Fakultät für Elektrotechnik, Informatik und Mathematik

Situationsgerechte Methodenweiterentwicklung auf Basis von MetaMe am Beispiel der Server-System-Entwicklung

Zur
Erlangung des Grades eines
"Doktor der Naturwissenschaften"
(Dr. rer. nat.)
genehmigte

D i s s e r t a t i o n

vorgelegt von

Dipl.Inform. Michael Spijkerman

aus Paderborn.

Paderborn, Juni 2015

Diese Dissertation wurde im Januar 2015 bei der Fakultät für Elektrotechnik, Informatik und Mathematik der Universität Paderborn eingereicht. Sie wurde auf der Grundlage der Gutachten von der Promotionskommission angenommen. Die mündliche Prüfung fand am 24. März 2015 in Paderborn statt.

Gutachter:

- Prof. Dr. Gregor Engels
- Prof. Dr. Jürgen Ebert

Promotionskommission:

- Prof. Dr. Gregor Engels (Vorsitzender)
- Prof. Dr. Jürgen Ebert
- Prof. Dr. Franz J. Rammig
- Prof. Dr. Stefan Böttcher
- Dr. Stefan Sauer

Danksagung

An erster Stelle möchte ich meinem Doktorvater Prof. Dr. Gregor Engels für die wissenschaftliche Betreuung meines Themas danken. Seine Vision, ein Methodenentwicklungsthema praxisnah aufzuarbeiten, hat die vorliegende Arbeit maßgeblich geprägt. Ich danke Prof. Dr. Jürgen Ebert für die Übernahme des zweiten Gutachtens. Mein Dank geht auch an die Geschäftsführung des s-lab, Stefan Sauer und dem ehemaligen Geschäftsführer Dr. Matthias Meyer. Ihr habt mich durch viele Diskussionen über mein Thema unterstützt. Des Weiteren war es durch euren Einsatz möglich, über eine lange Zeit, die Projekte mit der Firma Fujitsu Technology Solutions GmbH durchzuführen.

Dem gegenüber steht der Dank an die Firma Fujitsu Technology Solutions GmbH, in Person Karl Lüttgenau, Bernhard Homölle, Bernhard Schräder, Georg Müller und Alois Lichtenstern dafür, dass der Nutzen meiner praktischen Arbeit erkannt und wertgeschätzt wurde. Das hat wiederum dazu geführt meine Projekte erfolgreich durchzuführen und die Ergebnisse für diese Dissertation wiederzuverwenden. Ich danke auch meinen aktuellen Vorgesetzten Peter Kalthoff, der mir bei der Finalisierung dieser Dissertation den notwendigen Freiraum eingeräumt hat.

Weiterhin danke ich den (ehemaligen) Kolleginnen und Kollegen des s-lab – Software Quality Lab und des Lehrstuhls für Datenbanken und Informationsysteme. Allen voran meinen Bürokolleginnen und Kollegen Dr. Baris Güldali, Yavuz Sancar, Dr. Andreas Wübbeke, Dr. Michael Mlynarski und Zille Huma. Durch euch war ein tolles Arbeitsklima gewährleistet. Ihr hattet auch immer ein Ohr für mich, falls es mal nicht mit großen Schritten voran ging.

Mein allergrößter Dank aber gilt meiner Ehefrau Verena, die es in den letzten beiden Jahren wohl am schwierigsten mit mir hatte. Du hast mich vor allem in der Phase des Aufschreibens immer wieder motiviert und mir die dir möglichen Alltagsaufgaben abgenommen. Vielen lieben Dank dafür.

Zum Schluss möchte ich meinen Eltern dafür danken, dass sie mich Zeit meines Lebens unterstützt und gefördert haben. Ihr habt mir stets Vertrauen entgegen gebracht und mir immer die Freiräume gelassen, die notwendig waren.

Zusammenfassung

Software-Produkte und -Systeme werden immer komplexer. Damit ihre Entwicklung beherrschbar und kontrollierbar bleibt, wird es stets wichtiger geeignete Entwicklungsmethoden zu nutzen. Es gibt bereits standardisierte Vorgehensmodelle, die einen strukturierten Entwicklungsprozess vorgeben. Ebenso sind für die verschiedensten Entwicklungsdomänen die speziellen Entwicklungsartefakte bekannt und können über domänenspezifische Sprachen beschrieben werden. Das Forschungsgebiet der Methodenentwicklung kombiniert diese Eigenschaften und ist in der Lage domänenspezifische Entwicklungsmethoden zu erstellen.

Die Einführung domänenspezifischer Entwicklungsmethoden in Unternehmen birgt Risiken. Der Aufwand, der aus einer kompletten Ersetzung einer bestehenden Entwicklungsmethode hervorgeht, kann sehr umfangreich sein und dadurch sind die Investitionskosten schlecht planbar. Darüber hinaus sind die erwarteten Vorteile mit Unsicherheiten behaftet, denn jedes Unternehmen besitzt schlecht änderbare, individuelle Eigenschaften, die so genannten Situationsfaktoren. Sie beschreiben den Methodenkontext, in dem die Entwicklungsmethode angewendet wird. Passt die domänenspezifische Entwicklungsmethode nicht zu dem gegebenen Methodenkontext, ist eine erfolgreiche Einführung gefährdet.

Zur Reduzierung der genannten Risiken erweitert diese Arbeit den bestehenden Ansatz MetaMe, eine Meta-Methode zur Entwicklung von individuellen Softwareentwicklungsmethoden, um die Eigenschaften der situationsgerechten Methodenweiterentwicklung. Eine zusätzliche Ist-Analyse identifiziert die vorhandene Entwicklungsmethode, die mittels einer neuen Sprache modellbasiert dokumentiert wird. Eine Verbesserungs-Analyse ermittelt darauf aufbauend die Optimierungspotenziale und Situationsfaktoren, aus denen geeignete Methodenanforderungen erstellt werden. Mit Hilfe der Methodenanforderungen werden iterative Projekte zur Methodenweiterentwicklung definiert. Das ermöglicht die von Unternehmen gewünschte schrittweise Änderung der Entwicklungsmethoden. Weiterhin wird gezeigt, wie die Methode mittels Methodenanforderungen an den Methodenkontext angepasst werden kann.

Das in dieser Arbeit vorgestellte Vorgehen wird exemplarisch auf zwei industrielle Anwendungsbeispiele aus dem Bereich der Server-System-Entwicklung angewendet. Für jedes Beispiel wird eine individuell entwickelte Entwicklungsmethode vorgestellt.

Abstract

Software-Products and -Systems become more complex. To master and control the development it is important to use a suitable engineering method. There are several standardized process models available that are structuring the development tasks. Beside that, in several development domains the special artifacts are known and can described with domain-specific languages. In the research area method engineering these circumstances are combined. As a result, different domain-specific engineering methods can be created.

It is risky to introduce domain-specific engineering methods to different companies. The effort to replace an existing engineering method can be extensive so that investment costs are hard to judge. Beside that, the expected advantages have uncertainties because each company has poorly changeable, individual properties, so called situational factors. They describe the method context in that the engineering method will be applied. If the engineering method does not fit to the method context, a successful introduction is jeopardized.

To reduce the mentioned risks, this thesis enhances the known approach, MetaMe, a meta-method to develop individual software engineering methods, regarding the characteristics of situational method enhancements. It adds the analysis of the current state to identify the existing engineering method. The engineering method will be described in a model-based manner with a newly introduced language. A demand analysis identifies single improvements and situational factors. Upon that information method requirements are created. Based on the method requirements iterative method enhancement projects are defined. With that it is possible to do a step wise change of engineering methods, which is the desired approach of the companies. Furthermore, it will be shown how to adjust the engineering method according to the method context based on the method requirements.

The new method engineering approach is substantiated by two industrial case studies in the area of server-systems-engineering. For each example an individually developed engineering method will be presented.

Inhaltsverzeichnis

1 Einleitung

(Software-) Systeme werden komplexer

Heutzutage werden (Software-)Systeme immer komplexer. Wie in [NFG+06] dargestellt wird, erhöht sich die Anzahl der Programmzeilen, der an dem Entwicklungsprozess beteiligten Personen, der zu speichernden, zugreifbaren und manipulierbaren Daten, der Verbindungen und Abhängigkeiten zwischen Softwarekomponenten und der beteiligten Hardware-Elemente im Vergleich zu heutigen Systemen. Diese Entwicklung wird auch bei Business- und Informationssystemen sowie bei eingebetteten Systemen beobachtet. Damit die Erstellung der (Software-)Systeme beherrschbar, kontrollierbar und wiederholbar bleibt, wird die Etablierung von geeigneten Entwicklungsmethoden stets wichtiger.

Strukturierung des Entwicklungsprozesses

Entwicklungsansätze wie modellgetriebene Entwicklung (Model Driven Engineering, MDE) und standardisierte Vorgehensmodelle wie der Rational Unified Process (RUP) und V-Modell XT, neben den agilen Repräsentanten SCRUM und Extreme Programming (XP), versuchen die Entwicklungsprozesse zu strukturieren. Sie helfen bei der praktischen Durchführung von Entwicklungsvorhaben [Sch06, JBR99, FHKS09, SB01, Bec99].

Qualitätssicherung durch Entwicklungsmethoden

Bei der täglichen Arbeit im s-lab – Software Quality Lab[1] werden Unternehmen bei der Erhöhung der Qualität ihrer Softwareprodukte unterstützt. In den meisten Fällen werden Verbesserungspotenziale in der Spezifikation und in der Entwicklungsmethode zur Erstellung der Spezifikationen festgestellt. Die Aufbesserung der Entwicklungsmethode und somit der Spezifikation bietet die Möglichkeit die analytische und konstruktive Qualitätssicherung voranzutreiben.

Standardisierte Vorgehensmodelle sind nicht anwendbar

Eigenschaften der Unternehmen, der zu entwickelnden Produkte sowie der s-lab – Software Quality Lab-Projekte verhindern die Einführung standardisierter Vorgehensmodelle. Erstens wird nicht gewünscht einen vorhandenen Entwicklungsprozess durch eine allumfassende Methode komplett neu zu strukturieren,

1 http://www.s-lab.de, letzter Aufruf: 2015-01

denn, wenn auch nicht dokumentiert, liegt bereits eine implizite Entwicklungsmethode vor. Zweitens stellen allgemeine Entwicklungsprozesse keine Lösung für spezielle zu berücksichtigende Produktcharakteristiken dar. Drittens sind Umfang und Ziel der Projekte nicht auf solch allumfassende Änderungen ausgelegt. Punktuelle, individuelle Lösungen für die akuten Probleme werden hier bevorzugt. Know-how-Aufbau beim Unternehmen und die Nachhaltigkeit der erwirkten Optimierungen sind weitere Ziele.

Entwicklungsmethode für die Server-System-Entwicklung

Diese Arbeit wird motiviert durch ein konkretes Forschungs- und Entwicklungsprojektaus dem s-lab – Software Quality Labin Kooperation mit Fujitsu Technology Solutions GmbH (FTS)[2] im Bereich der Server-System-Entwicklung. Neben mechanischen und elektrotechnischen Produktanteilen gibt es Softwareanteile, die die Verwaltung und Konfiguration der Server-Systeme über Managementschnittstellen ermöglichen. Die Erhöhung der Qualität der Management-Funktionalität soll durch eine individuelle, auf den Projektpartner angepasste, Methodenweiterentwicklung erreicht werden.

1.1 Verbesserungspotenziale bei der Methodenweiterentwicklung

Methoden und Sprachen

Wie in der Einleitung erwähnt, trägt eine strukturierte Entwicklungsmethode sowie die Nutzung formalisierter Sprachen bei der Erstellung des Produkts dazu bei, die analytische und konstruktive Qualitätssicherung voranzutreiben.

Entwicklungsmethode

Methoden sind planmäßig angewandte, begründete Vorgehensweisen zur Erreichung von festgelegten Zielen ([Bal09], S. 53). Eine strukturierte Entwicklungsmethode hat die Aufgabe bei der Entwicklung zu unterstützen, indem der Erstellungsprozess eines Produkts planbar, kontrollierbar und wiederholbar ist.

Entwicklungssprachen

Formalisierte (Entwicklungs-)Sprachen basieren auf einer formalisierten Syntax und Semantik und werden dadurch eindeutig. Sie bieten eine bessere Lesbarkeit und Verständlichkeit. Verschiedenste Ausprägungen von rein textuellen bis hin zu ausschließlich grafischen Sprachelementen sind möglich. Entwicklungssprachen können eine hohe Problemspezifität hervorbringen. Solche spezifischen Sprachen werden auch Domänenspezifische Sprachen (Domain Specific Language

2 http://www.fujitsu.com/de/about/fts/, letzter Aufruf: 2015-01

(DSL)) genannt. DSLs gelten in ihrem Einsatzgebiet leicht verständlicher als allgemein gültige Sprachen. Weiterhin bieten formalisierte Sprachen die Möglichkeit der Standardisierung und maschinellen Verarbeitung an. Sie sind die Grundlage für die modellgetriebene Entwicklung mit dem Vorteil der hohen Automatisierung des Entwicklungsprozesses und ermöglichen so konstruktive Qualitätssicherungsmaßnahmen. Die Abbildung 1.1 stellt die zwei in dieser Arbeit zu betrachtenden

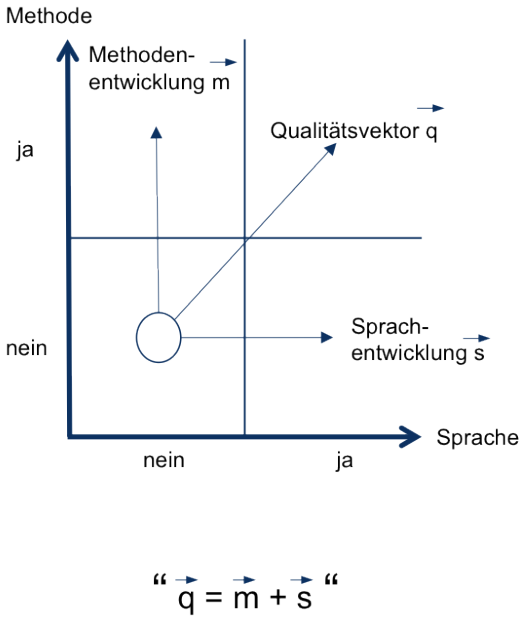


Abbildung 1.1: *Optimierungspotenziale der Entwicklung*

Optimierungspotenziale

Verbesserungspotenziale dar, das Vorgehen der Entwicklung über eine Entwicklungsmethode zu strukturieren und die Nutzung formalisierter Entwicklungssprachen. Der vertikale Pfeil repräsentiert das Forschungsgebiet der Methodenentwicklung, der horizontale Pfeil symbolisiert das Forschungsgebiet der Sprachentwicklung. Als Qualitätssteigerung wird hier jede Verbesserung, entweder durch die Sprach- oder Methoden-Entwicklung, angesehen. Die gesamte Verbesserung kann durch den diagonalen Qualtitätsvektor dargestellt werden.

kombinierter Ansatz

Es gibt den von [Sau11b] definierten Ansatz zur Entwicklung von Software-Entwicklungsmethoden, welcher auf dem Konzept der Meta-Modellierung [AK03] basiert. Die dort beschriebene Methode mit dem Namen MetaMe berücksichtigt explizit die Sprachauswahl und kombiniert dadurch die beiden Verbesserungspotenziale, Methoden- und Sprachentwicklung. Das geschieht, indem durch die Ausführung von MetaMe ein Prozessmodell und Produktmodell der Software-

Entwicklungsmethode definiert wird. Das Prozessmodell stellt die ausführbaren Aktivitäten und das Produktmodell den Zusammenhang der Artefakttypen der Software-Entwicklungsmethode dar. Ein Artefakttyp ist die Typdefinition eines Artefakts, welches bei der Ausführung der Software-Entwicklungsmethode als Zwischen- oder Endprodukt der Spezifikation erstellt wird. In der Softwareentwicklung wird das auch Instanziierung genannt. Unter dem Begriff Spezifikation werden alle zur Entwicklung notwendigen Zwischen- oder Endprodukte gezählt. Das können, abhängig vom Abstraktionsgrad, die unterschiedlichsten Artefakte sein, beispielsweise Dokumente, Modelle aber auch Programmcodes. Bei der Entwicklung von DSLs gibt es ein Sprachmetamodell, welches bei der Anwendung der Sprache instanziiert wird. Bei der Kombination der Methoden und Sprachentwicklung wird das Produktmodell als Sprachmetamodell genutzt. Dieser Zusammenhang wird in Abbildung 1.2 vorgestellt. Die Softwareentwicklung benutzt le-

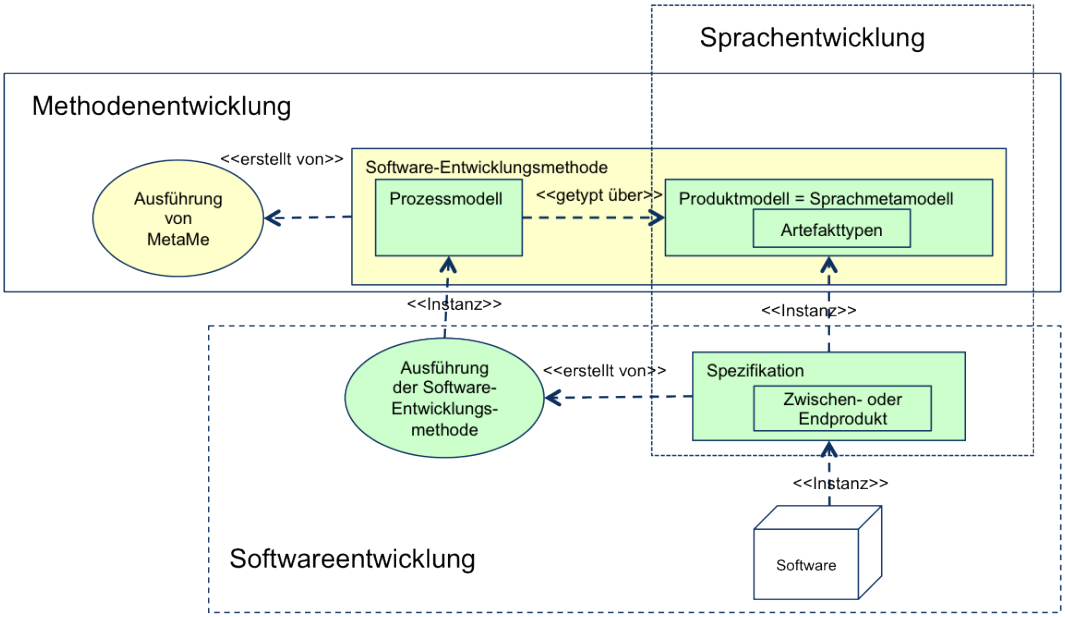


Abbildung 1.2: *Kombinierte Sprach- und Methodenentwicklung*

SWE

diglich die Softwareentwicklungsmethode sowie die Sprachen zur Spezifikation der Software, insbesondere Programmiersprachen.

1.2 Problemstellung bei der Methodenweiterentwicklung

Herausforderungen für den Methodenentwickler

Die Methodenweiterentwicklung wird von der Rolle Methodenentwickler durchgeführt. In der Praxis ist der Methodenentwickler mehr oder weniger Experte in der Domäne der Methodenentwicklung. Er wird auf einige Herausforderungen treffen. Darum ist es sinnvoll den Methodenentwickler durch eine geeignete Methode zur Methodenweiterentwicklung zu unterstützen, die die im Folgenden beschriebenen Probleme auflöst.

Implizite Methoden

Es sei vorweggenommen, dass in dem Forschungs- und Entwicklungsprojektdie Server-System-Entwicklung keiner explizit festgelegten Methode folgt. Die Spezifizierung geschieht deshalb keinesfalls willkürlich, sondern auf Basis des Vorwissens der Entwickler. Das bedeutet, die Herangehensweise zur Spezifikation der Server-Systeme ist implizit vorhanden. In dieser Arbeit wird immer von einer (implizit) vorhandenen Ist-Methode ausgegangen. Eine erste Aufgabe ist das explizite Darlegen der implizit vorhandenen Entwicklungsmethode.

Ermittlung der Ist-Methode

Zur Ermittlung der (impliziten) Ist-Methode wird konkretes Wissen über tatsächlich durchgeführte Arbeiten der Systementwickler benötigt. Eine Zusammenarbeit vom Methodenentwickler mit den Systementwicklern ist erforderlich. Zum einen müssen Informationen vom Systementwickler abgefragt werden, die in einem weiteren Schritt vom Methodenentwickler konsolidiert, analysiert und dokumentiert werden. Zum anderen muss die dokumentierte Information über das methodische Vorgehen mit dem Systementwickler wieder abgeglichen werden. Deshalb muss das dokumentierte Methodenwissen in einer einfachen und verständlichen Form vorliegen.

Methodendarstellung

Für die Darstellung von Prozessen gibt es eine Reihe an standardisierten Prozessbeschreibungssprachen. Im Bereich der Methodenentwicklung ist das Software & Systems Process Engineering Meta-Model (SPEM, siehe [Obj08]) der bekannteste Repräsentant. SPEM beschreibt neben Prozessmodellelementen zusätzlich auch weitere Methodenelemente aus dem Produktmodell und verschiedene Hilfsmittel (vgl. Abbildung 1.3). SPEM dient jedoch lediglich zur Darstellung, aber nicht zur Entwicklung einer Methode. Vorgehen zur Ermittlung von Methoden nutzen oft eigene, schwer verständliche Sprachkonstrukte und lassen sich schwer auf standardisierte Sprachen wie, beispielsweise SPEM, abbilden. Die Integrati-

on von standardisierten Sprachkonstrukten für Methodenelemente und Vorgehen zur Ermittlung von Methoden ist momentan nicht vorhanden.

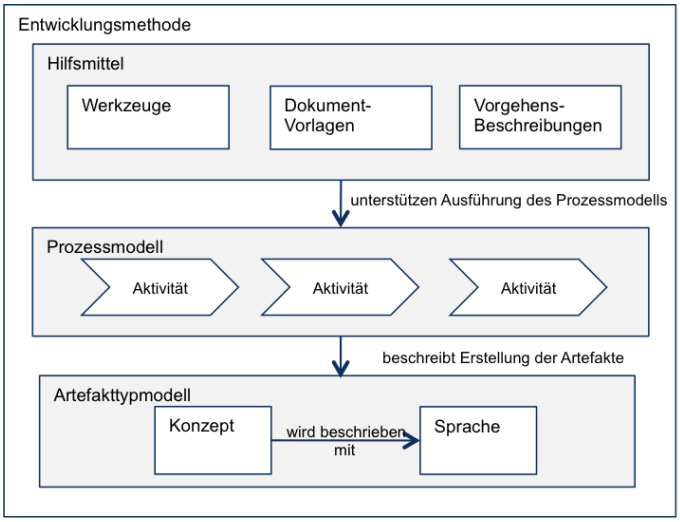


Abbildung 1.3: *Exemplarische Darstellung der Elemente einer Entwicklungsmethode*

Problemdefinition der Methodenentwicklung 1
Ermittlung und explizite Darstellung der Ist-Methode.

