Lamsweerde, A. (2007): *Requirements engineering. From system goals to UML models and software specifications*. Hoboken, N.J, Chichester: Wiley.
- van Ostaeyen, Joris; van Horenbeek, Adriaan; Pintelon, Liliane; Duflou, Joost R. (2013): A refined typology of product-service systems based on functional hierarchy modeling. In *Journal of Cleaner Production* 51, pp. 261–276.
- Varró, Dániel; Pataricza, András (2004): Generic and meta-transformations for model transformation engineering. In: \guillemotright 2004—The Unified Modeling Language. Modeling Languages and Applications: Springer, pp. 290–304.

- Vasantha, Gokula Vijaykumar Annamalai; Roy, Rajkumar; Lelah, Alan; Brissaud, Daniel (2012): A review of product-service systems design methodologies. In *Journal of Engineering Design* 23 (9), pp. 635–659.
- VDI/VDE (2005): Norm 3682 “Formalised process descriptions”.
- Velamuri, V. K.; Neyer, A.-K.; Möslin, K. M. (2011): Hybrid Value Creation. A systematic Review of an evolving Research Area. In *Journal für Betriebswirtschaft* 61 (1), pp. 3–35.
- Versteegen, Gerhard (2003): Anforderungsmanagement. 1st ed. Berlin: Springer.
- Vezzoli, Carlo; Ceschin, Fabrizio; Diehl, Jan Carel; Kohtala, Cindy (2012): Why have ‘Sustainable Product-Service Systems’ not been widely implemented?: Meeting new design challenges to achieve societal sustainability. In *Journal of Cleaner Production* 35, pp. 288–290.
- Vezzoli, Carlo; Sciama, Dalia (2006): Life Cycle Design: from general methods to product type specific guidelines and checklists: a method adopted to develop a set of guidelines/checklist handbook for the eco-efficient design of NECTA vending machines. In *Journal of Cleaner Production* 14 (15), pp. 1319–1325.
- Vilela, Jéssyka; Castro, Jaelson; Martins, Luiz Eduardo G.; Gorschek, Tony (2017): Integration between requirements engineering and safety analysis. A systematic literature review. In *Journal of Systems and Software* 125, pp. 68–92.
- Vogel-Heuser, Birgit; Witsch, Daniel; Katzke, Uwe (2005): Automatic code generation from a UML model to IEC 61131-3 and system configuration tools. In: Control and Automation, 2005. ICCA’05. International Conference on, vol. 2. IEEE, pp. 1034–1039.
- Vom Brocke, Jan; Simons, Alexander; Niehaves, Bjoern; Riemer, Kai; Plattfaut, Ralf; Cleven, Anne; others (2009): Reconstructing the giant: On the importance of rigour in documenting the literature search process. In: ECIS, vol. 9, pp. 2206–2217.
- Wang, P. P.; Ming, X. G.; Li, D.; Kong, F. B.; Wang, L.; Wu, Z. Y. (2011): Status review and research strategies on product-service systems. In *International Journal of Production Research* 49 (22), pp. 6863–6883.
- Watkins, R.; Neal, M. (1994): Why and how of requirements tracing. In *IEEE Software* 11 (4), pp. 104–106.
- Weber, C.; Pohl, M.; Steinbach, M.; Botta, C. (2002): Diskussion der Probleme bei der integrierten Betrachtung von Sach- und Dienstleistungen – „Kovalente Produkte“. In: 13. Symposium „Design for X“. Saarbrücken, pp. 61–70.
- Webster, Jane; Watson, Richard T. (2002): Analyzing the Past to Prepare for the Future: Writing a Literature Review. In *MIS Quarterly* 26 (2), pp. 13–23.
- Wiegers, K. E. (2009): Software Requirements: Microsoft Press.
- Wieringa, R. J. (2008): Conceptual modeling in social and physical contexts.
- Wieringa, Roel (2009): Design science as nested problem solving. In: Proceedings of the 4th international conference on design science research in information systems and technology. ACM, p. 8.
- Wiesner, Stefan; Freitag, Mike; Westphal, Ingo; Thoben, Klaus-Dieter (2015): Interactions between service and product lifecycle management. In *Procedia CIRP* 30, pp. 36–41.
- Wilkinson, Adrian; Dainty, Andy; Neely, Andy; Pawar, Kulwant S.; Beltagui, Ahmad; Riedel, Johann CKH (2009): The PSO triangle: designing product, service and organisation to