Schwachstellen

Die Aufgabe der Methodenentwicklung ist es, die Schwachstellen zu ermitteln und geeignete Maßnahmen mit einer Weiterentwicklung der Methode zu ergreifen. Vorausgesetzt wird dabei, dass der Methodenentwickler eine Übersicht über das Domänenwissen, der passenden einsetzbaren Konzepte und verfügbaren Sprachen zur Entwicklung und Spezifikation von Server-Systemen hat. Die Weiterentwicklung der Entwicklungsmethode zur Aufhebung vorkommender Schwächen in der Entwicklung ist ebenfalls das primäre Ziel im Forschungs- und Entwicklungsprojekt. Wir kennen die zwei Schwachstellen informelle Darstellung der Spezifikationsinhalte und fehlende Festlegung der Entwicklungsmethode. Im Forschungs- und Entwicklungsprojektbesteht die Spezifikation der Server-Systeme zum großen Teil aus natürlich sprachlicher, textueller Informationsdarstellung. Allseits bekannte Schwachpunkte der informellen Darstellung sind die Gefahren der Mehrdeutigkeit und Unvollständigkeit der Inhalte, die während der Entwicklung erstellt werden. Werden die Spezifikationsinhalte untersucht und können sogar mit auftretenden Problemen in der Entwicklung oder mit typischen Fehlern während der Qualitätssicherung in Verbindung gebracht werden, sind vorliegende Schwachstellen zu erkennen. Schwachpunkte, die durch die

Definition der Aktivitäten des Entwicklungsprozesses bedingt sind, liegen an den Abhängigkeiten der Aktivitäten untereinander. Diese Abhängigkeiten können nur bei umfassender expliziter Darstellung des Entwicklungsprozesses ermittelt werden. Umso mehr Aktivitäten explizit benannt sind, umso höher ist die Warscheinlichkeit Abhängigkeitsprobleme zu ermitteln. Das bedeutet, umso umfangreicher die Methode in expliziter Form vorliegt, umso besser können die Schwachpunkte ermittelt werden.

Verbesserungen

Der Methodenentwickler muss nun geeignete Maßnahmen zur Methodenverbesserung darstellen. Die Maßnahmen können als Anforderungen an die Soll-Methode, die durch eine Methodenweiterentwicklung erstellt werden soll, formuliert werden. Im Folgenden werden diese Anforderungen auch Methodenanforderungen genannt. Beispielsweise gibt das MetaMe-Prozessmodell die Erhebung von Methodenanforderungen vor (siehe Abbildung 1.4). Die MetaMe-Definition

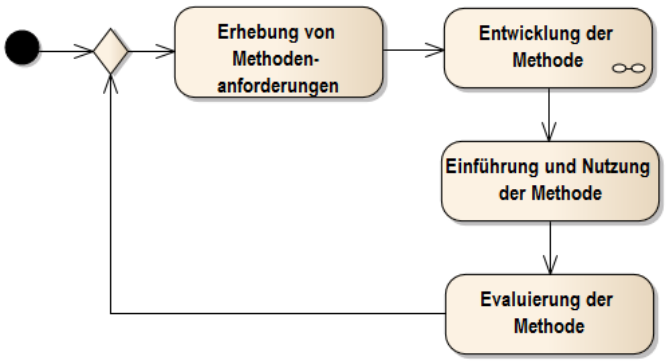


Abbildung 1.4: *Entwicklungsphasen von MetaMe*

Erhebung von Methodenanforderungen

geht allerdings nicht genauer auf die Erhebung und Darstellung von Methodenanforderungen ein. Methodenanforderungen werden in der Literatur oft als gegeben angenommen. In der Literatur gibt es jedoch keinen Ansatz zur Erhebung von Methodenanforderungen, dessen resultierende Anforderungen für die Weiterverarbeitung mit MetaMe geeignet sind.

Problemdefinition der Methodenentwicklung 2
Ermittlung und Darstellung von Methodenanforderungen als Ergebnis einer Verbesserungs-Analyse.

Eingeschränkter Zugriff auf Informationen

Der Methodenentwickler braucht für die Methodenweiterentwicklung Zugriff auf die verschiedensten Informationen. Eine Erfahrung aus dem Forschungs- und

Entwicklungsprojektist die eingeschränkte Einsicht in alle notwendigen Informationen für die Methodenweiterentwicklung. Die unterschiedlichen Gründe werden in der folgenden Auflistung beschrieben:

- Einarbeitungszeit: Der Methodenentwickler muss Informationen erheben und analysieren, um Schlussfolgerungen zu ziehen. Dafür muss Zeit eingeplant werden.
- Projektrahmen: Auftragvolumen und Projektlaufzeit schränken die Definition der Projekttätigkeiten ein. Bei geringer Laufzeit wird nur die Aufarbeitung spezieller Aspekte der Entwicklung möglich.
- Projektunterstützung: Bestimmte Entwicklungsteams oder einzelne Abteilungen suchen die Beratung. Die Unterstützung aller an dem gesamten Entwicklungsprozess beteiligten Akteure ist jedoch nicht gegeben.
- Vertraulichkeit: Der Einblick in Informationen wird aufgrund von sicherheitsrelevanten Aspekten und Betriebsgeheimnissen eingeschränkt.
- Dokumentation: Nicht immer sind alle notwendigen Informationen dokumentiert. Implizites, Domänen- und Betriebs-spezifisches Wissen steckt in den Köpfen der Mitarbeiter.
- Definition: Es fehlen Festlegungen wie Domänenmodelle oder Ontologien. Das Domänenwissen selbst ist nicht definiert. Unterschiedliche Vorstellungen und Definitionen liegen vor.

Projektrisiko

Neben dem eingeschränkten Zugriff auf die notwendigen Informationen birgt eine allumfassende Anpassung oder Etablierung einer neuen Entwicklungsmethode ein hohes, nicht kalkulierbares Projektrisiko.

schrittweise Einführung von Verbesserungen

Bedingt durch den eingeschränkten Zugriff auf Informationen und mögliches Projektrisiko sind schrittweise, kleine Verbesserungen gern gesehen. Überschaubare Ziele können schnell in laufenden Projekten begleitend eingeführt werden. Die Bereitstellung eines speziellen, eingeschränkten Zugangs zu notwendigen Informationen ist meistens zu bewerkstelligen. Weiterhin wird die Bewertung kleinerer Verbesserungen in Form von Kosten, Aufwand und Anwendbarkeit einfacher. Im erfolglosen Fall ist der Schaden gering. Im erfolgreichen Fall können weitere Verbesserungspotenziale betrachtet werden.

iterative Methodenentwicklung

Die Schwierigkeit für den Methodenentwickler ist es jedoch, eine Menge kleinerer Verbesserungen mit übergeordneten Zielen zu definieren und so umzusetzen, dass die Ergebnisse gut zusammenspielen. Die Gefahr von Insellösungen besteht. Das heißt, zwei unabhängig voneinander entwickelte Methodenweiterentwicklungen können sich bei der Integration der Teil-Methoden behindern. Ein rein inkrementeller Methodenentwicklungsansatz reicht nicht aus, da eine Verbesserung in einem ersten Schritt neue Voraussetzungen für die Methodenweiterentwicklung in einem zweiten Schritt bilden kann. Die Ergebnisse der vorangegangenen Methodenweiterentwicklung müssen in weiteren Schritten der Methodenweiterentwicklung berücksichtigt werden. Eine initiale Feedbackschleife ist bereits von MetaMe angedacht, indem eine Methodenevaluierung stattfindet und eine weitere Iteration der Methodenentwicklung durchgeführt wird (vgl. Abbildung 1.4). Die Definition der langfristigen Ziele wird momentan jedoch nicht betrachtet.

Problemdefinition der Methodenentwicklung 3
Definition einer inkrementellen Methodenentwicklung mit festgelegten Zielen

Rahmenbedingungen für die Entwicklungsmethode

Der Methodenentwickler hat bei der Methodenentwicklung stets Betriebs- und Projekt-spezifische Eigenschaften zu berücksichtigen [HSR10]. Diese Eigenschaften werden Situationsfaktoren genannt. Hierbei ist die Schwierigkeit festzustellen, welche Situationsfaktoren relevant für die Methodenentwicklung sind und noch wichtiger ist es zu verstehen, welche Auswirkungen sie auf die Entwicklungsmethodenerstellung haben. Zum einen können Inhalt und Struktur der Entwicklungsmethode betroffen sein, indem bestimmte Aktivitäten oder Artefakte gefordert werden. Zum anderen ist es eventuell notwendig, die Schrittweite einer gewollten Verbesserung anzupassen, da im Projekt gültige Situationsfaktoren das Erreichen der ursprünglich geplanten Verbesserung behindern. Beispiele aus dem Forschungs- und Entwicklungsprojektsind:

- Die Forderung nach bestimmten Dokumenten für die Management-Ebene.
- Die obligatorische Bildung interdisziplinärer Entscheidungsteams mit festgelegten Terminen.
- Fehlender Zugriff auf relevante Entwicklungswerkzeuge.

situationsgerechte Methode

Eine an Situationsfaktoren angepasste Methode wird situationsgerechte Methode genannt. Die Literatur gibt verschiedene Hinweise auf Situationsfaktoren. Allerdings gibt es keine konsolidierte Liste möglicher Situationsfaktoren und nur für

ein paar Situationsfaktoren sind sinnvolle Anpassungen der Entwicklungsmethode bekannt.

Problemdefinition der Methodenentwicklung 4
Erhebung und Berücksichtigung von Situationsfaktoren.

Nachvollziehbarkeit

Es gibt verschiedene Gründe für die Dokumentation des Methodendesigns. Nicht nur die Methode selber soll adäquat beschrieben werden, sondern die Erkenntnisse und Erfahrung, wie die entwickelte Methode erstellt worden ist, sind wichtig. Die Methodenentwicklung soll nachvollziehbar sein.

Konservierung des Vorgehens

Das Anwendungsbeispiel ist eine schrittweise Einführung von Optimierungen, die jeweils für sich als (Teil-)Entwicklungsmethoden zu verstehen sind. Es werden demnach immer Entwicklungsmethoden erstellt, die Teil einer übergeordneten Methoden sein können. Gibt es bereits Entwicklungsmethoden als Teil einer Gesamtmethode, entsteht eine Integrationsproblematik. Damit diese Integration möglich ist, müssen frühere Designentscheidungen nachhaltig verstanden werden, da zum einen eine geringe zeitliche Abfolge der Optimierungsschritte und zum anderen die Garantie denselben Methodenentwickler verpflichten zu können, nicht gegeben sein muss. Der Methodenentwickler muss daran interessiert sein, Wissen über verschiedene Betriebsspezifika und der unterschiedlichen Entwicklungsdomänen zu konservieren, um diese gegebenenfalls wiederverwenden und weitergeben zu können. Darüber hinaus wird die Dokumentation benutzt, aktuelle Designentscheidungen zu begründen.

Analyse der Methodenentwicklung

Im Hinblick auf die Problemdefinitionen der Methodenentwicklung 4 besteht die Möglichkeit, auf der Basis der Dokumentation der Methodenerstellung, nachträglich Auswirkungen von Betriebsspezifika auf das Methodendesign ableiten zu können.

ganzheitliche Sicht

Das Problem ist vergleichbar mit der Darstellung der Ist-Methode. Die Weiterentwicklung resultiert in einer Soll-Methode, die vergleichbar zur Ist-Methode explizit dargestellt werden soll. Allerdings gibt es weitere Informationen, wie beispielsweise die gültigen Situationsfaktoren, Methodenanforderungen oder aber die Festlegung der Iterationen mit ihren übergeordneten Zielen, die bei der Methodenentwicklung dokumentiert werden sollen.

Problemdefinition der Methodenentwicklung 5
Dokumentation der Methodenentwicklung

Die Summe der Problemdefinitionen der Methodenentwicklung 1-5 führt letztendlich zur konkreten Aufgabenstellung.

Aufgabe der Dissertation
Die Entwicklung einer Methode zur Methodenweiterentwicklung auf Basis von MetaMe. Die Anforderungen der problemspezifischen, iterativen und situationsgerechten Methodenentwicklung sollen unterstützt werden. Die Methode soll geeignete Sprachen zur Dokumentation von zu entwickelnden problemspezifischen Methoden und der Methodenentwicklung definieren.

Für die Aufgabenstellung wird in Abschnitt 1.3 ein möglicher Lösungsansatz vorgestellt.

1.3 Lösungsansatz

Methoden-weiterentwicklung

Der Lösungsansatz der im vorangegangenen Abschnitt vorgestellten Probleme wird in diesem Abschnitt vorgestellt. Die Abbildung 1.5 stellt das Gesamtvorgehen der Methodenweiterentwicklung vor.

Ist-Analyse

Eingangsinformation ist die vorhandene und eventuell nur implizit vorliegende Ist-Methode. Im Sinne der modellbasierten Entwicklung wird die Ist-Methode in einem ersten Schritt modellbasiert beschrieben. Dadurch wird die Ist-Methode explizit dargestellt, mit der Möglichkeit von nicht relevanten Informationen zu abstrahieren. Das führt auch zur Lösung der Problemdefinition der Methodenentwicklung 1. Neben den Informationen über die Methode selber müssen an dieser Stelle die Menge der gültigen Situationsfaktoren, also der Methodenkontext im konkreten Methodenweiterentwicklungsprojekt ermittelt werden. Ist-Methode und Methodenkontext bilden dann den modellbasiert definierten Ist-Zustand.

Verbesserungs-Analyse

Der Ist-Zustand bildet die Eingangsinformationen für die sich anschließende Phase *Verbesserungs-Analyse*. Die Verbesserungs-Analyse soll die Verbesserungspotenziale ermitteln. Zwei grobe Verbesserungspotenziale, Formalisierung der Spezifikationsinhalte und Detaillierung und Erweiterung der expliziten Methode,

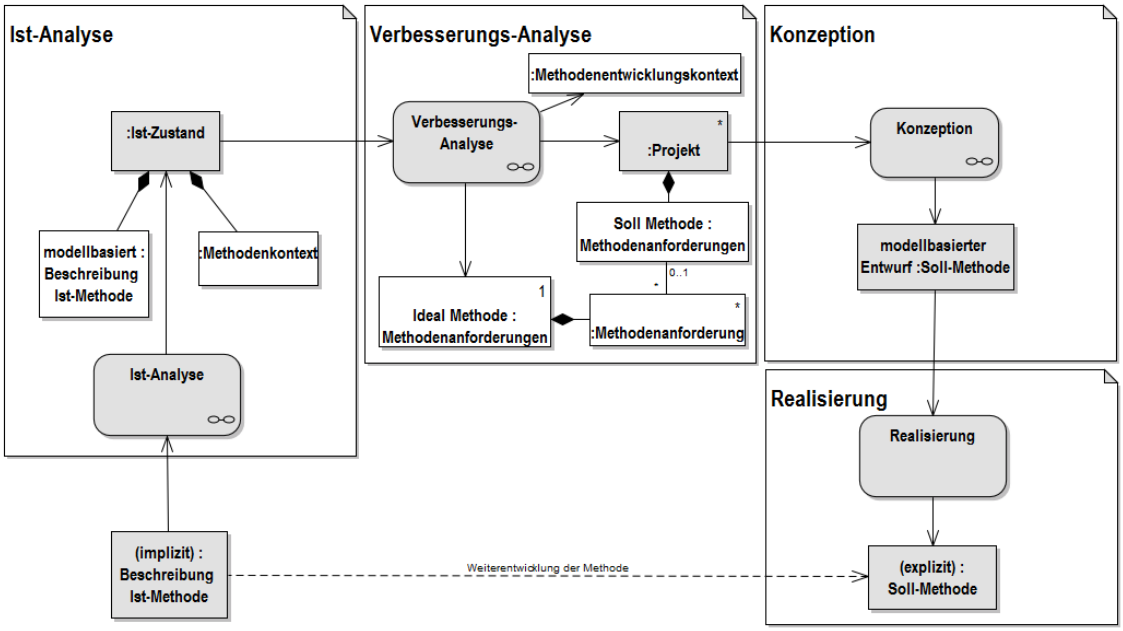


Abbildung 1.5: *Übersicht über den Lösungsansatz*

sind bekannt und sollen über Methodenanforderungen im Hinblick auf die konkreten Schwachstellen der Methode verfeinert werden. Das soll die Problemdefinition der Methodenentwicklung 2 lösen. Eine Menge von Methodenanforderungen beschreiben das langfristig zu erreichende Ziel einer Ideal-Methode. Wie in Abschnitt 1.2 beschrieben, können Einschränkungen des Methodenweiterentwicklungsprojekts vorliegen. Projektrisiko, Projektrahmen, Projektunterstützung und weitere Eigenschaften bilden den Methodenentwicklungskontext. Auf Basis des Methodenentwicklungskontexts können die Methodenanforderungen analysiert und in inkrementelle Teilmengen zerlegt werden. Die Anforderungen gruppiert in inkrementellen Teilmengen bilden inkrementelle Teil-Projekte und beschreiben letztendlich inkrementell zu entwickelnde Soll-Methoden. Aufgrund der erhobenen Gesamtmenge an Methodenanforderungen soll sichergestellt werden, dass eine Soll-Methode auf einem relevanten Migrationspfad zur Ideal-Methode liegt (vgl. Problemdefinition der Methodenentwicklung 3). Zur Lösung der Problemdefinition der Methodenentwicklung 4 soll das Vorgehen zur Erhebung von Methodenanforderungen die gültigen Situationsfaktoren berücksichtigen und situative Methodenanforderungen hervorbringen. Die situativen Methodenanforderungen sollen, wie die Methodenanforderungen auf Basis der Schwachstellen, in die Methodenweiterentwicklung einfließen.

Konzeption

Für ein definiertes Teil-Projekt mit den dazugehörigen Methodenanforderungen wird in der Phase *Konzeption* die Ist-Methode unter Berücksichtigung der Methodenanforderungen in ein Entwurfsmodell der Soll-Methode überführt. Diese Aufgabe wird durch das bestehende MetaMe gelöst.

Realisierung

Das Entwurfsmodell der Soll-Methode ist noch keine ausführbare Methode. Dafür fehlt es an konkreten Hilfsmitteln und Handlungsanweisungen für die Durchführung der Methode. Dafür schließt sich die Phase *Realisierung* an, die letztendlich die Methode scharf schaltet.

MetaMe++

Der Übergang von einer Ist-Methode zu einer Soll-Methode im Sinne einer Methodenweiterentwicklung wird dann über die Phasen *Ist-Analyse*, *Verbesserungs-Analyse*, *Konzeption* und *Realisierung* durchgeführt werden. MetaMe wird mit den Anforderungen der Methodenweiterentwicklung und Selbstanwendung der Methodenweiterentwicklung nach MetaMe++ weiterentwickelt. MetaMe++ wird streng nach dem in Abbildung 1.2 vorgestellten Konzept der integrierten Methoden- und Sprachenentwicklung entworfen und stellt somit die notwendige Grundlage zur Sprachdefinition für die Dokumentation der Methode und Methodenentwicklung bereit, was zur Lösung der Problemdefinition der Methodenentwicklung 5 beiträgt.

Hypothese

Der Lösungsansatz MetaMe++ führt zur folgenden Hypothese, die in dieser Dissertation zu überprüfen ist.

Hypothese
Mit MetaMe++ ist eine Methodenweiterentwicklung durchführbar.

Überprüfung

Zur Überprüfung der Hypothese werden zwei Anwendungsfälle aus Forschungs- und Entwicklungsprojekten vorgestellt, die im s-lab – Software Quality Labin Kooperation mit der Firma Fujitsu Technology Solutions GmbH durchgeführt worden sind.

1.4 Aufbau der Arbeit

Einleitung

In diesem einleitenden Kapitel ist die Motivation für eine Methodenweiterentwick-

lung mit geeigneten Verbesserungspotenzialen vorgestellt worden. Für die Aufgabe der Methodenweiterentwicklung mit ihren Problemdefinitionen ist als Lösung ein Vorgehen vorgeschlagen worden, welches durch die Weiterentwicklung von MetaMe nach MetaMe++ umgesetzt wird.

Kapitel 2 Das Kapitel 2 fasst die Grundlagen der Methodenentwicklung zusammen. Zuerst werden grundsätzliche Begriffe erläutert und daraus ein grundlegendes Metamodell für Entwicklungsmethoden abgeleitet. Darauf aufbauend wird das dieser Arbeit zugrunde liegende Vorgehen der Methodenweiterentwicklung vorgestellt. Die bisher genutzte Metamethode zur Erstellung von Softwareentwicklungsmethoden MetaMe wird diskutiert. Zur Integration von MetaMe in das Konzept der Methodenweiterentwicklung ist es notwendig das Metamodell der Entwicklungsmethoden mit dem Metamodell von MetaMe zu verschmelzen und die Anforderungen an ein Metamodell zur Nutzung als abstrakte Syntax für eine Sprache zur Darstellung von Methoden zu berücksichtigen. Aus diesen Erkenntnissen wird im ersten Lösungsbaustein ein linguistisches Metamodell entwickelt. Mit Hilfe des linguistischen Metamodells wird eine Sprache zur Beschreibung von Entwicklungsmethoden definiert.

Kapitel 3 Das Kapitel 3 beschreibt die Grundlagen für die Server-System-Entwicklung. Das Themengebiet der Server-System-Entwicklung dient als Anwendungs- und Evaluationsbeispiel für die Anwendung der Methodenweiterentwicklung. Diese Grundlagen beschreiben den aktuellen Stand der Technik, auf dessen Informationsbasis eine Ist-Methode ermittelt werden muss. Das Vorgehen zur Ermittlung der Ist-Methode wird als Lösungsbaustein formalisiert. Basierend auf zwei realen Anwendungsfällen wird jeweils die extrahierte Ist-Methode als Ergebnis der Anwendung des Lösungsbausteins vorgestellt. Das erste Anwendungsbeispiel beschreibt allgemein die Spezifikation von Server-System-Architekturen, das zweite Anwendungsbeispiel die Entwicklung einer speziellen Verwaltungsschnittstelle für Server-Systeme.

Kapitel 4 Eine weitere Aufgabe in der Phase *Ist-Analyse* der Methodenweiterentwicklung ist die Beschreibung des Methodenkontexts, was vergleichbar mit einer Situationsbeschreibung ist, welche in dem Forschungsgebiet des Situational Method Engineering genutzt wird. Auf dem Konzept Situationsbeschreibung aufbauend wird in Kapitel 4 eine Checkliste möglicher Situationsfaktoren vorgestellt. Die Checkliste wird anschließend genutzt, um gültige Situationsfaktoren zu ermitteln, die eine konkrete Situation beschreiben. Das Vorgehen der Ermittlung des Metho-

denkontexts wird formalisiert und stellt einen weiteren Lösungsbaustein dar. Für die beiden Anwendungsfälle aus dem Forschungs- und Entwicklungsprojektwird jeweils exemplarisch der Methodenkontext bestimmt.

Beim Vorgehen der Methodenweiterentwicklung schließt sich der Phase *Ist-Analyse* die Phase *Verbesserungs-Analyse* an. In dieser Phase besteht die Hauptaufgabe in der Ermittlung von Methodenanforderungen. Kapitel 5 beschreibt die Entwicklung eines Vorgehens zur Ermittlung von Methodenanforderungen und berücksichtigt dabei auch situative Methodenanforderungen, damit die weiterentwickelte Methode an die jeweils vorliegende Situation, definiert über den Methodenkontext, angepasst werden kann. Das Kapitel definiert den Lösungsbaustein *Verbesserungs-Analyse* und endet mit der Beschreibung der Methodenanforderungen für die Anwendungsbeispiele. Kapitel 5

Nach der *Verbesserungs-Analyse* schließt sich die Phase *Konzeption* der Soll-Methode an, die mittels dem von MetaMe vorgegeben Konzept realisiert werden soll. Diese wird im Kapitel 6 mit den herausgearbeiteten Erkenntnissen dieser Arbeit als Lösungsbaustein definiert. Konsequenterweise findet dort ebenfalls die Beschreibung der Anwendung für die Anwendungsbeispiele statt. Damit die Informationen der *Ist-* und *Verbesserungs-Analyse* vom MetaMe-Konzept genutzt werden können, wird anschließend MetaMe zu MetaMe++ weiterentwickelt, indem die bisherigen Lösungsbausteine als ganzheitliches Vorgehen zusammengefasst werden. Ein zentrales Konzept in MetaMe ist die kombinierte Methoden- und Sprachentwicklung. Auf sich selbst angewendet, also unter Annahme, dass MetaMe++ auf sich selbst angewendet werden kann, ist es notwendig geeignete Sprachen zur Dokumentation der Methodenweiterentwicklung zu definieren. Die Sprachfestlegungen schließen das Kapitel ab. Kapitel 6

Für die beiden Anwendungsfälle aus dem Forschungs- und Entwicklungsprojektist in den Kapiteln 3, 4, 5 und 6 die Methodenweiterentwicklung bis hin zur Festlegung der Soll-Methode fortgeschritten. In dem Kapitel 7 werden die individuellen Arbeiten zur Realisierung der weiterentwickelten Methoden vorgestellt. Das Kapitel dient der Verteidigung der entwickelten Gesamtlösung der Methodenweiterentwicklung mit MetaMe++. Es stellt einige Aspekte der weiterentwickelten Methoden aus den Anwendungsbeispielen vor. Kapitel 7

Die Arbeit endet mit der Zusammenfassung und dem Ausblick im Kapitel 8. Kapitel 8

2 Methodenweiterentwicklung

Lösungsidee

Für die Methodenweiterentwicklung wird in Abschnitt 1.3 ein Vorgehen aus den vier Phasen Ist-Analyse, Verbesserungs-Analyse, Konzeption und Realisierung vorgeschlagen. Zur Durchführung des Vorgehens muss dieses weiter detailliert werden, so dass ein Methodenentwickler genaue Vorgaben zur Abarbeitung der einzelnen Phasen erhält.

Entwicklungsmethoden

Bei der Ist-Analyse wird die modellbasierte Erhebung der Ist-Methode beschrieben. Dafür muss der Begriff der Entwicklungsmethode definiert werden, denn in der Literatur über Methodenentwicklung (Method Engineering) gibt es unterschiedliche Vorstellungen und Metamodelle von Methoden. Insbesondere die synonyme Benutzung der Begriffe Vorgehen, Prozess, Methode und Methodik führt zu Unverständlichkeiten. In dieser Arbeit soll das von MetaMe definierte Verständnis einer Entwicklungsmethode in den Vordergrund gestellt werden.

Methodenentwicklung

Die Begrifflichkeiten im Forschungsgebiet Methodenentwicklung sind vielfältig. Dort gibt es Ansätze zur Methodenkonstruktion, Methodenanpassung, Methodenkonfiguration und situationsgerechter Methodenentwicklung (Situational Method Engineering). Eine Eingliederung und Abgrenzung des Begriffs *Methodenweiterentwicklung* in das Forschungsgebiet Methodenentwicklung muss durchgeführt werden, damit der Methodenentwickler ein besseres Verständnis der angedachten Aufgaben erhält.

Aufgaben des Methodenentwicklers

Nachdem ein grundlegendes Verständnis über die Methodenweiterentwicklung vorhanden ist, wird das Vorgehen der Methodenweiterentwicklung detailliert. Das führt zur folgenden Aufgabe.

Teilaufgabe der Dissertation 1
Detaillierung des Vorgehens der Methodenweiterentwicklung, so dass ein Methodenentwickler das Vorgehen versteht und durchführen kann.

modellbasierte Darstellung einer Methode

Eine erste Aufgabe des Methodenentwicklers ist die modellbasierte Beschreibung der Ist-Methode. Dafür wird eine geeignete Sprache benötigt. Die Sprache muss aus einer abstrakten Syntax in Form eines Metamodells, dessen Instanziierung das Modell einer Entwicklungsmethode bildet, einer konkreten Syntax zur Darstellung der Modellinhalte, also der grafischen Beschreibung der Methode, und einer Semantikbeschreibung bestehen. SPEM ist der einzige standardisierte Repräsentant einer solchen Sprache. Das Metamodell von SPEM unterscheidet sich aber im Detail von dem Metamodell von MetaMe. Einzelne Sprachkonstrukte von SPEM sind nicht in MetaMe vorgesehen und deshalb bestehen Probleme eine über SPEM beschriebene Ist-Methode später während der Konzeption mit MetaMe in ein Entwurfsmodell der Soll-Methode zu überführen. Eine zu MetaMe passende Methodenbeschreibungssprache wird benötigt.

Teilaufgabe der Dissertation 2
Definition einer MetaMe-kompatiblen Methodenbeschreibungssprache.

Struktur des Kapitels

Zur Lösung der in diesem Kapitel vorgestellten Aufgaben wird zunächst im Abschnitt 2.1 eine Festlegung der in dieser Arbeit benutzten Begrifflichkeiten gemacht. Weiterhin wird der Ansatz der Methodenweiterentwicklung in das Forschungsgebiet Methodenentwicklung eingeführt (siehe Abschnitt 2.2). Nach den Grundlagen sind die allgemeinen Tätigkeiten des Methodenentwicklers als Lösung der Teilaufgabe der Aufgabenstellung der Dissertation 1 *Detaillierung des Vorgehens der Methodenweiterentwicklung, so dass ein Methodenentwickler das Vorgehen versteht und durchführen kann* zu definieren, was in Abschnitt 2.3 geschieht. Eine Einführung in die der Arbeit zugrunde liegenden Metamethode zur Erstellung von Softwareentwicklungsmethoden MetaMe und deren Verbesserungspotenzial wird im Abschnitt 2.4 gegeben. Zum einen, da der MetaMe-Ansatz um Situationsbestimmung angepasst werden soll, aber auch, da von der Entwicklung von Softwareentwicklungsmethoden verallgemeinert werden soll. Anschließend wird vorgestellt, dass das Metamodell von Entwicklungsmethoden, welches von MetaMe bereitgestellt wird, durchaus alle bei der Methodenweiterentwicklung relevanten Methodenelemente definiert. Die Detaillierung des Metamodells ermöglicht die Definition einer abstrakten Syntax und eine linguistische Instanziierung des Metamodells wird ermöglicht. Dadurch wird die Teilaufgabe der Aufgabenstellung der Dissertation 2 gelöst. Die Entwicklung und Festlegung der Sprache MetaMe-Methodenbeschreibung (MMMB) ist das Thema des Abschnitts 2.5. Dieses Kapitel endet mit einer Zusammenfassung in Abschnitt 2.6, bei der die

aus diesem Kapitel resultierenden Anforderungen an die Weiterentwicklung von MetaMe nach MetaMe++ vorgestellt werden.

2.1 Entwicklungsmethoden

Bei der Erstellung von Entwicklungsmethoden muss der Methodenentwickler die einzelnen Elemente einer Entwicklungsmethode definieren. Zuerst wird der Begriff *Entwicklungsmethode* definiert und danach die einzelnen zu berücksichtigenden Elemente vorgestellt.

Begriffsabgrenzung Methode

Zur Vermeidung von Missverständnissen wird die Bedeutung des Begriffs *Entwicklungsmethode* von dem Begriff *Methode* abgeleitet, als die *planmäßig angewandte, begründete Vorgehensweise zur Erreichung von festgelegten Zielen*, wie Balzert in [Bal09] definiert. Der verwandte Begriff *Entwicklungsmethodik* umfasst gemäß der Definition des Begriffs *Methodik* aus dem Duden[1] *die Wissenschaft der angewandten Methoden*, aber auch die *festgelegte Art des Vorgehens*. Wobei sprachlich gesehen das Wort *Vorgehen*[2] als Synonym der Worte *Verfahren*[3] und *Methode* angesehen werden kann. Eine Übersetzung des Wortes *Methodik* in die englische Sprache würde korrekt zu *methodology* führen, welches fälschlicherweise als *Methodologie*[4] in die deutsche Sprache zurück übersetzt werden könnte und dann als *Lehre und Theorie der wissenschaftlichen Methoden* verstanden werden kann. Demnach führt der Begriff *Entwicklungsmethodik* zu Missverständnissen und wird in dieser Arbeit nicht benutzt. Eine ähnliche Diskussion wird in der Domäne des Situational Method Engineering geführt mit dem Ergebnis der Möglichkeit, Methode und Methodik als synonyme Begriffe zu benutzen [HSR10].

Definition *Entwicklungsmethode*

Eine *Entwicklungsmethode* wird auf Basis der vorangestellten Argumentation in dieser Arbeit wie folgt definiert.

Definition 1
Eine (Entwicklungs-)Methode ist das systematische Vorgehen zum Erreichen eines Entwicklungsziels.

1 http://www.duden.de/rechtschreibung/Methodik, letzter Aufruf: 2015-01
2 http://www.duden.de/rechtschreibung/Vorgehen, letzter Aufruf: 2015-01
3 http://www.duden.de/rechtschreibung/Verfahren, letzter Aufruf: 2015-01
4 http://www.duden.de/rechtschreibung/Methodologie, letzter Aufruf: 2015-01

Weiterhin wird beim Begriff Methode im Kontext dieser Arbeit immer eine Entwicklungsmethode gemeint.

Teil-Methoden

Bei komplexen Entwicklungsmethoden müssen verschiedene Teildisziplinen berücksichtigt werden, wie beispielsweise die Anforderungserhebung, Entwurf, Design, Implementierung oder Testen. Innerhalb der Teildisziplin Anforderungserhebung wird von einer Anforderungserhebungsmethode gesprochen werden. In dieser Arbeit werden Methoden der Teildisziplinen als *Teilmethoden* betitelt, da beispielsweise hier die Anforderungserhebung als notwendiges Zwischenergebnis oder *Teil-Entwicklungsziel* für das übergeordnete Entwicklungsziel angesehen wird. Eine Methode besteht demnach aus verschiedenen Teilmethoden für die verschiedenen Teildisziplinen.

Definition 2
Eine Entwicklungsmethode besteht aus Teilmethoden. Eine Entwicklungsmethode definiert eine Teilmethode in einer übergeordneten Disziplin.

unternehmensweit verstehen

In [ESS08] wird die Erkenntnis vorgestellt, dass eine situationsgerechte Entwicklungsmethode auf dem unternehmensweiten Verständnis der am Entwicklungsprozess beteiligen Artefakte, deren Struktur und Zusammenhang, basieren muss. Die Grundidee basiert auf dem grundlegenden Schritten *Ermittlung von Produktkonzepten*, welches ein Domänenmodell der Produktkonzepte erstellt, und *Ermittlung von Sprachen zur Beschreibung der Produktkonzepte*. Darauf aufbauend wird mit dem Schritt *Erstellung von Artefakttypen* ein Domänenmodell von Artefakttypen erstellt, bei dem die Zuweisung einer oder mehrerer Sprachen zur Beschreibung eines Produktkonzept stattfindet. Aufbauend darauf wird ein Prozess zur Erstellung konkreter Artefakte erstellt, wobei ein Artefakt eine ontologische Instanz eines Artefakttyps darstellt. Die Unterstützung des Entwicklungsprozesses durch Hilfsmittel, wie geeignete Werkzeuge, finalisiert die entstandene Entwicklungsmethode.

Es wird folgende Definition abgeleitet.

Definition 3
Eine Methode besteht aus dem Zusammenspiel aller Artefakte und Aktivitäten zur Erstellung der Artefakte, die zum Erreichen des Entwicklungsziels notwendig sind. Ein Prozessmodell definiert eine durchführbare Reihenfolge der Aktivitäten. Ein Artefakt ist eine ontologische Instanz eines Artefakttyps. Ein Artefakttyp besteht aus einem zu beschreibenden Konzept, welches mit einer geeigneten Sprache dargestellt wird, wobei ein Konzept ein oder mehrere Eigenschaften des Produkts beschreibt. Techniken und Hilfsmittel zur Erstellung der Artefakte und Durchführung der Aktivitäten sind ebenfalls Bestandteile einer Methode.

Begriffsabgrenzung Prozess

Die Benutzung von Begrifflichkeiten wird bei Methodenentwicklern wieder schwer verständlich, wenn Begriffe wie *Prozess*, *Software(-entwicklungs)prozess*, *Softwareprozessmodell* und *Vorgehensmodell* synonym verwendet werden. Sommerville beschreibt einen *Softwareprozess* als *Menge von Tätigkeiten und damit zusammenhängenden Ergebnissen*. Das passt zu meiner *Prozess*-Definition als Teil der Methode. *Tätigkeiten* können mit *Aktivitäten* und *Ergebnisse* mit *Artefakten* gleichgesetzt werden. Sommerville beschreibt das *Vorgehensmodell* als *eine vereinfachte Beschreibung eines Softwareprozesses, welche eine Sicht auf den Softwareprozess darstellt*. Weiterhin fügt ein Vorgehensmodell dem Softwareprozess Rollen und Produkte hinzu (siehe [Som07] S. 34f). Produkte (Ergebnisse) werden in dieser Arbeit ebenfalls als *Artefakte* angesehen. Die englische Übersetzung von *Vorgehensmodell* lautet *Software Process Model*, welches bei ungenauer Übersetzung in die deutsche Sprache zu Softwareprozessmodell wird und demnach synonym zum Begriff *Vorgehensmodell* genutzt wird.

Aus der Diskussion im vorigen Absatz wird erkannt, dass es notwendig ist ein Prozessmodell zur Beschreibung eines Prozesses festzulegen.

Definition 4
Eine Methode besteht aus einem Prozessmodell, welches explizit alle Prozesse definiert. Ein Prozess legt Aktivitäten und ihre durchführbare Reihenfolge fest.

Die mit den Aktivitäten zusammenhängenden Artefakte müssen ebenfalls definiert werden. Die Artefakte haben Abhängigkeiten untereinander, die über die Artefakttypen auf Typebene beschrieben werden. In [ESS08] wird die Bildung eines Domänenmodells der Artefakttypen empfohlen. In dieser Arbeit wird das Domänenmodell der Artefakttypen Produktmodell genannt.

Definition 5
Eine Methode besteht aus einem Produktmodell zur Definition der Artefakttypen und deren Abhängigkeiten untereinander. Die Artefakttypen definieren die Struktur instanziierbarer Artefakte, die wiederum von Aktivitäten aus dem Prozessmodell referenziert werden können.

Artefakte

Artefakte sind Instanzen der Artefakttypen. Während der Benutzung einer Methode werden unterschiedliche Artefakte erstellt. Die Zustände eines Artefakts sind ebenfalls wichtig. Zum Beispiel wird ein Dokument von einer Person mit einer bestimmten Rolle erstellt, bestimmte Inhalte des Dokuments werden aber von anderen Personen in anderen Rollen zugeliefert. Bei der Fertigstellung der Artefakte sind ebenfalls Zustände wie der Arbeitsfortschritt oder bei der Begutachtung der Finalisierungsstatus (bspw.: *in review*, *approved*, etc.) wichtig. Dazu wird eine Lebenszyklusdefinition der Artefakte benötigt.

Definition 6
Ein Lebenszyklusmodell definiert mögliche Zustände eines Artefakts auf Typebene.

Erweiterung durch Vorgehensmodelle

Der Begriff Vorgehensmodell wird in der weitergehenden Literatur auch für weitere Entwicklungsansätze genutzt, beispielsweise für Wasserfall-Modell, RUP, V-Modell, SCRUM oder Extreme Programming. Solche Entwicklungsansätze sind weniger auf die konkret zu lösenden Entwicklungsprobleme ausgerichtet, jedoch mehr auf allgemeine strukturelle Aspekte der Softwareentwicklung. Sie werden in dieser Arbeit als Sichten auf einen Prozess angesehen. Diese Sichten erweitern den Prozess um Rollen, Phasen, Meilensteine und Organisationsstrukturen.

Definition 7
Ein Prozessmodell ist über Sichten strukturierbar. Rollen, Meilensteine und Phasen beschreiben den Prozess.

Auf Basis der Definitionen in diesem Abschnitt ergibt sich das Metamodell für Methoden in Abbildung 2.1.

Methodenmetamodell

Das Methodenmetamodell definiert die grundlegenden Elemente einer Methode. Die Definition des Metamodellelements *ZusammengesetzteAktivität* als Spezialisierung einer *Aktivität* im Metamodell ermöglicht die Konstruktion komplexer Prozesse mit unterschiedlichen Detaillierungsgraden. Aktivitäten werden im

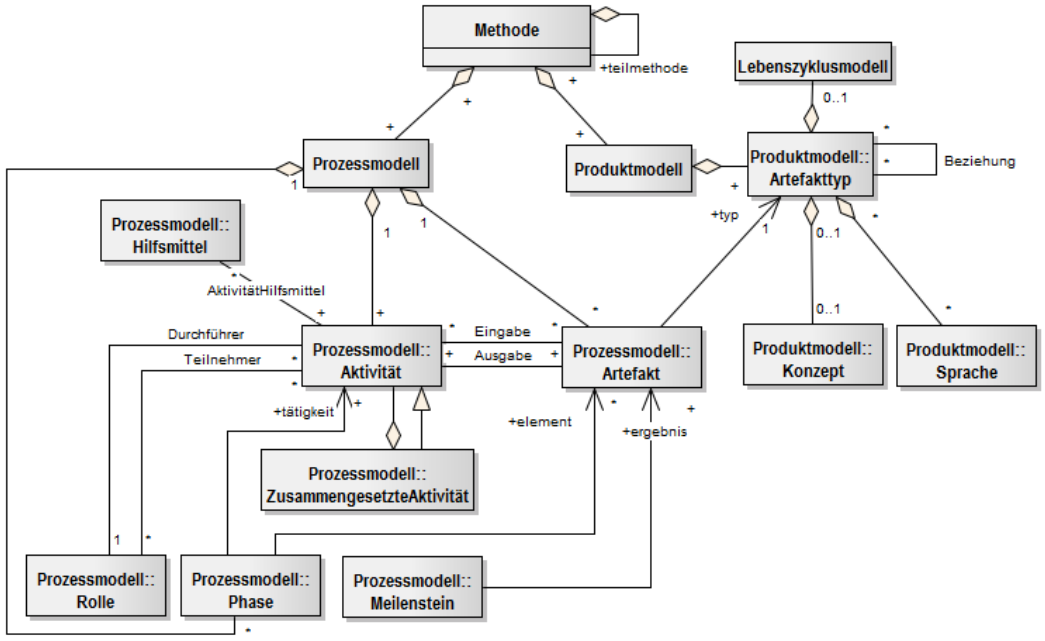


Abbildung 2.1: *Metamodell der Entwicklungsmethoden*

Prozessmodell aggregiert. *Artefakte* können von Aktivitäten als Ein- oder Ausgabe referenziert werden. Ein Artefakt wird getypt über einen *Artefakttyp*, welcher das *Lebenszyklusmodell* aggregiert. Die Beziehungen der Artefakttypen untereinander werden erstmal nur abstrakt definiert. Vorstellbar sind die Beziehungsstypen Generalisierung, Aggregation und Assoziation, wie sie in der Abbildung selber genutzt werden (vgl. UML-Klassendiagramm). Artefakttypen werden über das *Produktmodell* und Artefakte über das *Prozessmodell* aggregiert. Eine Methode definiert *Teilmethoden*. Die weiteren Struktureigenschaften eines Prozessmodells sind *Rollen*, die den Akteur festlegen, der eine bestimmte Aktivität durchführt. *Phasen* definieren eine Menge der Phase zugehörenden Aktivitäten und Artefakte. *Meilensteine* werden über die Fertigstellung bestimmter Artefakte definiert. *Hilfsmittel* werden Aktivitäten zugewiesen. Nicht alle Aggregationen sind explizit dargestellt, sondern nur implizit über die Paketaggregation. Rolle, Phase, Meilenstein, Artefakt und Hilfsmittel sind dem Prozessmodell angehörig (Prozessmodell::). Konzept und Sprachen werden im Produktmodell (Produktmodell::) definiert.

2.2 Methodenentwicklung

Method Engineering vs. SME

Im Allgemeinen befasst sich die Methodenentwicklung mit den Themen Konstruktion neuer, Adaption bestehender sowie der Formalisierung von Entwicklungsmethoden bei der Systementwicklung [HSR10]. Diese Arbeit befasst sich weitergehend mit der Berücksichtigung von gegebenen Einflussfaktoren auf die zu erstellende Methode. Diese wichtige Aufgabe ist durch das Forschungsgebiet Situational Method Engineering (SME) geprägt worden, bei dem das Ziel ist, die am besten passende Entwicklungsmethode hinsichtlich einer Situation zu erstellen. Dieser Abschnitt gibt eine kurze Zusammenfassung von SME mit seinen Begriffen und grundlegenden Paradigmen. Anschließend findet die Eingliederung des Begriffs *Methodenweiterentwicklung* in das Forschungsgebiet Methodenentwicklung statt.

Situation

Für den Begriff *Situation* gibt es keine eindeutige Definition, wie in [BKKW07] diskutiert wird. Eine Auswahl der Konzepte, die für die Beschreibung von situativen Eigenschaften genutzt werden, heißen *Reference Context*, *Project Environment*, *Project Situation*, *Development Situation* oder *Project Context*. Im Allgemeinen beschreiben die Konzepte die Einflussfaktoren auf die zu erstellende Methode. Am besten nachvollziehbar ist hier die Bestimmung des *Project Context* mit den 17 *contingency factors* von [SH96].

Vorgehen im SME

Die Community des SME ist sich einig, dass es keine allgemein passende, allumfassende Entwicklungsmethode gibt, die in sämtlichen Projekten und Organisationen für die Software-/Systementwicklung nutzbar ist (siehe [BKKW07] und [HS06]). Jedoch ist ein Zwiespalt vorhanden, ob es sinnvoller ist, eine bestehende Methode anzupassen oder neu zu entwickeln [HSR10]. Das Anpassen von Methoden fällt ebenfalls unter die Begriffe eine Methode adaptieren, zuschneiden und konfigurieren (Method Adaptation, Tailoring, Configuration). Das Neuentwickeln von Methoden wird auch Methodenkonstruktion (Method Construction) genannt und ist der größere Forschungsbereich.

Methodenkonstruktion

Bei der Methodenkonstruktion gibt es die Voraussetzung einer vorhandenen Methodenbasis, die Methodenfragmente oder auch größere Methodenstücke (vgl. engl. method fragment and method chunk) enthält. Die Methodenelemente sollen bereits über Metainformationen zur Bestimmung des möglichen Einsatzes in einer bestimmten Situation verfügen ([KDS07], [ARB07]). Das Methodenfragment

wird dabei definiert als ein atomarer Teil einer Methode, welcher als eine Klasse in einem Methodenmetamodell repräsentiert wird. Im Bereich SME findet im Allgemeinen eine konzeptuelle Trennung von Produkt- und Prozessanteilen einer Methode statt. Ein größeres Methodenstück (method chunk) ist definiert als die Kombination von Prozess- und Produktfragmenten, wobei der Produktanteil alle Eingabe- und Ausgabefragmente umfasst, die zur Ausführung des Prozessanteils benötigt werden.