- create value. In *International Journal of Operations & Production Management* 29 (5), pp. 468–493.
- Williams, Laurie; Cockburn, Alistair (2003): Agile Software Development: It's about Feedback and Change. Guest Editors' Introduction. In *Computer* (6), pp. 39–43.
- Williams, R. D. (1975): Managing the development of reliable software. In *ACM SIGPLAN Notices* 10 (6), pp. 3–8.
- Winkler, Stefan; Pilgrim, Jens (2010): A survey of traceability in requirements engineering and model-driven development. In *Software and Systems Modeling (SoSyM)* 9 (4), pp. 529–565.
- Wolf, Nico; Siener, Martin; Clement, Michael H.; Jenne, Frank; Fuchs, Christian (2010): Konfiguration investiver Produkt-Service Systeme. In: *Produkt-Service Systeme*: Springer, pp. 67–94.
- Wolfenstetter, T.; Goswami, S.; Krcmar, H. (2013): Herausforderungen auf dem Weg zu einer zyklengerechten Requirements Traceability für Produkt-Service Systeme. In *Zyklusmanagement Aktuell* 4, pp. 7–9.
- Wolfenstetter, Thomas; Floerecke, Sebastian; Böhm, Markus; Krcmar, Helmut (2015a): Analyse der Eignung domänenspezifischer Methoden der Anforderungsverfolgung für Produkt-Service-Systeme. In: *Wirtschaftsinformatik*, pp. 210–224.
- Wolfenstetter, Thomas; Füller, Kathrin; Böhm, Markus; Krcmar, Helmut; Bründl, Simon (2015b): Towards a requirements traceability reference model for Product Service Systems. In: *International Conference on Industrial Engineering and Systems Management (IESM) 2015*. IEEE, pp. 1213–1220.
- Wolfenstetter, Thomas; Kernschmidt, Konstantin; Munzberg, Christopher; Kammerl, Daniel; Goswami, Suparna; Lindemann, Udo et al. (2014): Supporting the cross-disciplinary development of product-service systems through model transformations. In: *Industrial Engineering and Engineering Management (IEEM), 2014 IEEE International Conference on*. IEEE, pp. 174–178.
- Wright, I. C. (1997): A review of research into engineering change management: implications for product design. In *Design Studies* 18 (1), pp. 33–42.
- Wu, Yi; Gao, Junjun (2010): A study on the model and characteristics of product-based service supply chain. In: *Logistics systems and intelligent management, 2010 international conference on*, vol. 2. IEEE, pp. 1127–1131.
- Wurtz, Gunther; Ardilio, Antonino; Lasi, Heiner; Warschat, Joachim (2013): Towards mass individualization: Life-cycle oriented configuration of time-variable product-service systems. In: *Technology Management in the IT-Driven Services (PICMET), 2013 Proceedings of PICMET'13*. IEEE, pp. 9–25.
- Wymore, A. Wayne (1993): *Model-based systems engineering*: CRC press (3).
- Xuanju, Yang; Jian, Li (2009): Research on supplier-buyer enterprise contract cooperation model based on product-service system. In: *Industrial Engineering and Engineering Management, 2009. IE&EM'09. 16th International Conference on*. IEEE, pp. 1372–1375.
- Yang, Lujing; Xing, Ke; Lee, Sang-Heon (2010): A new conceptual life cycle model for Result-Oriented Product-Service System development. In: *IEEE International Conference on Service Operations and Logistics and Informatics (SOLI), 2010*. IEEE, pp. 23–28.

- Yang, Lujing; Xing, Ke; Lee, Sang-Heon (2011): Life-cycle Oriented Design Model for Product-service System Development. In *IEEE International Conference on Service Operations and Logistics and Informatics*.
- Yang, Xiaoyu; Moore, Philip; Pu, Jun-Sheng; Wong, Chi-Biu (2009): A practical methodology for realizing product service systems for consumer products. In *Computers & Industrial Engineering* 56 (1), pp. 224–235.
- Yip, Man Hang; Phaal, Robert; Probert, David R. (2014): Stakeholder engagement in early stage product-service system development for healthcare informatics. In *Engineering Management Journal* 26 (3), pp. 52–62.
- Yu, Eric S. (2009): Social Modeling and i*. In Alexander T. Borgida, Vinay K. Chaudhri, Paolo Giorgini, Eric S. Yu (Eds.): *Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 99–121.
- Zacharewicz, Gregory; Diallo, Saikou; Ducq, Yves; Agostinho, Carlos; Jardim-Goncalves, Ricardo; Bazoun, Hassan et al. (2017): Model-based approaches for interoperability of next generation enterprise information systems. State of the art and future challenges. In *Information Systems and e-Business Management* 15 (2), pp. 229–256.
- Zisman, Andrea; Spanoudakis, George; Pérez-Minana, Elena; Krause, Paul (2002): Towards a Traceability Approach for Product Families Requirements. In: *Proceedings of the 3rd ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications*. Orlando, Florida, USA, 19.-25. Mai 2002.
- Zolnowski, Andreas; Semmann, Martin; Böhm, Tilo (2013): Vergleich von Metamodellen zur Repräsentation von Geschäftsmodellen im Service. In: *Dienstleistungsmodellierung 2012*: Springer, pp. 26–48.