Forschungsgebiete im SME

Das Forschungsgebiet SME umfasst die Bereiche Erstellung, Beschreibung und Ablage der *method chunks* unter Berücksichtigung der Situation. Die Bestimmung der Situation ist essentiell, jedoch muss auch verstanden werden, welcher Einfluss auf eine Methode durch eine bestimmte Eigenschaft zu erwarten ist. Darauf aufbauend findet die passende Auswahl der *method chunks* statt [RR01], [KDS07], [MR06]. Letztendlich führen die Erkenntnisse des SME zur Bereitstellung von Computer-Aided Method Engineering (CAME) Werkzeugen. Diese sind jedoch nur anwendbar, wenn die Situation bestimmbar ist und kompatible Methodenbausteine in einer Methodenbasis vorhanden sind.

fehlende Methodenbasis

Die fehlende Methodenbasis, also der Zugriff auf vorhandene *method chunks* für die Aufgabe der problemspezifischen Methodenkonstruktion, ist in der Praxis das größte Problem. Theoretisch müssten für alle möglichen Probleme Bausteine zur Verfügung stehen. Weiterhin müssten diese Bausteine auch für die unterschiedlichsten Situationen bereit stehen. Schwierig ist hier unter anderem die Handhabung von *method chunks* mit unterschiedlichem Abstraktionsgrad. Praktisch verfügt kein Projektpartner in den Forschungs- und Entwicklungsprojekten des s-lab – Software Quality Labüber eine benutzbare Methodenbasis.

Methoden-weiterentwicklung

Der Ansatz, welcher in dieser Arbeit eingeführt wird, befasst sich nun in einem ersten Schritt mit der Bestandsaufnahme der Ist-Methode. Auch wenn keine Methode explizit vorliegt, wird eine implizite Methode vorausgesetzt und das führt zu einer Informationsgrundlage auf deren Basis zuerst eine Schwachstellenanalyse durchgeführt werden kann. Eine Situationsbestimmung soll weitere Anpassungen in Form von Methodenanforderungen definieren. Mit Hilfe dieser Grundlagen soll die bestehende Ist-Methode in eine Soll-Methode überführt werden. Die Methodenentwicklung findet deswegen aber nicht auf *der grünen Wiese* statt, weswegen der Begriff Methodenneuentwicklung nicht passt. Die Ist-Methode beschreibt das Gegebene und ist so speziell, dass eine komplette Veränderung nicht gewünscht wird. Aus diesem Grund ist eine Adaption, Tailoring oder Konfiguration bestehen-

der allgemeiner Methoden nicht angebracht. Aufgrund der fehlenden Methodenbasis ist der in dieser Arbeit beschriebene Ansatz auch nicht dem Forschungsgebiet der Methodenkonstruktion zuzuweisen (auch wenn der Name das vermuten lässt). Weiterhin werden auch keine Methodenelemente einzeln entwickelt und später passend zusammengeführt. Es findet eine, mit Sicht auf die anliegende Problemsituation, ganzheitliche Entwicklung statt. CAME-Werkzeuge sind für unseren Ansatz nicht anwendbar. Dennoch, unter Berücksichtigung der Situation, wird der Ansatz der Domäne SME zugeordnet. Aufgrund der schwierigen Eingliederung in Themen des SME wird der neue Begriff *Methodenweiterentwicklung* genutzt.

Im folgenden Abschnitt befassen wir uns mit den Aufgaben des Methodenentwicklers bei der Methodenweiterentwicklung.

2.3 Das Vorgehen der Methodenweiterentwicklung

Aufgaben des Methodenentwicklers

Die Aufgabe des Methodenentwicklers ist, wie der Rollenname schon aussagt, die Entwicklung einer Methode. In diesem Abschnitt werden die Aufgaben des Methodenentwicklers bei der Methodenweiterentwicklung vorgestellt. Es basiert generell auf Erfahrungen von im s-lab – Software Quality Lab durchgeführten Projekten. Dieses Vorgehen wird erstmalig formalisiert und definiert notwendige Konzepte der Methodenentwicklung. Es werden die verschiedenen Phasen *Ist-*

Abbildung 2.2: *Vorgehen der Methodenweiterentwicklung*

Entwicklungsphasen

Analyse, *Verbesserungs-Analyse*, *Konzeption*, *Realisierung* und *Einführung* betrachtet (vgl. Abbildung 2.2), die in den folgenden Unterabschnitten vorgestellt werden.

2.3.1 Ist-Analyse

implizite Methoden

Diese Phase gestaltet die ersten Tätigkeiten bei der Methodenentwicklung. Es wird davon ausgegangen, dass, wenn auch nicht explizit beschrieben, eine bestimmte Arbeitsweise zur Erlangung eines Entwicklungsziels vorherrscht. Das heißt, auch wenn eine Organisation kein Methodenhandbuch mit einer definierten Methode vorliegen hat, die Mitarbeiter ihre Aufgaben dennoch nicht unstrukturiert bearbeiten, sondern implizit einer Methode folgen. Die *Ist-Analyse* hat zum einen die Aufgabe zu bewältigen, bestehende methodische Elemente zu identifizieren, also eine Übersicht über die *Ist-Methode* zu erlangen. Zum anderen ist für das hier beschriebene Vorgehen wichtig, den *Methodenkontext* zu ermitteln. Beide Erkenntnisse *Ist-Methode* und *Methodenkontext* liefern die Gesamtübersicht über den *Ist-Zustand* (siehe Abbildung 2.3).

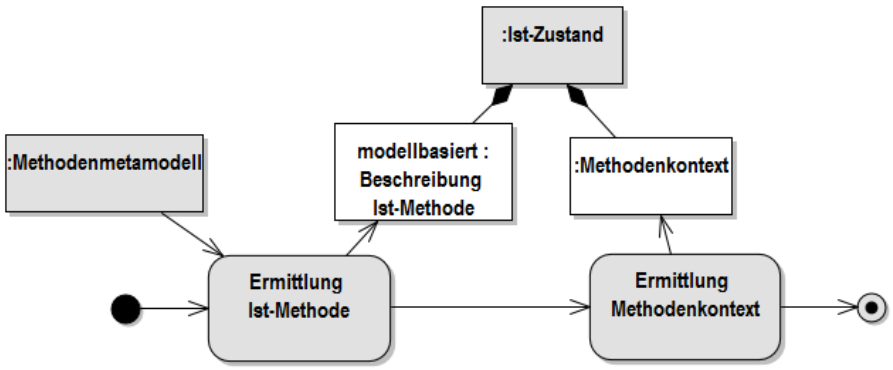


Abbildung 2.3: *Ist-Analyse*

Ermittlung Ist-Methode. Die erste Aufgabe des Methodenentwicklers ist die *Ermittlung Ist-Methode*. Die *Ist-Methode* umfasst existierende Methodenelemente der Entwicklungsmethode. Bei der Ermittlung von Methodenelementen hilft uns beispielsweise das Methodenmetamodell aus Abbildung 2.1. Dort sind alle relevanten Typen von Methodenelementen definiert, nach denen recherchiert werden muss.

Dokumentation der Ist-Methode

Eine geeignete Sprache zur Dokumentation der Ist-Methode ist SPEM, da mit ihr Arbeitsprodukte und Prozessbestandteile modelliert werden können. Eine weitere, in diesem Abschnitt genutzte Sprache ist das UML-Aktivitätsdiagramm in Kombination mit dem UML-Klassendiagramm. Mit dem UML-Klassendiagramm kön-

nen Arbeitsprodukte als Artefakttypen in Form von UML-Klassen definiert werden. Die Prozessbestandteile, meistens UML-Aktivitäten, können über Kontrollflusskanten miteinander verbunden werden und so einen Ablauf beschreiben. Weiterhin können mit Hilfe von Objektflusskanten Instanzen der Artefakttypen aus dem Klassendiagramm referenziert werden. In der Abbildung 2.3 werden die Instanzen der Artefakttypen, *Methodenmetamodell*, *Hilfsmittel*, *Ist-Methode*, *Methodenkontext* und *Ist-Zustand*, als Ein- und Ausgabeprodukte beschrieben sowie deren struktureller Zusammenhang. An dieser Stelle wird auf die Teilaufgabe der Aufgabenstellung der Dissertation 2 *Definition einer MetaMe kompatiblen Methodenbeschreibungssprache* hingewiesen.

Ermittlung Methodenkontext Neben den bestehenden Methodenelementen muss in der nächsten Aktivität *Ermittlung Methodenkontext* mittels einer Kontextanalyse der Methodenkontext ermittelt werden, da dieser beim weiteren Vorgehen stets berücksichtigt werden muss. Ein archäologisches Vorgehen zur Suche bestehender Methodenelemente und des Methodenkontexts ist in der Regel üblich. Für die Kontextanalyse müssen generell die Entwicklungsdomäne, betriebliche Organisationsumgebung sowie die Projektorganisation mit ihren speziellen Eigenschaften berücksichtigt werden. Abbildung 2.4 beschreibt den Methodenkontext in Form eines Schalenmodells.

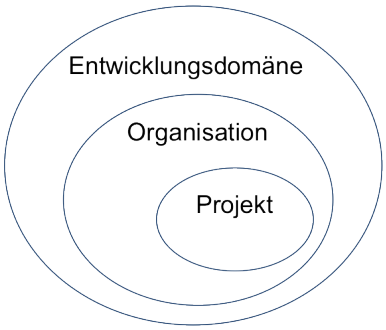


Abbildung 2.4: *Methodenkontext*

Unterstützende *Hilfsmittel* für die Ermittlung des Methodenkontexts sind Checklisten, Guidelines, Fragebögen und interaktive Treffen mit am Entwicklungsprozess beteiligten Personen. Das führt zur Verfeinerung der Teilaufgabe der Aufgabenstellung der Dissertation 1 *Detaillierung des Vorgehens der Methodenweiterentwicklung, so dass ein Methodenentwickler das Vorgehen versteht und durchführen kann.*

Teilaufgabe der Dissertation 3
Bereitstellung der Hilfsmittel zur Ermittlung des Methodenkontexts.

Eine wichtige Aufgabe ist die Aktivität *Dokumentation Ist-Zustand*. Der Ist-Zustand beschreibt den Stand der aktuellen Entwicklungsmethode im gegebenen Methodenkontext und wird benötigt, um das zusammengetragene Wissen zu konsolidieren. Weiterhin bietet der dokumentierte *Ist-Zustand* die Grundlage für die Darstellung von Optimierungspotenzialen. Der Ist-Zustand besteht neben der Ist-Methode aus dem Methodenkontext.

Dokumentation Ist-Zustand

Teilaufgabe der Dissertation 4
Dokumentation des Methodenkontexts.

2.3.2 Verbesserungs-Analyse

Nachdem der *Ist-Zustand* ermittelt ist, folgt die Phase der *Verbesserungs-Analyse*. In dieser Phase werden *Optimierungspotenziale* auf Basis des *Ist-Zustands* ermittelt, konkrete *Verbesserungen* beschrieben und Teil- oder Folge-Projekte zur Umsetzung der Verbesserungen festgelegt. Die Phase ist in Abbildung 2.5 dargestellt.

Ziele der Verbesserungs-Analyse

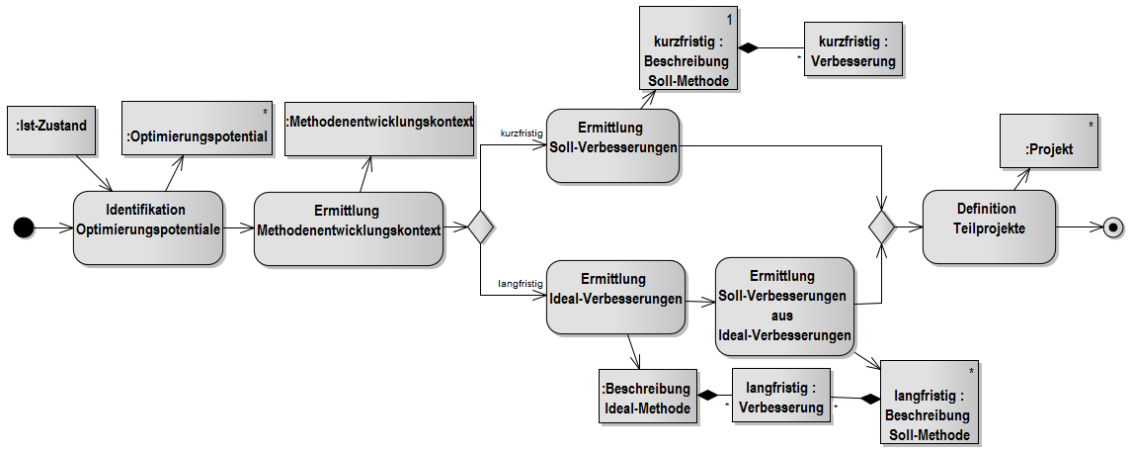


Abbildung 2.5: *Verbesserungs-Analyse*

Identifikation Optimierungspotenziale. Die Aktivität *Identifikation Optimierungspotenziale* dient zur Identifikation offensichtlicher Schwachpunkte der Ist-Methode. Beispiele hierfür sind:

- Fehlerhafte Inhalte in den Artefakten
- Inkonsistente Inhalte in den Artefakten
- Schwierigkeiten bei der Erstellung von Artefakten
- Schwierigkeiten bei der Verwendung von Artefakten
- Mögliche Entstehung von Schwachstellen in den Artefakten aufgrund von Schwachstellen in definierten Artefakttypen

Ermittlung Methodenentwicklungskontext. Ist eine Menge an Optimierungspotenzialen ermittelt, stellt sich die Frage, inwieweit diese Optimierungspotenziale bei der Methodenweiterentwicklung betrachtet werden können. Dafür ist es wichtig den Methodenentwicklungskontext zu kennen. Er definiert letztendlich wie viele Ressourcen bei der Methodenweiterentwicklung zur Verfügung stehen. Die wichtigsten Ressourcen sind Zeit, Kosten und verfügbares Know-how. Durch die Übersicht darüber wird rudimentär festgelegt, welche Themen überhaupt in Angriff genommen werden können. Weiterhin wird mit der Kenntnis über den Methodenentwicklungskontext das generelle Vorgehen festgelegt. Es wird unterschieden zwischen einer kurzfristigen, schnellen Lösung und einer langfristigen, strategisch sinnvollen Lösung.

kurzfristige Alternative

Ermittlung Soll-Verbesserungen. Die kurzfristige Alternative nimmt sich eine Auswahl der Optimierungspotenziale vor und mit der Aktivität *Ermittlung Soll-Verbesserungen* werden eine Menge von Verbesserungen ermittelt, die in ihrer Gesamtheit eine Soll-Methode beschreiben. Auf diese Weise können direkt Verbesserungen für die Aufhebung der offensichtlichen Schwachpunkte erarbeitet werden.

Gefahren bei kurzfristiger Planung

Die Schwierigkeit in diesem Ansatz ist die fehlende allumfassende Sicht, die Probleme mit sich bringt. Es ist denkbar, bereits vorhandene Artefakte, die in der Ist-Methode schon grundlegend zu Schwierigkeiten führen, zu verbessern. Beispielsweise ist das in der Ist-Situation ermittelte Artefakt unnötig, hat geringen Nutzen bei der Erreichung des Entwicklungsziels oder behindert weitere notwendige Artefakte. Weiterhin bringt diese Herangehensweise die mit Risiko behafte-

te Entwicklung von Insellösungen mit sich. Die Gefahr besteht, dass punktuelle Verbesserungen in verschiedenen Bereichen der Entwicklungsmethode nicht integrierbar sind. Das Integrationsproblem wird mit Hilfe des Vorgehensmodells V-Modell nach Boehm [Boe79] erklärt. Es werden vorhandene Spezifikationsinhalte nicht nur als Eingangsinformation zur Erstellung verfeinerter Spezifikationsinhalte oder der Implementierung genutzt, sondern auch als Testbasis in der Qualitätssicherung. Werden die späteren Testkonzepte bei der Verbesserung der Spezifikationsinhalte nicht berücksichtigt, wird eine aufwendige Verbesserung der Spezifikationsinhalte zwar die Entwicklung vereinfachen, jedoch hinderlich beim Testen sein. Die verbesserten Spezifikationsinhalte sind eventuell nicht als Testbasis nutzbar, sondern müssen übersetzt oder transformiert werden. Im schlimmsten Fall müssen die Testkonzepte mit zusätzlichem Aufwand angepasst werden.

langfristige Alternative

Ermittlung Ideal-Verbesserungen. Der weitaus sicherere Weg, aber natürlich auch aufwendigere, legt Wert auf die Ermittlung von Verbesserungen, die eine Ideal-Methode beschreiben (vgl. Aktivität *Ermittlung Ideal-Verbesserungen*). Der Vorteil liegt in der Festlegung langfristiger Ziele. Der Aufwand steigt bedingt durch die Definition einer Ideal-Methode als übergeordnetes Ziel. Die Beschreibung der Ideal-Methode definiert die bestmögliche Entwicklungsmethode unter Berücksichtigung des Methodenkontexts. Ein Vergleich der Ist-Methode mit der Beschreibung der Ideal-Methode ermöglicht eine systematische allumfassende Schwachstellenanalyse, die die im vorangegangenem Paragraphen beschriebenen Nachteile aufhebt. Die Beschreibung der Ideal-Methode stellt die Zusammenhänge aller im Entwicklungsprozess notwendigen Artefakte dar. Dadurch können falsche Artefakte der Ist-Methode aufgedeckt werden. In dieser Arbeit gehe ich davon aus, dass der Methodenentwickler Zugriff auf einen dokumentierten Ist-Zustand hat und vorausschauend eine Soll-Methode mit Wissen über die Ideal-Methode plant.

inkrementelle Soll-Verbesserung

Ermittlung Soll-Verbesserungen aus Ideal-Verbesserungen. Die Aktivität *Ermittlung Soll-Verbesserungen aus Ideal-Verbesserungen* bildet inkrementelle Teilmengen (Soll-Verbesserung) der gesamten Menge der Ideal-Verbesserungen. Der Vorteil der Berücksichtigung der Ideal-Methode wird in der Abbildung 2.6 skizziert. Es wird dargelegt, dass die Soll-Methode auf einem Migrationspfad zur Ideal-Situation liegt, während die Soll-Methode, ohne Definition der Ideal-Methode, auf einem von dem Ist-Zustand ausgehenden Migrationspfad liegt, der nicht

zwangsläufig in der Ideal-Methode endet. Sämtliche Verbesserungen, die die Soll-Methode beschreiben, werden in dieser Arbeit Soll-Verbesserungen genannt. Die Festlegung einer Soll-Methode mit den zugeordneten Soll-Verbesserungen ist ein Synonym für eine Iteration. Alle aus den Soll-Verbesserungen abgeleiteten Iterationen beschreiben einen Migrationspfad zur Soll-Situation.

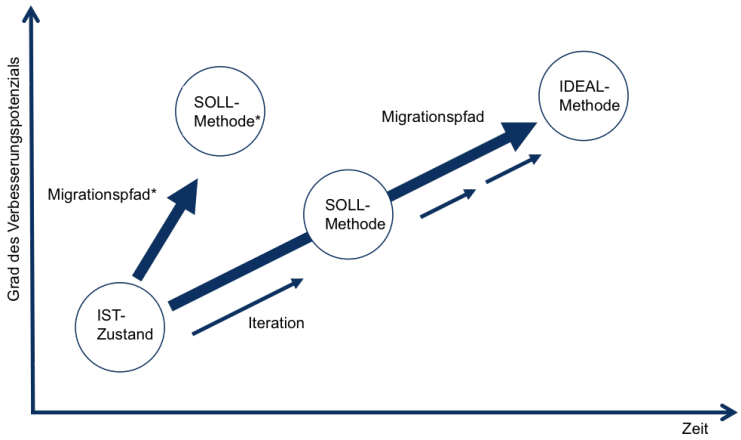


Abbildung 2.6: *Beispiel des Migrationspfads*

Sättigung

Die Bildung der inkrementellen Teilmengen von Soll-Verbesserungen ist ebenfalls abhängig vom Methodenentwicklungskontext. Die folgende Abbildung 2.7 beschreibt eine Sättigungskurve. In den frühen Iterationen der Methodenverbesserung sind mit geringem Aufwand größere Verbesserungen erreichbar als bei späteren Iterationen. Die Erreichung der Ideal-Methode wird letztendlich zu aufwendig. Eine mittelfristige Soll-Methode ist jedoch durch den reduzierten Gesamtaufwand oder ein Mindestziel im Optimierungsgrad gerechtfertigt. Diese Entscheidung wird ebenfalls auf Basis des Methodenentwicklungskontexts gefällt.

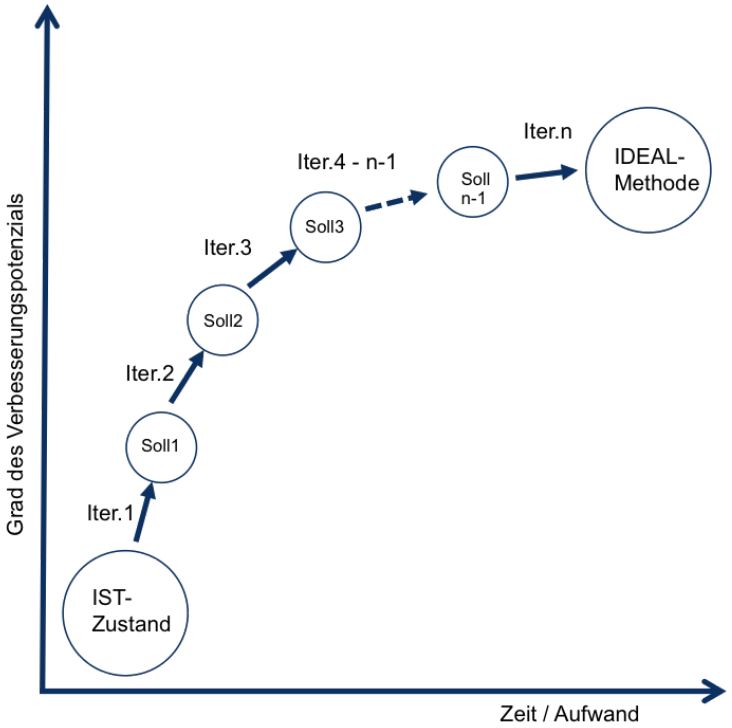


Abbildung 2.7: *Anstieg des Aufwands pro Iteration*

Eine Potenzialanalyse hilft bei der Priorisierung und Auswahl der sinnvollsten Optimierungen. Auf diese Weise können Verbesserungen gemäß ihrer Priorisierung zusammengefasst werden und eine Soll-Methode beschreiben. Potenzialanalyse

Die Ideal-Methode gilt als langfristig anzustrebendes Ziel für die Methodenverbesserung. Die Gleichsetzung der Ideal-Situation mit der Soll-Situation wird aufgrund des Methodenentwicklungskontexts generell nicht sinnvoll sein, es sei denn, der Methodenentwicklungskontext bietet keine Einschränkungen hinsichtlich der Entwicklung der Ideal-Methode. Ideal- vs. Soll-Situation

Initiierung von Projekten. Letztendlich, nachdem die Iterationen über die Soll-Verbesserungen festgelegt werden, bleibt die Aufgabe der Methodenerstellung. Nachdem Iterationen mit Soll-Verbesserungen definiert sind, folgt die Aktivität *Definition Teilprojekte*, die äquivalent zu den festgelegten Iterationen weitere (Teil-)Projekte für die *Konzeption*, *Realisierung* und *Einführung* der einzelnen Soll-Methoden einplant.

2.3.3 Konzeption

Ziele der Konzeption

Nachdem Projekte für die Optimierung der Ist-Methode definiert sind und die Verbesserung für jede Iteration zur Entwicklung einer Soll-Methode definiert ist, beginnt die Aufgabe der Konzeption der Soll-Methode. Nach [ESS08] sind die Aktivitäten *Konzeptermittlung*, *Sprachauswahl*, *Artefakttypdefinition*, *Prozessmodelldefinition* und *Hilfsmitteldefinition* zu betrachten (vgl. Abbildung 2.8).

Abbildung 2.8: *Konzeption*

Konzeptermittlung. Zuerst müssen die Konzepte erkannt werden, deren Umsetzung notwendig ist, um das Produkt zu erstellen. Das geschieht in der Aktivität *Konzeptermittlung*. [ESS08] empfiehlt die Erstellung eines Domänenmodells der Konzepte.

Produktkonzepte

Bei der Server-System-Entwicklung ist es beispielsweise unerlässlich eine Systemarchitektur zur Beschreibung von Systemkonfigurationen zu erstellen. Die Systemarchitektur ist ein domänenspezifisches Konzept oder allgemeiner ausgedrückt ein notwendiges Arbeitsprodukt. Solche Arbeitsprodukte werden als Produktkonzept definiert. Die ermittelten Produktkonzepte sind Ergebnisse einer detaillierten Analyse der Entwicklungsdomäne. Produktkonzepte beziehen sich konkret auf das zu entwickelnde Produkt.

Entwicklungskonzepte

Es gibt weitere Konzepte, die sich auf die Art und Weise der Entwicklung beziehen. Das sind Paradigmen oder Techniken und werden in dieser Arbeit Entwicklungskonzepte genannt. Beispielsweise unterliegen dem Paradigma der Objektorientierung die zwei Entwicklungskonzepte Klassenkonzept und Instanziierungskonzept. Berücksichtigt die Methodenentwicklung das Paradigma der Objektorientierung, wird das Produktkonzept *Systemkonfiguration* detailliert auf eine allgemeine Klassen-basierte Systemkonfiguration und eine konkrete Systemkonfiguration auf Instanzebene. Das Vorgehen des Methodenentwicklers muss ei-

ne Klassifizierung nach Produkt- und Entwicklungskonzept durchführen. Die Definition der Konzepttypen wird in Abbildung 2.9 dargestellt.

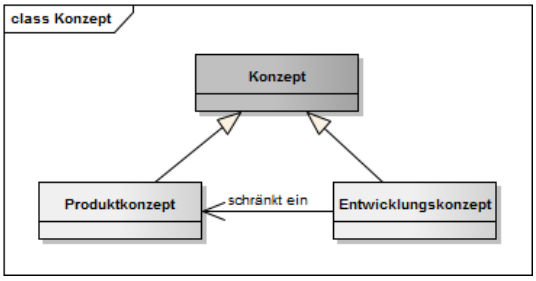


Abbildung 2.9: *Konzept*

Sprachauswahl. Die Auswahl einer passenden Sprache in der Aktivität *Sprachauswahl* impliziert unterschiedlichen Aufwand. Im Idealfall findet sich ein eindeutiger Kandidat. Eine Auswahl an mehreren Sprachen bedingt eine Evaluation zur Auswahl der am besten passenden Sprache. Im aufwendigsten Fall muss eine Sprach-Anpassung, -Weiterentwicklung oder -Neuentwicklung stattfinden. Als *Notlösung* wird häufig die natürliche verwendet. Jedoch birgt eine nicht präzise definierte Semantik, wie die der natürlichen Sprache, häufig einen Grund für mangelnde Qualität des Produkts.

Artefakttypdefinition. Die dokumentierte Zuordnung von Sprache zu Konzept bildet ein Artefakttyp und wird durch die Aktivität *Artefakttypdefinition* beschrieben. Die Beziehungen der Konzepte untereinander müssen an dieser Stelle berücksichtigt werden, da sich diese in den Beziehungen der Artefakttypen untereinander widerspiegeln. Wie bei den Konzepten wird hier ein Domänenmodell der Artefakttypen erstellt. Dieser Schritt bildet die wichtigste Grundlage der Methodenentwicklung. Hier werden die Strukturen des zu entwickelnden Produkts sichtbar, der zukünftige Einsatz von Sprachen wird definiert und der Umfang der Entwicklungsmethode wird abschätzbar. Diese Punkte müssen mit allen Beteiligten und Verantwortlichen abgeglichen werden. Erst die konkrete Beschreibung eines Konzepts mit einer Sprache erstellt eine Instanz eines Artefakttyps, also ein Artefakt.

Prozessdefinition. Kennt der Methodenentwickler das Domänenmodell der Artefakttypen, macht er sich Gedanken über den Entwicklungsprozess. Typische Fragestellungen in der Aktivität *Prozessdefinition* sind:

- Welche Aktivitäten sind für die Erstellung eines Artefakts durchzuführen?
- Welche Vorbedingungen sind notwendig, eine Aktivität durchzuführen?
- Welche Rollen sind an der Aktivität beteiligt?
- Gibt es strukturelle Vorgaben für den Entwicklungsprozess?

Strukturierung

Bei der letzten Frage werden standardisierte Vorgehensmodelle berücksichtigt. Beispielsweise definiert RUP Disziplinen und Phasen, mit denen der Entwicklungsprozess strukturiert wird. Darüber hinaus gibt es festgelegte Rollendefinitionen. Wenn kein standardisiertes Vorgehensmodell zu berücksichtigen ist, gibt es vielleicht andere betriebsspezifische Vorgaben zur Erstellung einer proprietären Struktur. Erkannte Aktivitäten werden vorgegebenen Disziplinen und Phasen zugeordnet. Nachdem die gestellten Fragen beantwortet sind, besteht die Möglichkeit einen auf das Betriebsumfeld passenden Prozess zu erstellen. Ein Prozess besteht jedoch nicht nur aus den Aktivitäten, sondern definiert ebenfalls die Abarbeitungsreihenfolge der Aktivitäten. Für die Beschreibung der Abarbeitungsreihenfolge gibt es verschiedene Möglichkeiten basierend auf:

- dem Kontrollfluss über den Aktivitäten
- dem Objektfluss der Artefakte zwischen den Aktivitäten
- Zustandsänderungen der Artefakte.

Hilfsmitteldefinition. Die Aktivität der Konzeption bei der Methodenentwicklung ist die *Hilfsmitteldefinition* für die Durchführung des Prozesses. Hilfsmittel können Werkzeuge, Leitfäden, Anleitungen, Dokumentvorlagen, usw. sein.

Ergebnis der Konzeption

Das Ergebnis der Konzeption ist ein modellbasierter Entwurf für eine Soll-Methode einer Iteration der Methodenweiterentwicklung, die im Weiteren realisiert und eingeführt werden muss.

2.3.4 Realisierung und Einführung

Nach der Konzeption einer Soll-Methode muss diese implementiert und anschließend eingeführt werden.

Realisierung. Bezüglich der Aktivität *Realisierung* sind eine Menge unterschiedlicher Tätigkeiten durchzuführen. Für die Benutzung einer Methode muss letztendlich sichergestellt werden, dass die Durchführung des Prozesses und die Erstellung der Artefakte gewährleistet ist. Im einfachsten Fall gibt es eine verfügbare Beschreibung des Prozesses. Sie besteht aus einer detaillierten Beschreibung, wie einzelne Aktivitäten des Prozesses durchzuführen sind sowie der Möglichkeit ein Artefakt zu erstellen, auch wenn es sich nur um die Bereitstellung eines Schreibprogramms zur Erstellung von Textinhalten handelt. Im komplexesten Fall gibt es hoch spezialisierte Prozessausführungswerkzeuge mit integrierter Werkzeugkette zur Erstellung der Artefakte.

Einführung. Vergleichen wir die Einführung einer Methode mit der Einführung einer Softwarelösung oder eines Prozesses ergeben sich verschiedene Einführungsstrategien. [HLL12] beschreibt beispielsweise die beiden Dimensionen *räumlich* und *zeitlich* mit den folgenden Strategien. Die Big-Bang-Einführung beschreibt eine unternehmensweite Ad-hoc-Einführung, Step-by-step eine sukzessive Einführung an verschiedenen Orten und ein pilotierter Roll-out die zeitliche Versetzung der Einführung, so dass die erste Einführung einen Pilotcharakter erhält. Weitere Einführungsstrategien können Top-Down, Bottom-Up, parallel, inkrementell oder evolutionär ausgelegt werden. An dieser Stelle muss genau geprüft werden, welche Einführungsstrategie ein geringes Risiko bei der Einführung mit sich bringt. Die Einführung muss zusammen mit den Verantwortlichen der Organisation geplant werden.

2.3.5 Übersicht der vorgestellten Produktkonzepte

Abschließend werden die in den vorangegangenen Unterabschnitten vorgestellten Produktkonzepte im Zusammenhang vorgestellt (siehe Abbildung 2.10).

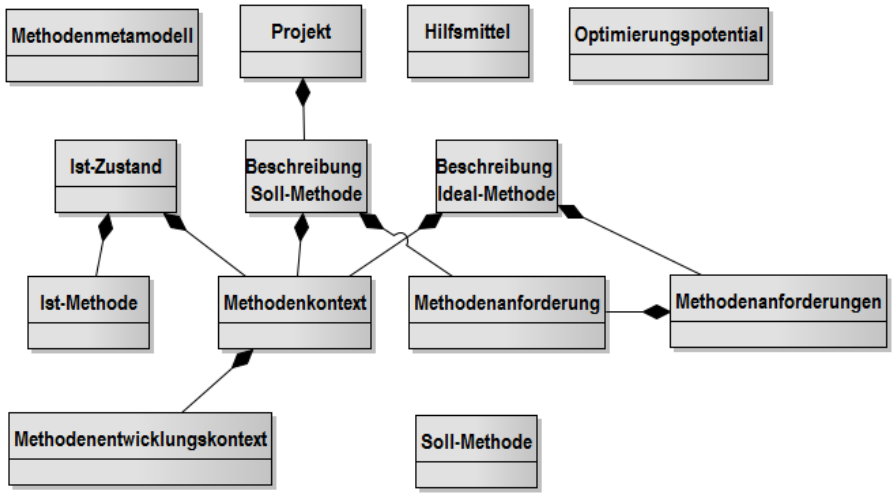


Abbildung 2.10: *Produktkonzepte der Methodenentwicklung*

2.4 MetaMe

bisheriger Ansatz

Bei der problemspezifischen Methoden-Entwicklung oder -Anpassung in s-lab – Software Quality LabForschungs- und Entwicklungsprojekten haben wir bisher den MetaMe-Ansatz verwendet. MetaMe ist ein systematischer Ansatz zur Methodenentwicklung im Bereich der Softwareentwicklungsmethoden. Die Basis bildet die grundlegende Idee aus [ESS08], die von Stefan Sauer in seiner Dissertation erweitert und detailliert worden ist [Sau11b]. MetaMe ist genauer gesagt eine Meta-Methode zur Entwicklung von Softwareentwicklungsmethoden.

Struktur des Abschnitts

Zuerst wird der Aufbau von MetaMe betrachtet. MetaMe selbst besteht aus einem MetaMe-Prozess- und MetaMe-Produktmodell genau wie im Methodenmetamodell aus Abbildung 2.1 definiert worden ist. Das MetaMe-Prozessmodell wird in Unterabschnitt 2.4.1 dargestellt, das MetaMe-Produktmodell in Unterabschnitt 2.4.2. Danach wird in Unterabschnitt 2.4.3 das Anwendungskonzept von MetaMe vorgestellt. Dieser Abschnitt schließt mit einer Diskussion der Schwachstellen von MetaMe mit Definition von Anforderungen an die Erweiterung von MetaMe.

2.4.1 MetaMe-Prozessmodell

Das MetaMe-Prozessmodell wird in Abbildung 2.11 vorgestellt.

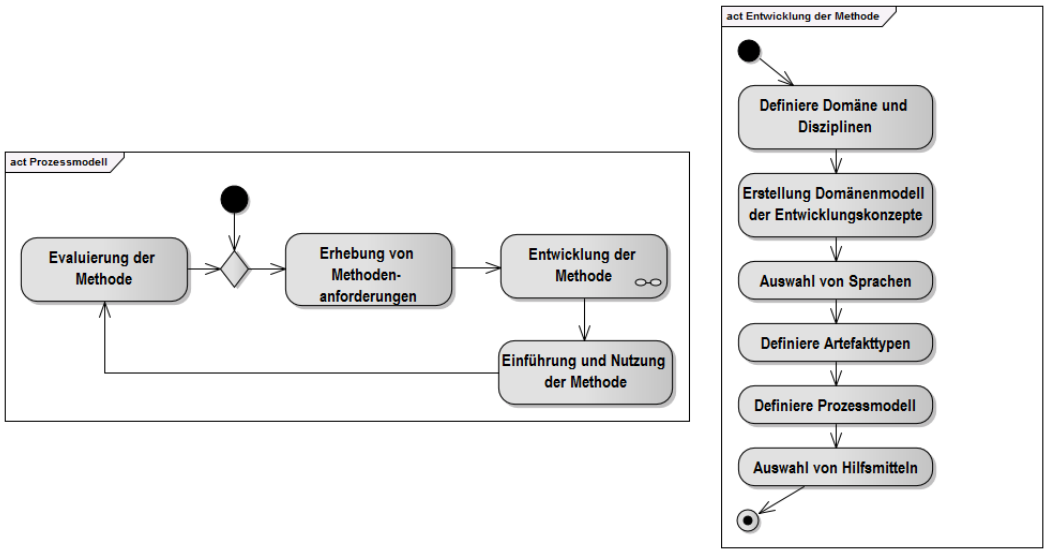


Abbildung 2.11: *MetaMe-Prozessmodell (vgl.: [Sau11b])*

Aufbau des Prozessmodells

Die Abbildung zeigt auf der linken Seite das übergeordnete Vorgehen mit den Schritten *Erhebung von Methodenanforderungen*, *Entwicklung der Methode*, *Einführung und Nutzung der Methode* und *Evaluierung der Methode*. Auf der rechten Seite der Abbildung wird die Aktivität *Entwicklung der Methode* verfeinert in die Schritte *Definiere Domäne und Disziplinen*, *Erstellung Domänenmodell der Entwicklungskonzepte*, *Auswahl von Sprachen*, *Definiere Artefakttypen*, *Definiere Prozessmodell* und *Auswahl von Hilfsmitteln*. Das Entwickeln der Methode folgt dem Ansatz aus [ESS08]. Die weiteren Aktivitäten werden nicht verfeinert.

2.4.2 MetaMe-Produktmodell

Elemente des Produktmodells

Das MetaMe-Produktmodell wird in Abbildung 2.12 dargestellt.

Das MetaMe-Produktmodell definiert die konzeptionellen Bestandteile von Methoden. Das Element *Method* referenziert *Discipline* als oberstes Strukturelement. Die *Discipline* referenziert zum einen für Artefakt relevante Elemente wie *Artifact*, *Artifact Lifecyle*, *Model*, *Notation* und *Concept*. Zum anderen werden die für den Prozess relevanten Elemente mit oberster Abstraktion in Form von *Work* referenziert.

Das MetaMe-Produktmodell erlaubt über die weitere Detaillierung von Aktivitäten (*Activity* und *Task*) sowie atomaren Schritten (*Action Step*) eine Verfeinerung von Prozessen (*Process*). Die weiteren, für den Prozess relevanten, Elemente sind *Role*, *Milestone* und *Phase*.

Die Methode referenziert Hilfsmittel mit oberster Abstraktion in Form von *Technique* und den Detaillierungen *Tool*, *Practice*, *Guidance* und *Utility*.

Konzepte werden über ein Domänenmodell (*Domain Model*) definiert.

Die von MetaMe angedachte Möglichkeit verschiedene Sichten auf das Domänenmodell zuzulassen wird über das Strukturelement *Aspect View* definiert.

MetaMe lässt es zu, Artefakte als Modell (*Model* und *Model Element*) zu definieren und über Aktivitäten Transformationen (*Transformation*) oder Modelltransfor-

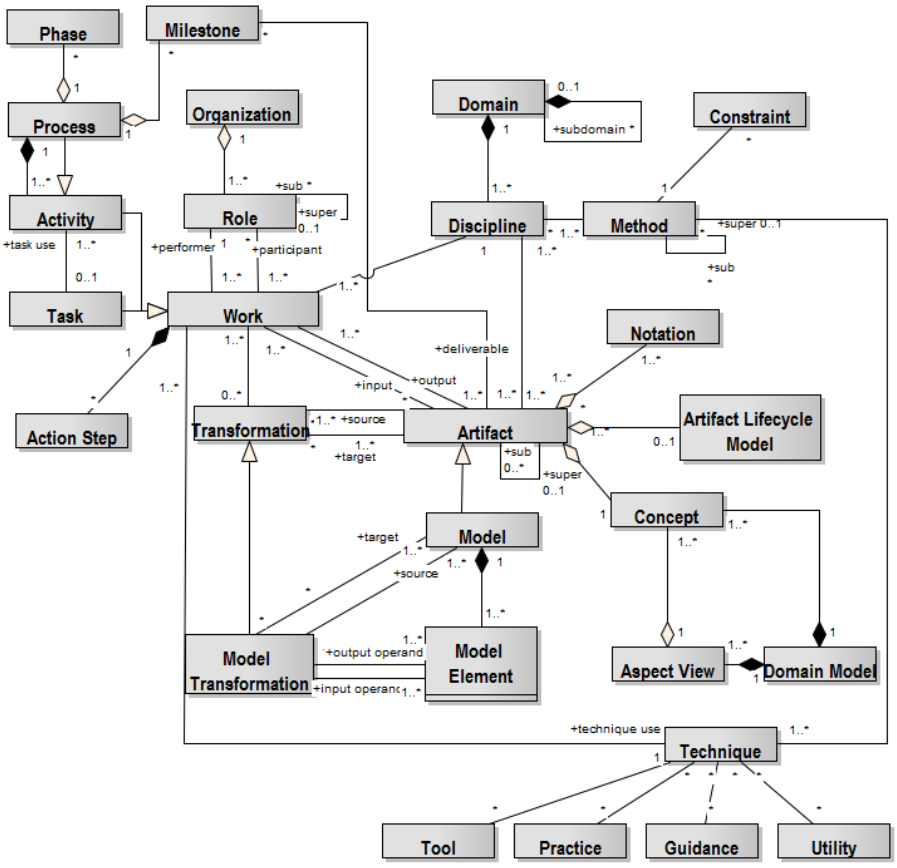


Abbildung 2.12: *MetaMe-Produktmodell (vgl.: [Sau11b])*

mationen (*Model Transformation*) zu beschreiben. Dadurch ist es möglich, im Sinne von Graphtransformationen, eine formalisierte Automatisierung von Prozessbestandteilen einer Methode zu beschreiben. Diese Möglichkeit wird in dieser Arbeit jedoch nicht weiter betrachtet.

Über die Elemente *Organization* und *Domain* werden die organisations- und domänenspezifischen Aspekte einer Methode berücksichtigt.

2.4.3 MetaMe-Anwendungskonzept

Methoden- und Softwareentwicklung

Im Folgenden betrachten wir die Anwendung von MetaMe und exemplarisch die Eingliederung der benutzten und erstellten Modelle in die unterschiedlichen Modellebenen der beiden Domänen Methoden- und Softwareentwicklung. Dadurch wird der Zusammenhang zwischen beiden Domänen in dem kombinierenden Ansatz von MetaMe verdeutlicht (siehe Abbildung 2.13).

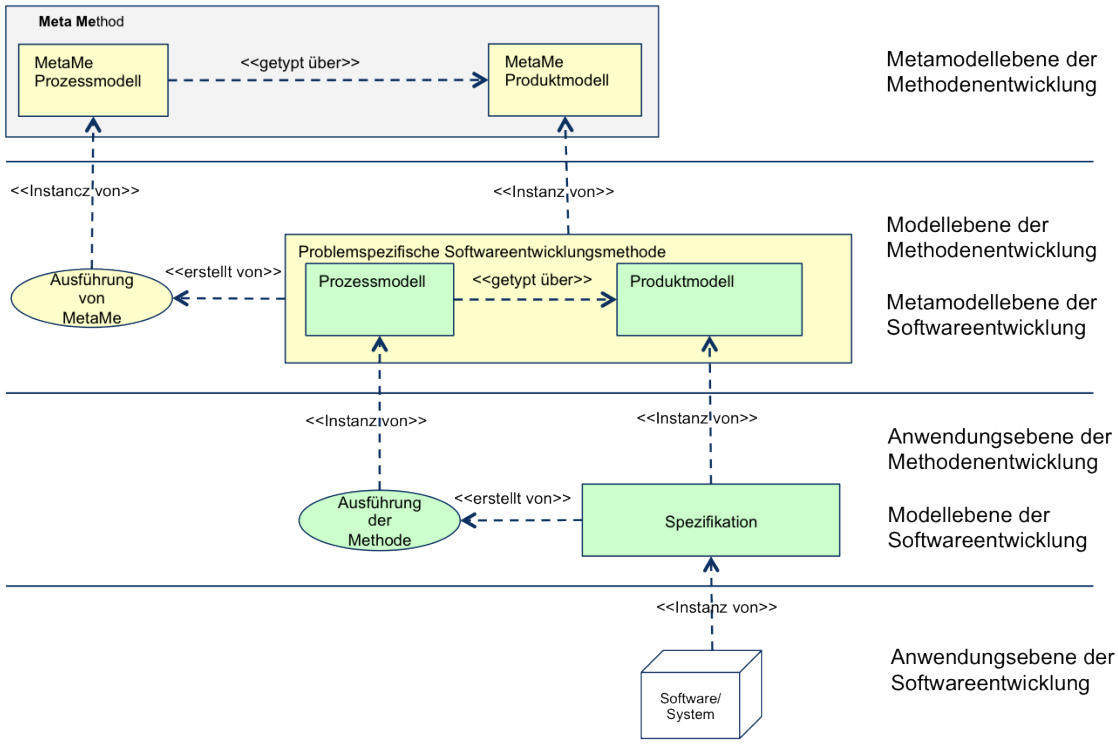


Abbildung 2.13: *Anwendung von MetaMe*

Anwendungskonzept

Bei der Methodenentwicklung wird das MetaMe-Prozessmodell ausgeführt, in-

dem die auf der Metamodellebene definierten Aktivitäten (vgl. 2.11) auf der Modellebene zur Erstellung der problemspezifischen Softwareentwicklungsmethode durchgeführt werden. Auf diese Weise wird ein neues Produkt- und Prozessmodell der problemspezifischen Methode auf Modellebene instanziiert. Auf der Anwendungsebene der Methodenentwicklung wird nun die problemspezifische Methode zur Erstellung der (Software-)Spezifikation durchgeführt, indem das Prozessmodell der problemspezifischen Methode ausgeführt wird. Die Spezifikation stellt letztendlich eine Instanz des Produktmodells der problemspezifischen Softwareentwicklungsmethode dar. Die Spezifikation einer Software oder eines Systems umfasst alle beschreibenden Aspekte des Zielprodukts. An dieser Stelle zählt beispielsweise auch der Programmcode zur Spezifikation. Die auf der Anwendungsebene der Methodenentwicklung erstellte Spezifikation befindet sich nun auf der Modellebene der Softwareentwicklung, denn eine weitere Instanziierung der Spezifikation führt zum realen Produkt und erst dann ist die Anwendungsebene der Softwareentwicklung erreicht. Die Metamodellebenen sind bei der Methodenentwicklung im Vergleich zur Softwareentwicklung also eine Ebene nach oben verschoben. Das bedeutet letztendlich für das Produktmodell der problemspezifischen Entwicklungsmethode, dass es das Metamodell der Spezifikation darstellt (vgl. 1.1).

Software- vs. Methoden- vs. Sprachentwicklung

MetaMe kombiniert die Methodenentwicklungsdomäne mit der Softwareentwicklungsdomäne, indem die Entwicklungsmethode spezifische Softwareentwicklungsmethoden hervorbringt. Ein weiterer Vorteil von MetaMe ist die integrierte Spracheentwicklung. Der Vorteil wird durch das folgende Beispiel einer ungenügenden Spezifikation motiviert.

Beispiel mangelhafter Spezifikation

Wir nehmen an, dass uns Schwächen in der Spezifikation und die daraus entstandenen Qualitätsschwächen des Produkts bekannt sind. Wenn Inhalte in der Spezifikation fehlen, besteht ein hohes Risiko, dass das Produkt in den nicht spezifizierten Bereichen fehlerhaft ist. Der Entwickler muss mittels seines Wissens und seiner Erfahrungen die Spezifikationslücken schließen. Gelingt das nicht adäquat, schleichen sich Fehler in die Instanziierung ein. Die Instanziierung besteht letztendlich aus der Überführung der Spezifikation, als Beschreibung der Software (Modellebene der Softwareentwicklung) in das reale System (Anwendungsebene der Softwareentwicklung). Ebenso verhält es sich mit der Spezifikation. Ist die Methode für die Erstellung der Spezifikation mit Schwachstellen versehen, ist es die Aufgabe des Spezifikationserstellers mit seiner Eigenverantwortung dafür zu sorgen eine gute Spezifikation zu erstellen. Umso vollständiger die Entwicklungs-

methode ist, umso vollständiger wird die aus der Entwicklungsmethode resultierende Spezifikation.

Sprachentwicklung mit MetaMe

Das Beispiel wird vervollständigt, indem die Domäne Sprachentwicklung in dem Anwendungskonzept visualisiert wird. Die Abbildung 2.14 stellt dar, wie MetaMe implizit die Sprachentwicklung integriert. Demnach kann die Methodenentwicklung mit MetaMe eine geeignete Sprache hervorbringen, denn das Produktmodell übernimmt die Rolle des Sprachmetamodells und definiert die Artefakttypen, also die Metamodellelemente der Sprache. Das ist letztendlich die abstrakte Syntax einer Sprache, deren konkrete Syntax dann in der Spezifikation genutzt werden kann.

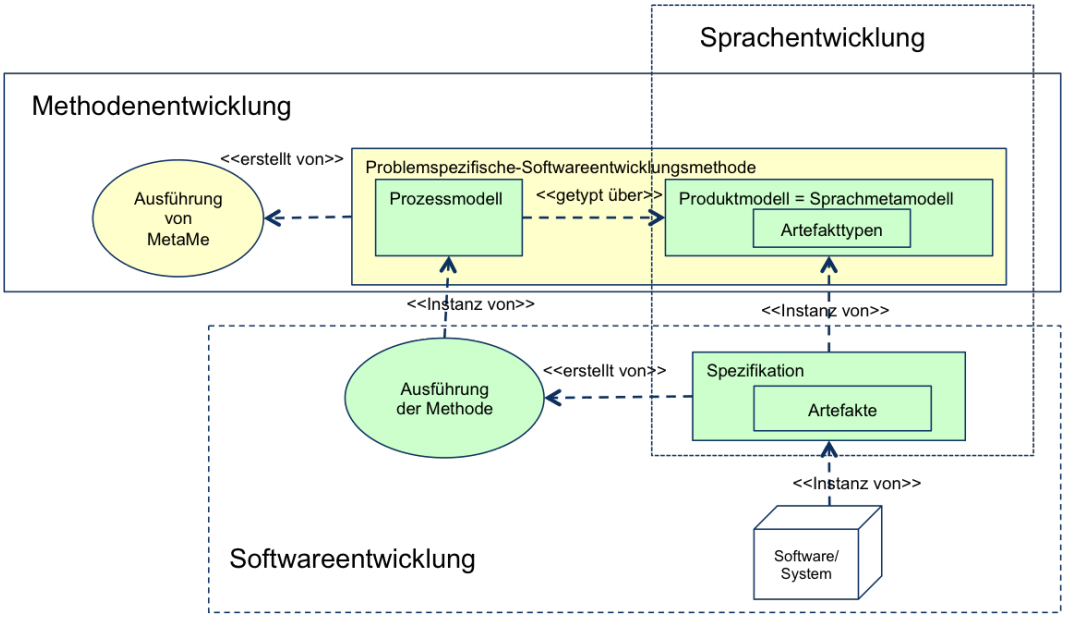


Abbildung 2.14: *MetaMe integriert die Sprachentwicklung*

2.4.4 Schwachstellen bei der Anwendung von MetaMe

MetaMe vs. Methoden-weiterentwicklung

In Abschnitt 2.3 werden die notwendigen Aufgaben des Methodenentwicklers aufgezeigt, die durchzuführen sind, wenn eine situationsgerechte Entwicklungsmethode entwickelt wird. Bildet MetaMe die Grundlage der Erstellung einer situationsgerechten Entwicklungsmethode, muss MetaMe den vorgestellten Vorgaben folgen. Dieser Abschnitt stellt die fehlende Ganzheitlichkeit von MetaMe für die situationsgerechte Methodenweiterentwicklung dar.

Methodenlebenszyklus

Nach Sauer besteht der Lebenszyklus einer (Software-)Entwicklungsmethode aus den Schritten *Erheben von Anforderungen an die Methode*, *Entwicklung der Methode*, *Einführung und Benutzung der Methode* sowie die *Evaluation der Methode*. Jedoch detailliert das Prozessmodell der Metamethode lediglich die Schritte der Aktivität *Entwicklung der Methode*, wie bereits in Abbildung 2.11 veranschaulicht worden ist.

Fehlende Situationsbestimmung

Die Bestimmung der Domäne und der Disziplinen ist die einzige Aktivität, die investigativen Charakter mit sich bringt und den Eindruck entstehen lässt zur Bestimmung einer Situation genutzt werden zu können. Das ist jedoch nicht der Fall. Die Domäne ist bei MetaMe fest auf *Softwareentwurf* gesetzt. Die Ermittlung der Disziplin wird lediglich für die Strukturierung der Softwareentwicklungsmethode genutzt und dient nicht der Situationsbestimmung (vgl. 2.4.2).

eingeschränkte Definition iterativer Methodenentwicklung

MetaMe definiert Iterationen der Methodenentwicklung, indem einer Evaluation nachgestellt ein neuer Entwicklungszyklus folgt (siehe Abbildung 2.11). Eine Definition über inkrementelle Teil-Projekte, wie in Abschnitt 2.3 definiert, ist in MetaMe nicht vorgesehen. Die Tätigkeiten zur Ermittlung der Ist-, Ideal und Soll-Situation sind nicht vorgesehen.

Weiteres Formalisierungspotenzial

In dieser Arbeit wird die Formalisierung von Spezifikationen als ein Optimierungspotenzial für Qualität beschrieben. Ebenso wie diese Optimierung für Produktspezifikationen gilt, muss sie auch Berücksichtigung bei der Beschreibung von MetaMe finden. Es liegt eine gute Basis vor, da das Produktmodell als Metamodell beschrieben ist und das Prozessmodell über das Produktmodell getypt ist. Die Beschreibung der im Prozessmodell definierten Schritte liegt im Gegensatz dazu nur in textueller, informeller Form vor.

fehlende Handlungsanweisungen

Aufgrund der allgemeinen Aktivitätsbeschreibung sind beliebige Instanziierungen des MetaMe-Produktmodells vorstellbar. Zum einen führt das zu einem umfangreichen Einsatzgebiet, aber zum anderen fehlt dadurch eine nötige Eingrenzung sinnvoller Instanziierungen. Die Integration von Erfahrungswerten oder die Berücksichtigung des Methodenkontexts und der Methodenanforderungen ist denkbar. Sauer schlägt die detailliertere Modellierung des Prozessmodells einer erstellten Softwareentwicklungsmethode mit Activity Pattern oder Transformationsregeln vor. Dieser Vorschlag ist bisher noch nicht auf sein MetaMe-Prozessmodell übertragen worden, um einem Methodenentwickler mit detaillierten Integritätsbedingungen, wie Best Practices oder No-Gos, Hilfestellung zu

leisten. Natürlich bedarf es zuerst der geeigneten Formalisierung oder Attributierung des Produktmodells, damit Invarianten oder Vor- und Nachbedingungen spezifiziert werden können.

frühe Analyseergebnisse

Bei einer ersten Verbesserung von MetaMe im Rahmen einer Diplomarbeit sind verschiedene Schwachstellen von MetaMe identifiziert worden (siehe [Sta09]). Es ist eine mit MetaMe erstellte Instanz einer Softwareentwicklungsmethode hinsichtlich einer gültigen Instanziierung des Metamodells für Vorgehensmodelle von Gutzwiller untersucht worden. Auf Basis des Vergleichs sind Anforderungen an das Prozessmodell von MetaMe gestellt worden. Es sind weitere Anforderungen, die auf Basis einer exemplarischen Ausführung des MetaMe-Prozessmodells, anhand eines theoretischen Beispiels, ermittelt wurde, vorgestellt worden. Die folgende Auflistung beschreibt die erkannten Optimierungspotenziale.

a) Die Produktkonzepte sollen den Typen *Strukturbeschreibung*, *Verhaltensbeschreibung*, *Struktur- und Verhaltensbeschreibung* oder *Sonstige* zugeordnet werden. Diese Anforderung steht in Bezug zur Eigenschaft *Separation of Concerns*. Die strikte Trennung von Struktur- und Verhaltensbeschreibung bringt Vorteile in der Übersichtlichkeit und Handhabung von komplexen Zusammenhängen. Insbesondere in der Softwareentwicklungsdomäne, geprägt durch die UML-Diagramme in Struktur- und Verhaltensdiagramme, ist dieses Vorgehen allgegenwärtig. In der Praxis wird keine hundertprozentige Trennung erreicht werden, darum gibt es die Mischform als Alternative. Eine erste Verbesserung ist jedoch nur die farbliche Markierung der Typen, welche nur die konkrete Syntax des Domänenmodells der Konzepte, aber nicht die abstrakte Syntax des MetaMe-Produktmodells betrachtet. Die Unterstützung zusätzlicher Typen von Produktkonzepten soll durch Erweiterung des MetaMe Produktmodells, also Anpassung der abstrakten Syntax, erreicht werden. Bereits in Abschnitt 2.3 sind zusätzliche Konzepttypen identifiziert worden, und zwar die unterschiedlichen Entwicklungskonzepte.
b) Die hierarchischen Strukturen von Konzepten sollen abgebildet werden können. Als Beispiel wird hier das Konzept Produktfunktion, welches in der konkreten Softwareentwicklungsmethode, die auch [ESS08] zugrunde liegt, angesehen. Es besteht aus GUI-Skizzen, Beschreibung von Standard- sowie Alternativabläufen und Szenarien. Die Optimierung ist durch eine Anpassung der konkreten Syntax vorgenommen worden. Der hierarchische Zusammenhang ist durch UML-Pakete darstellbar. Auch hier muss die Rückführung der Optimierung auf die abstrakte Syntax entsprechend durchgeführt werden.

c) Diverse Schwächen in der konkreten Syntax sind gefunden worden. Letztlich gibt dieser Sachverhalt einen Hinweis auf mangelhafte Festlegung der konkreten Syntax. Zur Lösung dieser Mängel muss zunächst definiert werden, welche Aspekte der abstrakten Syntax in welchen Zusammenhang stehen. Das bedeutet letztendlich die Artefakte von MetaMe formalisiert zu beschreiben. Die folgende Auflistung benennt die MetaMe eigenen Kandidaten für Artefakttypen:
   - Domänenmodell der Produktfunktionen
   - Sprachauswahl
   - Domänenmodell der Artefakttypen
   - Entwicklungsprozess, auch als Vorgehen bezeichnet
   - Werkzeug und Nutzungskonzept

d) Aufgrund der Tatsache, dass einige Elemente von Gutzwillers Metamodell in MetaMe nicht direkt vorhanden waren, ist das MetaMe-Vorgehen zur Erstellung des Prozessmodells einer zu entwickelnden Entwicklungsmethode, auf Basis von SPEM und deren Konzept Breakdown-Strukturen, detailliert worden. Die Erstellung von Breakdown-Strukturen ermöglicht die Identifikation von Aktivitäten und Rollen sowie eine sinnvolle Abarbeitungsreihenfolge der Aktivitäten festzulegen. Die Optimierung ist die Definition von Aktivitäten und Rollen nach Gutzwiller, wobei die Aktivitäten die Eigenschaften Strukturiertheit, Dekomposition und Hierarchie erfüllen. Das Vorgehen beschränkt sich lediglich auf die Beschreibung der MetaMe-Aktivität *Definiere Prozessmodell*. Weitere Vorgehensbeschreibungen sind notwendig. Konkret fehlen Vorgehensbeschreibungen für die folgenden MetaMe-Aktivitäten:
   - Definiere Domäne und Disziplinen
   - Erstelle Domänenmodell der Produktkonzepte
   - Führe Sprachauswahl durch
   - Erstelle Domänenmodell der Artefakttypen
   - Erstelle Werkzeug und Nutzungskonzept

Anpassung (engl.: Tailoring)

Sauer wendet MetaMe in [Sau11a] auf die Entwicklung von Advanced User Interfaces (AUI) an. Eine große Anzahl für die AUI-Entwicklung relevanter Artefakte sind Modelle. In dem Artikel wird eine Spezialisierung der Artefakte des MetaMe-Produktmodells auf Modelle beschrieben. Weiterhin wird das Vorgehen der modellgetriebenen Entwicklung (engl. Model Driven Development, MDD) in den Entwicklungsprozess integriert. Die Spezialisierung von Aktivitäten des MetaMe-Produktmodells auf Transformationen und weiterhin auf Modelltransformationen ist notwendig. Darauf aufbauend muss auch eine Anpassung des

MetaMe-Prozessmodells stattfinden. Die Beschreibung wird um die Festlegung von Modellen und Spezifikation von Modelltransformationen erweitert. Eine Spezialisierung von MetaMe auf die Domäne AUI ist definiert. Demnach ist MetaMa domänenspezifisch anpassbar. Doch für die situationsgerechte Methodenentwicklung muss die Anpassung generisch, abhängig von der Situation, ermöglicht werden. Es ist auch möglich von der Selbst-Anpassung von MetaMe zu sprechen. Hier ist es vorteilhaft zu verstehen, welche Schritte des MetaMe-Prozessmodells angepasst werden können und wo Erweiterungen stattfinden dürfen. Ebenso ist es sinnvoll die Erweiterung für die MDDAUI als optional zu definieren. Sind in einer Entwicklungsdomäne Modelle in der Entwicklung ähnlich wichtig wie in der MDDAUI-Domäne, wird die MetaMe-Erweiterung benutzt. Ansonsten werden entsprechende Schritte des MetaMe-Prozessmodells ausgelassen.

Werkzeug vs. Sprache

In [Sau11a] wird ebenfalls ein Zusammenhang zwischen Sprache und Werkzeug festgestellt. Ein Werkzeug definiert die Benutzung einer Sprache. Wird im MetaMe-Prozessmodell bei der Bildung des Domänenmodells der Artefakttypen einem Konzept eine Sprache zugeordnet, hat dies bereits Auswirkungen auf einsetzbare Werkzeuge. Die Werkzeuge werden bei MetaMe erst später, und zwar nachdem ein Entwicklungsprozess zur Erstellung der Artefakte definiert worden ist, dem Entwicklungsprozess unterstützend zugeordnet. Nehmen wir an, dass bereits früh im Methodenentwicklungsprozess Anforderungen für bestimmte Werkzeuge vorhanden sind. Dann ist zu überprüfen, ob wir mit MetaMe auch frühzeitig Rücksicht darauf nehmen können.

fehlende linguistische Eigenschaften

Insbesondere bei der Anwendung der zu erstellenden Modelle (Prozess- und Produktmodell) als Instanz des MetaMe-Produktmodells ist es hinderlich, dass das MetaMe-Produktmodell lediglich ein ontologisches Metamodell, also eine fachliche Generalisierung, darstellt. Beispielsweise wird im MetaMe-Anwendungskonzept (Abschnitt 2.4.3) der Zusammenhang zwischen Prozess- und Produktmodell über eine *getypt über* Beziehung beschrieben. Soll diese Typbeziehung nun in einem Anwendungsbeispiel beschrieben werden, müsste im MetaMe-Produktmodell neben der vorhandenen Typdefinition noch eine Instanz- oder Typbeziehung innerhalb des MetaMe-Produktmodells definiert sein, vergleichbar mit der Klassen-Objekt-Beziehung der UML (siehe Abbildung 2.15).

Beispiel linguistischer Instanziierung

In der Abbildung 2.15 sind zwei Arten von Instanzbeziehungen zu erkennen, linguistisch und ontologisch (vgl.: [AK03]). Die linguistische Instanzbeziehung besteht zwischen der Klasse *Video* auf der Modellebene und der Klasse *Class*

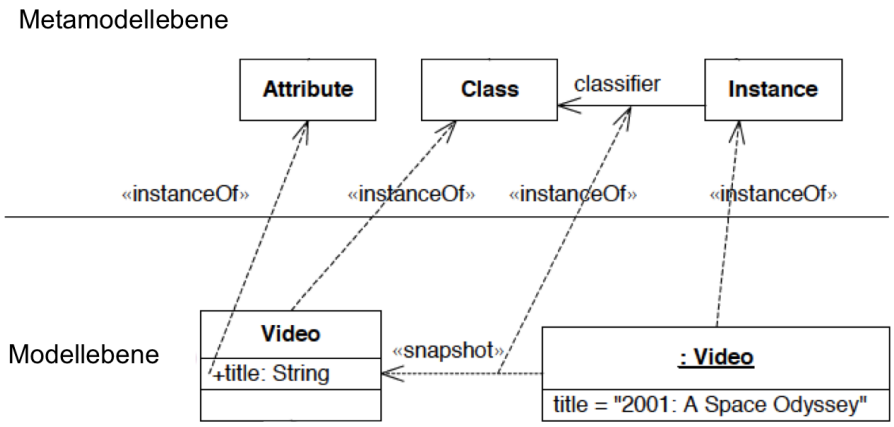


Abbildung 2.15: *Klasse-Objekt-Beziehung der UML (vgl. [Obj11b] Seite 20)*

auf Metamodellebene sowie der Instanz *:Video* auf der Modellebene und der Instanz *Instance* auf Metamodellebene. Eine *classifier*-Beziehung als ontologische Instanzbeziehung ist zwischen der Klasse *Class* und Instanz *Instance* auf Metamodellebene definiert, so dass die Instanz *:Video* über die Klasse *Video* getypt ist. Solche ontologischen Instanzbeziehungen werden im MetaMe-Produktmodell nicht definiert. Die Instanziierung des Metamodells der Metamodellebene auf Modellebene der Methodenentwicklung ist konzeptionell definiert, aber nicht im Sinne einer linguistischen Instanziierung durchführbar. Neben den fehlenden ontologischen Beziehungen fehlen auch die expliziten Typ-Definitionen für das Produkt- und Prozessmodell. Darum ist das MetaMe-Produktmodell nicht als Sprachmodell nutzbar, um direkt Methoden über das MetaMe-Produktmodell zu beschreiben.

Prozess- und Produktmodell

MetaMe sieht eine Trennung von Produkt- und Prozessmodell vor. Das MetaMe-Produktmodell lässt aber eine explizite Kennung oder Trennung von Prozessmodell- und Produktmodell-Elementen vermissen.

Zusammenfassend stellt die folgende Auflistung die erkannten Optimierungspotenziale dar:

- Integration der Situationsbestimmung
- Formalisierung des Vorgehens
- Definition detaillierter Handlungsanweisungen für den Methodenentwickler
- Erweiterung der Konzepttypen im MetaMe-Produktmodell

- Unterstützung von hierarchischen Konzepten im MetaMe-Produktmodell
- Definition der konkreten Syntax von MetaMe-Artefakten
- Detaillierung des MetaMe-Prozessmodells
- Definition von MetaMe-Anpassung an bestimmte Vorbedingungen
- Frühere Berücksichtigung von Werkzeugabhängigkeiten ermöglichen
- Detaillierung des MetaMe-Produktmodells als ein linguistisches Metamodell
- Unterteilung des MetaMe-Produktmodells in Prozessmodell und Produktmodell

nächste Schritte

Die dargestellten Optimierungspotenziale werden in der Überführung von MetaMe nach MetaMe++ berücksichtigt und werden in den folgenden Abschnitten und Kapiteln berücksichtigt. Insbesondere der letzte Punkt der *Überführung des MetaMe-Produktmodells in ein linguistisches Metamodell* ist für die Definition einer Sprache zur Beschreibung von Methoden wichtig. Die Sprachdefinition ist Inhalt des folgenden Abschnitts.

2.5 Lösungsbaustein: MetaMe-Methodenbeschreibung (MMMB)

Aufgabe

Dieser Abschnitt befasst sich mit der Teilaufgabe der Dissertation 2 *Definition einer MetaMe kompatiblen Methodenbeschreibung*.

Methoden-beschreibung

Bei der Methodenweiterentwicklung werden an zwei Stellen Methodenbeschreibungen benötigt. Zum einen für die Beschreibung der Ist-Methode in der Phase *Ist-Analyse* und zum anderen beim modellbasierten Entwurf der Soll-Methode in der Phase *Konzeption* (siehe Abschnitt 2.3).

MetaMe kompatibel

Ist die Methodenbeschreibung kompatibel zum MetaMe-Anwendungskonzept (siehe Abschnitt 2.4.3), heißt das, die Beschreibung einer Methode besteht aus Prozess- und Produktmodell. Das Produktmodell hat die Eigenschaft das Metamodell einer Spezifikation zu sein. Die Ausführung des Prozessmodells erstellt eine Spezifikation konform zum Produktmodell (siehe Abbildung 2.13).

ein Sprachmetamodell wird benötigt

Für die Methodenbeschreibung wird eine Methodenbeschreibungssprache benötigt. Eine geeignete Methodenbeschreibungssprache besteht aus einem

Sprachmetamodell, welches die abstrakte Syntax der Sprache definiert, einer Semantik und einer konkreten Syntax.

MetaMe-Produktmodell

Im MetaMe-Anwendungskonzept ist vorgesehen, dass das MetaMe-Produktmodell ein Sprachmetamodell darstellt. Allerdings ist dieses aus folgenden Gründen als solches nicht verwendbar (vgl. Abschnitt 2.4.4).

- Das MetaMe-Produktmodell stellt keine konkreten Metamodellelemente für das Produkt- und Prozessmodell dar.
- Das MetaMe-Produktmodell macht keinen Unterschied zwischen einem Artefakttyp und Artefakt, als Instanz des Artefakttyps. Deshalb ist es nicht möglich beide Elemente auf derselben Modellebene (Methodenbeschreibung) darzustellen.
- Das MetaMe-Produktmodell definiert abstrahierte Methodenelemente lediglich als Klassen, detailliert aber nicht deren Eigenschaften und bietet dadurch nicht die notwendige Granularität, um als Sprachmetamodell benutzt zu werden.

Methoden-metamodell

In Abschnitt 2.1 ist auf Basis der Grundlagen von Entwicklungsmethoden ein Methodenmetamodell definiert worden. Es stellt eine Konkurrenz zum MetaMe-Produktmodell dar. Im Gegensatz zum MetaMe-Produktmodell werden die Elemente Produkt- und Prozessmodell sowie Artefakttyp und Artefakt als Elemente einer Methode definiert. Dadurch eignet es sich besser als Sprachmetamodell als das MetaMe-Produktmodell.

Anforderungen am Sprachmetamodell

Weiterhin muss sichergestellt werden, dass das Methodenmetamodell den Ansprüchen an einem Sprachmetamodell und den Anforderungen zur Benutzung im MetaMe-Anwendungskonzept genügt. Dafür wird in einem ersten Schritt das Methodenmetamodell mit dem MetaMe-Produktmodell verglichen, um festzustellen, welche Aspekte aus dem MetaMe-Produktmodell im Methodenmetamodell fehlen. In einem zweiten Schritt wird das Methodenmetamodell in ein Sprachmetamodell für die MetaMe-Methodenbeschreibung (MMMB) überführt. Dabei wird das Methodenmetamodell um die fehlenden Aspekte des MetaMe-Produktmodells erweitert und weitere identifizierte Anforderungen an MetaMe sowie generelle Anforderungen an Sprachmetamodelle berücksichtigt (siehe Abbildung 2.16).

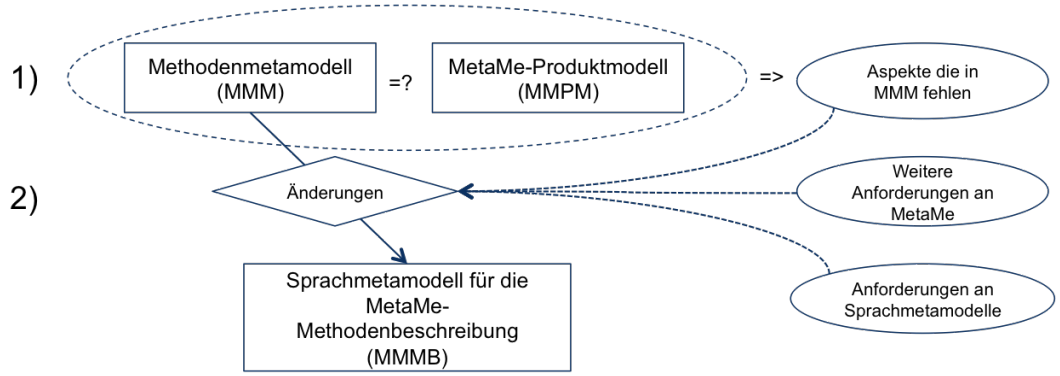


Abbildung 2.16: *Übersicht des Vorgehens zur Erstellung des Sprachmetamodells*

Die relevanten Unterschiede zwischen dem Methodenmetamodell und dem MetaMe-Produktmodell werden in der folgenden Tabelle dargestellt. Unterschiede

Tabelle 2.1: *Vergleich Methodenmetamodell mit MetaMe-Produktmodell*

MetaMe-Produktmodell	**Methodenmetamodell**
Domänenmodell	
Das Domänenmodell wird explizit als Klasse *Domain Model* dargestellt, welches Konzepte (*Concept*) aggregiert.	Das Domänenmodell wird nicht explizit dargestellt. In der Praxis erweist sich die grundlegende Tätigkeit der Definition eines Domänenmodells und nachträgliche Überführung in ein Artefakttypmodell als umständlich, da bei der Überführung einem Konzept lediglich eine Sprache zugewiesen wird und dadurch ein Artefakttyp entsteht. Das Methodenmetamodell erlaubt es Domänemodelle als Artefakttypmodelle zu erstellen, bei denen Artefakttypen, ohne zugewiesene Sprache, erlaubt sind.
Transformationen	
Es werden Modelle als Unterklasse von *Artifact* und Transformationen als Unterklasse von *Work* definiert. Auf diese Weise ist es möglich Transformationen über Artefakttypen zu beschreiben.	Es gibt keine expliziten Elemente zur Darstellung von Transformationen. Der Spezialfall wird vom Methodenmetamodell in dieser Arbeit nicht explizit unterstützt.
Prozessbeschreibung	

Es werden die Klassen *Work*, *Activity*, *Task*, *Process* und *ActivityStep* definiert.	Es wird eine reduzierte Menge an Prozessbestandteilen (*Aktivität* und *ZusammengesetzteAktivität*) definiert, die jedoch ausreichend zur Beschreibung von Prozessen ist.
Domäne und Organisation	
Explizite Definition der Klasse *Domain* und *Organization*.	In dieser Arbeit wird von der Weiterentwicklung von Organisations- und Domänen-spezifischen Methoden ausgegangen. Das bedeutet, dass eine Methode implizit für eine Domäne und für eine Organisation definiert wird.
Trennung von Artefakttyp und Artefakt	
Es gibt eine implizite Unterstützung der Trennung über Instanziierung. Artefakt und Instanz des Artefakts liegen auf unterschiedlichen linguistischen Ebenen	Implizite Unterstützung als ontologische Instanz auf einer linguistischen Ebene.
Sichten	
Explizite Definition des Elements *AspectView* als Eigenschaft von Konzepten erlaubt die Erstellung verschiedener Sichten auf das Domänenmodell.	Nicht vorhanden.

Änderungen

Im Vergleich zum MetaMe-Produktmodell fehlt lediglich die Unterstützung Aspekt-orientierter Sichten im Methodenmetamodell. Es wird die folgende Änderung abgeleitet.

Tabelle 2.2: *Änderungen an MMM aufgrund fehlender Aspekte aus MMPM*

Unterstützung Aspekt-orientierter Sichten.
Dem Element *Artefakttyp* wird ein Attribut *Konzepttyp* hinzugefügt, dessen Werte verschiedene Aspekte beschreiben können. Dadurch wird es möglich Elemente des Produktmodells auf Basis der Aspekte zu filtern.

Weitere Anforderungen an MetaMe sind bereits in der Diskussion der Schwachstellen von MetaMe (Abschnitt 2.4.4) sowie in der Darstellung des Vorgehens der Methodenweiterentwicklung (Abschnitt 2.3) erkannt worden. Einige der Anforderungen beziehen sich auf die Darstellung von Methoden. Die daraus abgeleiteten Änderungen werden in der Tabelle 2.3 dargestellt.

Tabelle 2.3: *Änderungen am MMM aufgrund weiterer Anforderungen an MetaMe*

Definition Aspekt-orientierter Sichten.

Definition der folgenden Aspekte:
- Entwicklungskonzept
- Produktkonzept mit Untertypen
  - Struktur
  - Verhalten
  - Struktur und Verhalten
Bereitstellung der Werte für das Attribut *Artefakttyp.Konzepttyp* als *Enumeration*.

Detaillierung der Beziehungen zwischen Artefakttypen.
Hierarchien, das sind strukturelle Verfeinerungen, werden bereits im Methodenmetamodell durch die Aggregationsbeziehung dargestellt. Darüber hinaus gibt es fachliche Verfeinerungen und einfache Abhängigkeiten zwischen Artefakttypen. Eine fachliche Verfeinerung besteht beispielsweise zwischen Zielen und Anforderungen oder Analysemodell und Entwurfsmodell. Die Beziehungen überschreiten verschiedene Ebenen der Abstraktion (Abbildung 2 in [ESS08]). Weiterhin können einfache Abhängigkeiten zwischen Artefakttypen bestehen. Es werden die Assoziationen *FachlicheVerfeinerung* und *Abhängigkeit* zwischen Elementen vom Typ *Artefakttyp* definiert.
Darstellung der Abarbeitungsreihenfolge von Aktivitäten.
Ein Objektfluss kann bereits über *Eingabe* und *Ausgabe* von Artefakten modelliert werden. Zusätzlich soll eine Kontrollflussmodellierung zwischen Aktivitäten ermöglicht werden. Es wird die Assoziation *Nachfolger* zwischen Elementen vom Typ *Aktivität* definiert.
Definition von Zuständen für ein Artefakt.
Jeder Instanz eines Artefakts im Prozessmodell soll ein Zustand, welcher auf Typebene durch das Lebenszyklusmodell definiert wird, zugewiesen werden können. Für ein Element *Artefakt* wird das Attribut *Zustand* vom Typ *Zustand* definiert. Der Typ *Zustand* wird explizit als Element des Metamodells definiert. Er wird im Lebenszyklusmodell definiert.

Bei der Sprachdefinition über Sprachmetamodelle gibt es einige Anforderungen und verschiedene Möglichkeiten der Design-Entscheidung, die in der folgenden Auflistung beschrieben werden.

- Assoziationen sollen eindeutig benannt werden [FV04].
- Sämtliche Elemente, die keine Assoziationen darstellen, sollen einem aggregierenden Modellelement zugewiesen werden.
- Unidirektionale Assoziationen können als Attribute der referenzierenden Elemente dargestellt werden [ASB04].

- Einfache Elemente des Metamodells, ohne weitere strukturelle Information, können als Attribute dargestellt werden.

Auf Basis der Auflistung werden folgende Änderungen definiert.

Tabelle 2.4: *Änderungen an MMM aufgrund von Anforderungen an Sprachmetamodelle*

Modellelemente aggregieren.
Die Elemente *Rolle*, *Meilenstein* und *Hilfsmittel* werden dem Element *Prozessmodell* mittels einer Aggregation angehängt.
Eindeutige Benennung von Assoziationen.
Jede bidirektionale Assoziation bekommt einen eindeutigen Namen.
Unidirektionale Assoziationen werden über Attribute definiert.
Das Element *Artefakt* bekommt aus deskriptiven Gründen das Attribut *Typ*. Die entsprechende Assoziation ist bei einer Umsetzung der Sprache eventuell nur implizit vorhanden, das Attribut wird auf jeden Fall explizit sichtbar sein. Weiterhin bekommt das *Artefakt* das Attribut *Zustand*.
Element versus Attribute.
Die Elemente *Konzept* und *Sprache* werden als Attribut des Elements *Artefakttyp* dargestellt, da keine weiteren strukturellen Informationen für die Elemente definiert sind.

Das Methodenmetamodell wird auf Basis der definierten Änderungen aus den Tabellen 2.2, 2.3 und 2.4 in das Sprachmetamodell überführt. Das resultierende Sprachmetamodell wird in Abbildung 2.17 dargestellt.

Realisierung

Auf Basis des Sprachmetamodells wird nun eine Sprachdefinition für die Sprache MetaMe-Methodenbeschreibung (MMMB) durchgeführt. Das Methodenmetamodell stellt die abstrakte Syntax in Form eines Metamodells dar. Zur Definition der Semantik und der konkreten Syntax wird ein UML-Profil erstellt, da dadurch die praktische Bereitstellung der Sprache gewährleistet werden kann. Viele UML-Modellierungswerkzeuge unterstützen UML-Profile. Aus diesem Grund ist ein UML-Profil schnell und einfach in den verschiedensten Werkzeugumgebungen zu realisieren. Eine erste Realisierung wird für das UML-Modellierungswerkzeug Enterprise Architekt der Firma SparxSystems Software GmbH[5] bereitgestellt.

Vorgehen

Ein Vorgehen für die Definition von UML-Profilen wird in [FV04] dargestellt. Dessen Ergebnis und eine Übersicht über die Sprachdefinition finden sich in Abschnitt A.3 des Anhangs.

5 http://www.sparxsystems.de, letzter Aufruf: 2015-01

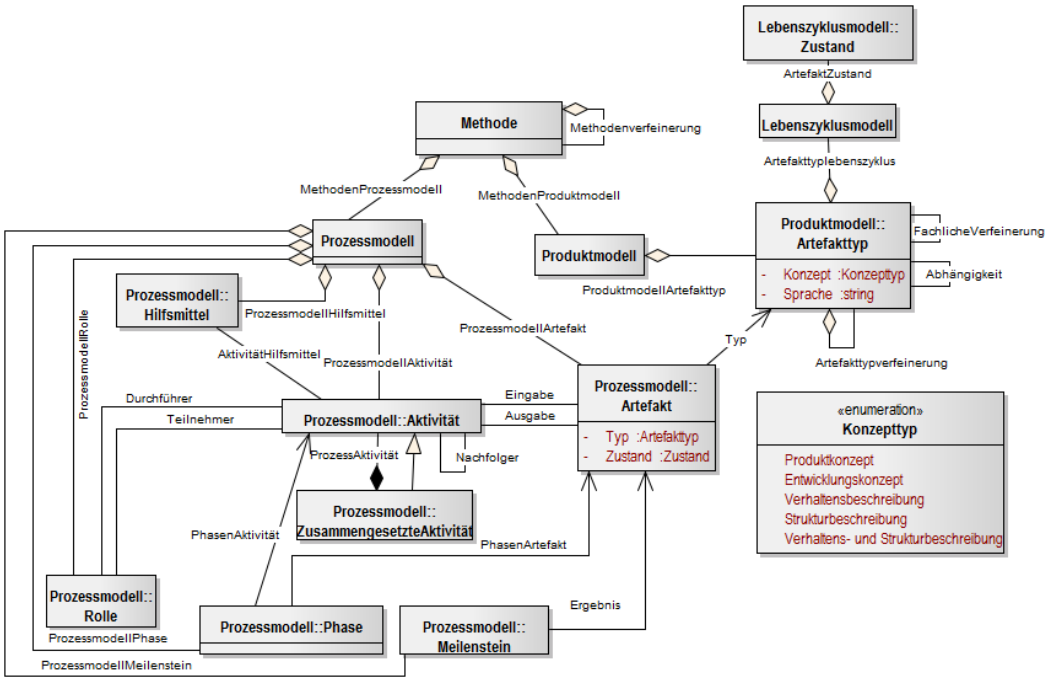


Abbildung 2.17: *Sprachmetamodell für MMMB*

Dieser Abschnitt über die Sprachentwicklung endet mit einem konstruierten Beispiel der Sprache MMMB, welches alle definierten Sprachkonstrukte darstellt (siehe Abbildung 2.18). Sprachreferenz

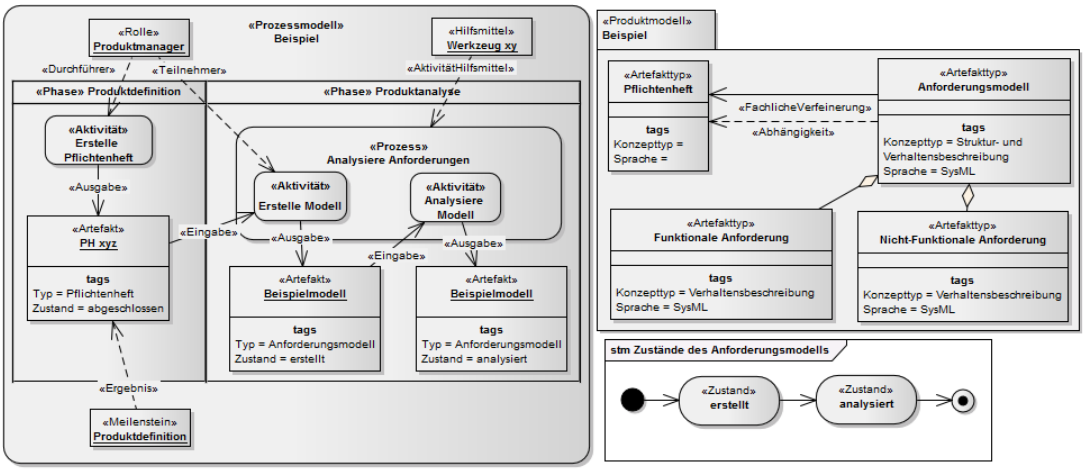


Abbildung 2.18: *Sprachreferenz MMMB*

Die Sprachreferenz zeigt ein *Prozessmodell*, *Produktmodell* und ein *Lebenzy-* Erläuterung der Sprachreferenz

klusmodell. In dem *Prozessmodell* sind die zwei Elemente vom Typ *Phase Produktdefinition* und *Produktanalyse* definiert. Die *Produktdefinition* beschreibt die *Aktivität Erstelle Pflichtenheft*, welche das abgeschlossene Pflichtenheft *PH xyz* als *Ausgabe* produziert. Die *Phase Produktanalyse* ist komplexer und enthält die *ZusammengesetzteAktivität Analysiere Anforderungen*, die aus den Aktivitäten *Erstelle Modell* und *Analysiere Modell* besteht. Erstere beschreibt die Überführung der Inhalte des Pflichtenhefts in ein Beispielmodell. Letztere entwickelt das Beispielmodell vom *Zustand erstellt* weiter zum *Zustand analysiert*. Die *Rolle Produktmanager* ist definiert als durchzuführende Entität der Erstellung des Pflichtenhefts und teilnehmende Entität bei der Erstellung des *Beispielmodells*. Für die Durchführung der *Aktivität Analysiere Anforderungen* wird das *Hilfsmittel Werkzeug xy* definiert. Der *Meilenstein Produktdefinition* wird über das abgeschlossene Pflichtenheft definiert. Die Typen, der im Prozessmodell genutzten Artefakte, werden im *Produktmodell* definiert. Es gibt die *Artefakttypen Pflichtenheft* und *Anforderungsmodell*. Neben der *FachlichenVerfeinerung* und der beschriebenen *Abhängigkeit* zwischen den beiden Elementen wird zusätzlich die *Artefakttypverfeinerung* skizziert, indem die Elemente vom Typ *Artefakttyp Funktionale Anforderung* und *Nicht-Funktionale Anforderung* beschrieben werden. Das *Lebenszyklusmodell* definiert die Zustände des Anforderungsmodells.

2.6 Zusammenfassung

Grundlagen

In diesem Abschnitt sind in einem ersten Schritt Definitionen für Methoden und Methodenelemente vorgestellt worden. Das Resultat ist ein zu den Definitionen konformes Methodenmetamodell (vgl. Abschnitt 2.1). In einem zweiten Schritt ist der Begriff Methodenweiterentwicklung in das Thema SME eingegliedert (vgl. Abschnitt 2.2) worden. Auf Basis der Festlegungen in den ersten beiden Abschnitten konnten die Aufgaben des Methodenentwicklers bei der Methodenweiterentwicklung in Abschnitt 2.3 detailliert werden.

MetaMe vs. Grundlagen vs. Methodenweiterentwicklung

Der Abschnitt 2.4 stellt die bisher genutzte Metamethode zur Erstellung von Softwareentwicklungsmethoden vor. Des Weiteren konnten große Ähnlichkeiten zwischen dem MetaMe-Produktmodell und dem entwickelten Methodenmetamodell herausgearbeitet werden. Das führt zu der Möglichkeit, das bisherige MetaMe-Produktmodell durch das Methodenmetamodell zu ersetzen. Während das MetaMe-Produktmodell als ontologisches Modell definiert war, bietet das Me-

thodenmetamodell die Möglichkeit als linguistisches Metamodell benutzt zu werden. Weiterhin wurden auf Basis der definierten Aufgaben des Methodenentwicklers eine Menge von Optimierungspotenzialen von MetaMe identifiziert, deren Umsetzung notwendig wird, wenn das MetaMe-Konzept für die Methodenweiterentwicklung genutzt werden soll.

MMMB

In Abschnitt 2.5 wird der Vorschlag der Ersetzung des MetaMe-Produktmodells durch das Methodenmetamodell teilweise umgesetzt. Auf Basis des Methodenmetamodells ist eine Sprache zur Beschreibung von Methoden *MMMB* vorgestellt worden. Damit ist die Teilaufgabe der Aufgabenstellung der Dissertation 2 *Definition einer MetaMe kompatiblen Methodenbeschreibungssprache* teilweise erfüllt. Eine vollständige Umsetzung geschieht, wenn bei der Weiterentwicklung von MetaMe nach MetaMe++ das MetaMe-Produktmodell durch das Methodenmetamodell auch formal ersetzt wird.

Ausblick

Auf Basis dieses Kapitels sind die Phasen der Methodenweiterentwicklung weiter detailliert worden. Die Phase *Ist-Analyse* ist geprägt durch die Bestimmung eines Ist-Zustands, der aus einer Ist-Methode und dem Methodenkontext besteht. Zur Beschreibung von Ist-Methoden wird nun die Sprache MMMB genutzt. Die Anwendbarkeit dieser Sprache wird im folgenden Kapitel 3 überprüft, indem für zwei Fallbeispiele der Server-System-Entwicklung die Beschreibung der Ist-Methoden mit MMMB durchgeführt wird. Die Definition des Methodenkontexts bedarf weiterer Konkretisierung des Begriffs *Situation*, die im Kapitel 4 stattfindet. Das hauptsächliche Thema der Verbesserungs-Analyse ist die Erstellung von Methodenanforderungen (siehe Kapitel 5). In der Phase *Konzeption* soll MetaMe genutzt werden. Dafür besteht jedoch Anpassungsbedarf von MetaMe an die Methodenweiterentwicklung, der im Kapitel 6 thematisiert wird.

3 Server-System-Entwicklung

Anwendungsbeispiele

In dem vorigen Kapitel haben wir uns mit den Grundlagen der Methodenweiterentwicklung beschäftigt. Nun kommen wir zu den Anwendungsbeispielen der Methodenweiterentwicklung. Die Anwendungsbeispiele liegen in der Domäne der Server-System-Entwicklung und basieren auf konkreten Forschungs- und Entwicklungsprojekten, die vom s-lab – Software Quality Labbeim Projektpartner Fujitsu Technology Solutions GmbH(FTS) durchgeführt worden sind.

Ziele

Die Entwicklung der Server-Systeme bei FTS birgt verschiedene Schwierigkeiten. Zum einen werden Fehler in den Systemen erst in späten Phasen der Systementwicklung, bei der Qualitätssicherung, festgestellt. Das führt zu aufwendigen und kostspieligen Fehlerbehebungsmaßnahmen. Zum anderen herrscht Unzufriedenheit über die Qualität von in Auftrag gegebenen zugelieferten Softwareelementen. Das führt wiederum zu einem großen Aufwand in der Diskussion mit den Zulieferern, um die Richtigstellung der Sachverhalte oder zur Ermittlung der Partei, die den Fehler verursacht hat. Insbesondere diese beiden Beispiele drängen FTS zur Verbesserung und Weiterentwicklung ihrer Entwicklungsmethode mit dem Ziel früher Fehler erkennen zu können und die Eindeutigkeit der Spezifikationsdokumente zu verbessern. Zu Beginn des Kapitels wird in Abschnitt 3.1 dieser Sachverhalt detailliert, die Gegebenheiten im Umfeld der Server-System-Entwicklung aufgezählt und die entsprechenden Herausforderungen definiert, die in einer Beschreibung von weiteren Zielen der Server-System-Entwicklung enden. Diese Ziele werden in den Anwendungsbeispielen der Methodenweiterentwicklung betrachtet.

Grundlagen

Bei der Methodenweiterentwicklung wird in einem ersten Schritt die Ist-Situation definiert, um in einem zweiten Schritt die Analyse der Schwachstellen auf Basis der Ist-Situation durchführen zu können. Für die Ermittlung der Ist-Situation und der Schwachstellen ist Domänenwissen erforderlich. Zum besseren Verständnis der Domäne Server-System-Entwicklung werden die verschiedenen Aufgaben und Arten der Server-Systeme (siehe Abschnitt 3.2) und die Aufgaben

des System Managements (siehe Abschnitt 3.3) dargestellt. Abschließend lassen sich strukturelle Eigenschaften von Server-Systemen ableiten (Abschnitt 3.4).

konkrete Methoden-weiterentwicklung

Im letzten Abschnitt widmet sich dieses Kapitel der konkreten Methodenweiterentwicklung im Forschungs- und Entwicklungsprojektbei FTS. Da bisher nur die Darstellung der Ist-Methode aus der Phase Ist-Analyse der Methodenweiterentwicklung definiert ist, wird diese Aufgabe in Abschnitt 3.6 mit Hilfe der Erkenntnisse aus dem Forschungs- und Entwicklungsprojektverfeinert und exemplarisch durchgeführt. Die Beschreibung des Methodenkontexts und der darauf aufbauenden Erhebung von Methodenanforderungen wird in Kapitel 4 und 5, nachdem geeignete Grundlagen der Methodenweiterentwicklung definiert worden sind, vorgestellt. Die Überführung der Ist-Methode auf Basis der Methodenanforderungen in eine Soll-Methode wird in Kapitel 6 vorgestellt.

3.1 Herausforderungen und Ziele der Server-System-Entwicklung

Dieser Abschnitt zählt eine Reihe an Eigenschaften auf, die bei der Server-System-Entwicklung berücksichtigt werden müssen. Die daraus resultierenden Herausforderungen werden in Ziele für die Weiterentwicklung der Server-System-Entwicklung überführt.

Server-Systeme

Ein Server-System ist ein Hardware-System zur Ausführung von Server-Software. Neben der Rechen- und Speicherleistung stellen moderne Server-Systeme Schnittstellen zur Administration und Konfiguration der Hard- und Software bereit. Die Verwaltung eines Server-Systems geschieht über diese Schnittstellen und ist entweder unabhängig von einem installierten Betriebssystem (out-of-band) oder nutzt dieses ebenfalls zur Verwaltung des Systems (in-band). Die out-of-band-Funktionalität wird in der Regel von einem Baseboard-Management-Controller (BMC), ein spezieller Mikrocontroller auf dem Server-Mainboard, bereitgestellt. Server-Systeme reichen von einfachen Rack-Systemen bis zu modularen Server-Systemen. Einzelne Rack-Server können in einem dafür vorgesehenen Einbaurahmen (Rack) installiert werden und mit einer externen Verkabelung beliebig miteinander verbunden werden. Ein modulares Server-System schließt eine eine Menge an modularen Servern mit weiteren Infrastrukturkomponenten, wie Stromversorgung und Kühlkomponenten, in einer proprietären, festgelegten In-

frastruktur (Modular-Chassis), zusammen. Konfigurierbare Eigenschaften in solchen Systemen reichen vom Eintragen von „Asset-Informationen“, wie Inventarnummer einzelner Hardware-Komponenten, über die Möglichkeit der Installation von Systemsoftware, wie Betriebssysteme oder Firmware, bis hin zum Einstellen komplexen, adaptiven Verhaltens, beispielsweise der Lüfter-Steuerung oder Maßnahmen zur Fehlerbehebung. Die Konfigurationseigenschaften sind vielfältig. Zum einen ist das System in unterschiedlichen Einsatzgebieten nutzbar und muss dafür entsprechend eingestellt werden. Zum anderen können automatisierte Aufgaben konfiguriert werden, die zur Vereinfachung der Verwaltung der Server-Systeme führen. Das ist für den Systemadministrator hilfreich, der immer größere Mengen an Servern betrachten muss.

Hohe Anzahl an Servern

Der Webhoster 1&1 besitzt nach eigenen Angaben bereits 70.000 High-Performance-Server[1]. Im momentan beginnenden Cloud-Zeitalter werden immer größere Rechenkapazitäten im Internet benötigt. Der steigende Bedarf an Server-Systemen führt zu der Entwicklung neuer Systemarchitekturen, die mehr Rechenkapazitäten auf kleinerem Raum bereitstellen können.

effizientere Technologien

Neue Konzepte ermöglichen immer effizientere Infrastrukturen. Versorgungsmodule, wie Stromversorgungsmodule oder Lüfter, werden nicht mehr für jeden Server einzeln, sondern innerhalb eines physikalischen Zusammenschlusses für mehrere Systemkomponenten bereitgestellt. Beispielsweise verfügt ein Primergy CX1000 System[2] der Firma Fujitsu Technology Solutions GmbH (FTS) über bis zu 38 Server und zwei Hochleistungslüfter, die - im Vergleich mit Produkten bei denen jeder Server einen eigenen Lüfter besitzt - effizienter kühlen. Effizient wird hier als Synonym für geringere Hardware- und Energiekosten genutzt.

Flexibilität durch Modularität

Die effizienten Systeme müssen gleichzeitig flexibler werden und kundenspezifische Systemkonfigurationen sowie laufzeitspezifische Systemanpassungen ermöglichen. Flexibilität wird durch modulare Systeme erreicht, bei denen die verschiedenen Aufgaben eines Server-Systems in unterschiedlichen Modulen gekapselt werden. Wird viel Rechenkapazität benötigt, werden mehr Servermodule bereitgestellt, wird viel Speicherkapazität benötigt, werden mehr Festplattenmodule bereitgestellt. Sind wenige Server- oder Speichermodule im Einsatz, können Stromversorgungsmodule weggelassen werden. Die Flexibilität bringt eine große Vielfalt an möglichen Ausprägungen von Server-Systemen hervor.

1 http://www.1und1.de/UnternehmenRechenzentren, letzter Aufruf: 2015-01

2 http://sp.ts.fujitsu.com/dmsp/Publications/public/ds-py-cx1000-s1-de.pdf, Onlinedokument, letzter Aufruf: 2015-01

Definition von Domänenwissen

Bei der Vielfalt an verschiedenen Ausprägungen des Systems mit ihren unterschiedlichen Einsatzgebieten und Konfigurationseigenschaften ist es wichtig wohl definierte Begriffe und Namen zu benutzen. Bei der Entwicklung eines Server-Systems ist ein Gesamtverständnis für das zu erstellende System als Grundlage notwendig. Anforderungen an das System müssen definiert werden können. Die Anforderungsdefinition ist jedoch schwierig, wenn passende Begriffsmodelle fehlen. Missverständnisse und Fehlinterpretationen werden durch ungenaue Beschreibung begünstigt. Das führt zur ersten Problemdefinition der Server-System-Entwicklung.

Herausforderung bei der Server-System-Entwicklung 1
Fehlende Festlegung des Domänenwissens in der Server-System-Entwicklung

Komplexe Verwaltungsfunktionalität

Die große Anzahl an effizienten und flexiblen Server-Systemen führt dazu, dass mehr Funktionalität und Logik bei der Verwaltung der Server-Systeme entsteht, die in der Verwaltungssoftware abgebildet werden muss. Die Komplexität der Verwaltungssoftware steigt an. Beispielsweise ist bisher die Lüftersteuerung als hardwarenahe Software in der Firmware eines eigenen Mikrocontrollers im Server bereitgestellt worden. Beim oben genannten CX-System hingegen müssen die Algorithmen zur Steuerung der Kühlung, die die Gesamtheit aller Server sowie konfigurierbare Systemeigenschaften berücksichtigt, bereits in einem Verwaltungsmodul mit der Rechenkapazität eines PCs untergebracht werden.

Reduzierung des Verwaltungsaufwands

Die Betriebskosten der Server-Systeme enthalten neben den Energiekosten auch Verwaltungskosten. Verwaltungskosten treten bei der Inbetriebnahme durch initiale Konfiguration sowie im laufenden Betrieb durch kontinuierliche Wartung auf. Server müssen hoch-verfügbar sein. Obwohl heutzutage durch Redundanzkonzepte die Ausfälle der Server für Kunden nicht erkennbar sind, müssen die Server überwacht sowie Software-Fehler und Hardware-Defekte behoben werden. Die Verwaltung jedes einzelnen Hardware-Elements ist aufwendig und deshalb werden neue Verwaltungskonzepte benötigt. Zum einen sollen Verwaltungsinformationen mehrerer Hardware-Elemente konsolidiert werden, zum anderen wird versucht, den manuellen Verwaltungsaufwand durch selbst-adaptive Mechanismen zu reduzieren.

Heterogene Schnittstellen

Wie bereits erwähnt , setzen sich modulare Server-Systeme aus verschiedenen Modulen zusammen. Die Module müssen nicht zwingend aus einem Hause stammen und bieten daher meist unterschiedliche Schnittstellen an. Auf Basis

der Schnittstellen muss die Verwaltungssoftware entwickelt werden. Zulieferer- und Fremdherstellerthematiken prägen die Entwicklung eines Gesamtsystems. Eine Trennung der Verwaltungsdomänen in Serververwaltung, Speicherverwaltung und Netzwerk bringt weitere unterschiedliche und spezialisierte Schnittstellen hervor. Die daraus resultierende Menge an heterogenen Schnittstellen wird durch die Systemarchitektur vorgegeben. Fehlt eine wohl spezifizierte Übersicht über die Systemarchitektur, resultieren daraus Schwierigkeiten bei der Systemintegration und insbesondere bei der Implementierung der Verwaltungsfunktionalität.

Herausforderung bei der Server-System-Entwicklung 2
Darstellung der Systemarchitekturen in der Server-System-Entwicklung

Systemkonzepte

Der Verwaltungszugriff auf ein modulares Server-System wird über ein Verwaltungsmodul realisiert. Das Verwaltungsmodul konsolidiert die Schnittstellen zur Verwaltung der Systemmodule. Das führt zur Vereinfachung der Systemadministration, indem das Gesamtsystem über eine einzige Verwaltungschnittstelle (Single-Point-of-Administration) verwaltet werden kann. Die Konsolidierung muss dabei die verschiedensten Systemkonzepte berücksichtigen. Das sind in der Regel die Steuerung und die Überwachung verschiedener Systemelemente. Die Schwierigkeit bei der Entwicklung der Systemkonzepte besteht durch die nicht immer offensichtlichen Abhängigkeiten der Systemkonzepte untereinander. Wird beispielsweise die Temperatur im System überwacht und es kommt zum Anstieg der Temperatur, können verschiedene Steuerungsmaßnahmen eingesetzt werden, um die Temperatur zu senken. Die Nachregelung der Lüfterdrehzahlen, die Drosselung der Prozessoren oder aber die Abschaltung von Systemmodulen durch Sicherheitskonzepte sind einige Beispiele. Auf diese Weise entsteht durch die Abhängigkeiten der Systemkonzepte ein große Menge an möglichen Systemzuständen, die in ihrer Gesamtheit bei der Entwicklung nicht berücksichtigt werden können. Demnach müssen die Systemkonzepte ermittelt und definiert werden. Auf diese Weise können Anforderungen und Abhängigkeiten für unterschiedliche Systemmodule innerhalb des komplexen Server-Systems definiert werden. Die Schnittstellen der Systemkonzepte untereinander werden wiederum über die Verwaltungsschnittstellen realisiert. Die detaillierte Spezifikation der Systemkonzepte wird bei der Server-System-Entwicklung notwendig.

Herausforderung bei der Server-System-Entwicklung 3
Ermittlung von Systemkonzepten in der Server-System-Entwicklung

Anwendung Entwicklungskonzepte

Der Softwareanteil in modularen Server-Systemen wird komplexer. Im Bereich der Softwareentwicklung werden Software-Entwicklungskonzepte, wie beispielsweise modellgetriebene oder komponentenbasierte Entwicklung, eingesetzt. Das hilft die Entwicklung von komplexen Softwaresystemen beherrschbar zu machen. Bei der Server-System-Entwicklung sind Entwickler mit mechanischen und elektrotechnischen Kenntnissen vorherrschend. Es fehlt ein Grundwissen über Software-Entwicklungskonzepte und das erschwert deren Einführung.

Herausforderung bei der Server-System-Entwicklung 4
Fehlende Software-Entwicklungskonzepte in der Server-System-Entwicklung

Fehlende Modellierungssprachen

Die Darstellung der Systemkonzepte mit geeigneten Beschreibungssprachen bildet eine weitere Schwierigkeit, da momentan keine spezialisierten Beschreibungssprachen für die Server-System-Entwicklung bekannt sind. Es muss analysiert werden, inwieweit bestehende Sprachen wiederverwendet werden können oder ob neue domänenspezifische Sprachen entworfen werden müssen. Sprachen zur Beschreibung der Systemkonzepte sollen auf passenden Software-Entwicklungskonzepten basieren.

Herausforderung bei der Server-System-Entwicklung 5
Fehlende Sprachen für die Beschreibung von Server-Systemen

Marktabhängige Anforderungen

Dem Projektpartner liegen von den Gegebenheiten des Marktes abhängige Anforderungen vor (siehe Tabelle 3.1).

Tabelle 3.1: *Marktabhängige Anforderungen*

Gegebenheiten	Anforderungen
Zum einen besteht ein großer Bedarf an Servern, zum anderen an flexiblen und modularen Server-Systemen	Reduzierung der Kosten durch effizientere System-Architekturen
Es herrscht eine komplexe Verwaltungsfunktionalität vor	Reduzierung des Verwaltungsaufwands durch Informationskonsolidierung und Einsatz adaptiver Mechanismen

Systeme bieten heterogene Schnittstellen zur Verwaltung der Systemkomponenten an	Vereinfachung der Server-System-Verwaltung, beispielsweise durch Single-Point-of-Administration

Für die Umsetzung der Anforderungen an die Server-System-Entwicklung müssen die in diesem Abschnitt definierten Herausforderungen bewältigt werden. Dafür muss die Entwicklung der Server-Systeme besser strukturiert werden. Die implizit bestehende Entwicklungsmethode muss angepasst werden. Der Fokus liegt auf der Verbesserung der Spezifikation von Anforderungen, System-Architekturen, adaptiver Funktionalität und Schnittstellen. Modellbasierte Entwicklungsmethoden sollen anvisiert werden. Folgende Vorteile werden erhofft:

- Bessere Übersicht über das Gesamtsystem für Stakeholder, Zulieferer und Entwickler
- Reduzierung des Entwicklungsaufwands durch Wiederverwendung modellbasierter Spezifikationsinhalte
- Durch konstruktive Qualitätssicherungsmaßnahmen sollen Fehler reduziert und früher erkannt werden, um den Fehlerbehebungsaufwand in späteren Entwicklungsphasen zu reduzieren
- Reduzierung von Missverständnissen durch formalisierte Spezifikationsinhalte

Lösung: Methodenweiterentwicklung

Im Sinne dieser Arbeit ist die Lösung für die Probleme der Server-System-Entwicklung eine Methodenweiterentwicklung auf Basis der genannten Ziele. Erstens stellt die Methodenweiterentwicklung eine Lösung für die verschiedenen Herausforderungen der Server-System-Entwicklung dar, indem die Methodenweiterentwicklung das grundlegende Domänenwissen definiert. Zweitens bringt der kombinierte Methoden- und Sprachentwicklungsansatz von MetaMe entsprechende Sprachen für die Server-System-Entwicklung hervor. Der Lösungsansatz führt zur folgenden verfeinerten Aufgabe der Dissertation.

Teilaufgabe der Dissertation 5
Durchführung einer Methodenweiterentwicklung für die Server-System-Entwicklung.

Nachdem nun die Aufgabe der Methodenweiterentwicklung für die Server-System-Entwicklung definiert ist, brauchen wir Grundlagenwissen über die

Server-System-Entwicklung. Dafür werden in dem folgenden Abschnitt die Aufgaben und Arten von Server-Systemen vorgestellt.

3.2 Aufgaben und Arten von Server-Systemen

Bei der Methodenweiterentwicklung in der Server-System-Entwicklung ist es notwendig einen Überblick über die zu entwickelnden Systeme zu erhalten. Insbesondere für die Bewältigung der Herausforderungen bei der Server-System-Entwicklung 1 und 2 ist es wichtig, grundlegende Begriffe und Systemelemente zu kennen, die in diesem Abschnitt vorgestellt werden.

Die Abbildung 3.1 gibt eine Übersicht über die für den Server relevanten Begriffe.

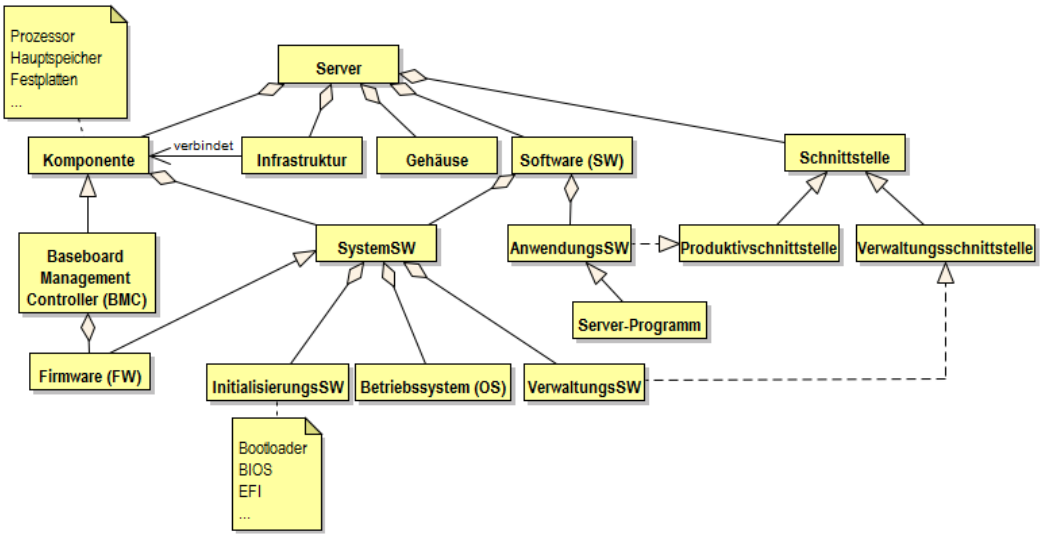


Abbildung 3.1: *Begriffe: Server*

Server

Als Erstes wird der Begriff Server definiert. In dieser Arbeit wird ein Server in der Gesamtheit seines physischen Gehäuses mit seinen Komponenten und seiner Software, bestehend aus Systemsoftware und Anwendungssoftware, angesehen.

Systemsoftware

Als Systemsoftware werden Initialisierungssoftware, Betriebssystem und Verwaltungssoftware bezeichnet. Initialisierungssoftware wird ausgeführt zum Initialisieren der Hardware und zum Start eines Betriebssystems. Beispiele dafür sind

Bootloader, das Basic Input Output System (BIOS) oder das Extensible Firmware Interface (EFI). Das Betriebssystem stellt grundlegende Operationen der Hardware bereit und verwaltet deren Ressourcen. Der Begriff Verwaltungssoftware beschreibt in dieser Arbeit die Software, die die Verwaltungsfunktionalität zur Administration des Servers bereitstellt.

Anwendungssoftware

Die Hauptaufgabe eines Servers besteht in der autonomen Ausführung der Anwendungssoftware, mit möglichst wenig manuellen administrativen Eingriffen. Anwendungssoftware ist Software zur Lösung anwenderspezifischer Aufgabenstellungen. Als Beispiele dienen hier ERP-Systeme, Datenbankanwendung und weitere.

Server-Programm

Unter dem Begriff Server wird auch eine Server-Software im Sinne der Client-Server-Architektur verstanden. Im Folgenden wird dafür explizit der Begriff Server-Programm genutzt. Ein Server-Programm ist eine Anwendungssoftware. Als Beispiel ist die Webserver-Software zu nennen.

Schnittstellen

Ein Server ist mit Schnittstellen ausgestattet. Zum einen mit Produktiv- und zum anderen mit Verwaltungsschnittstellen. Produktivschnittstellen werden ausschließlich von der Anwendungssoftware bereitgestellt und genutzt. Die Verwaltungsschnittstellen sind die Schnittstellen der Verwaltungssoftware. Mit ihnen wird die Möglichkeit der einfachen Konfiguration und Wartung des Servers zur Verfügung gestellt. Auf die Verwaltungsfunktionalität wird in Abschnitt 3.3 genauer eingegangen.

Komponenten

Ein Server besteht aus weiteren Komponenten wie Prozessor, Hauptspeicher und Festplatten.

Verwaltung des Servers

Für die Verwaltung eines Servers wird in der Regel ein zusätzlicher Mikrochip (Baseboard Management Controller (BMC)) als eigenständige Komponente im Server bereitgestellt. Der BMC verfügt über eine Firmware. Der Begriff Firmware ist der Name der Systemsoftware für ein eingebettetes System. Die Firmware stellt ebenfalls Verwaltungssoftware bereit und kann dadurch Verwaltungsfunktionalität eines Servers, unabhängig von einem gestarteten Betriebssystem im Server, zur Verfügung stellen. Eine interne Infrastruktur ist notwendig, die für die Kommunikation der Komponenten untereinander notwendig ist.

Die Begriffsdefinition eines Server-Systems wird in Abbildung 3.2 dargestellt. Ein Server-System besteht aus den verschiedenen System-Elementen. Elemente

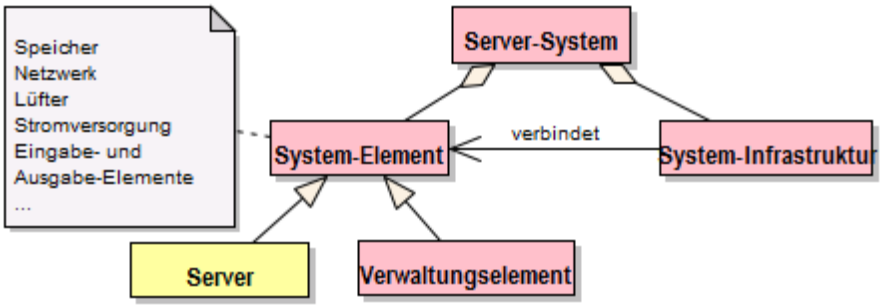


Abbildung 3.2: *Begriffe: Server-System*

Server-System

eines Server-Systems sind Server, Speicherelemente, Netzwerkelemente, Versorgungselemente sowie Eingabe- und Ausgabeelemente. Damit der Begriff Server-System gültig ist, muss mindestens ein Server und ein Versorgungselement vorhanden sein. Die System-Elemente des Server-Systems sind, wie bei den Komponenten eines einzelnen Servers, über eine System-Infrastruktur verbunden.

Arten von Server-Systemen

In den folgenden Abschnitten werden die verschiedenen Arten von Server-Systemen vorgestellt. Sie reichen von einfachen Standalone-Servern über Rack-Server-Systeme und modularen Server-Systemen bis hin zu Rechenzentren, die eine Menge von Server-Systemen zu einem Gesamtsystem vereinen. FTS geht insbesondere von der Weiterentwicklung der modularen Systeme aus, jedoch wird die Koexistenz von Rack-Server-Systemen aufgrund der großen Anzahl vorhandener Systeme erwartet. Diese Einschätzung wird auch von [Gan10] geteilt. Demnach ist es sinnvoll alle Arten von Server-Systemen zu betrachten.

3.2.1 Standalone-Server

Beschreibung

Ein einzelner Standalone-Server ist eine einfache Variante eines Server-Systems. Er besteht aus einem einzelnen Server und enthält eine Rechnerarchitektur wie sie aus einem PC bekannt ist. Im Vergleich zu einem PC sind moderne, industrielle Server-Systeme jedoch mit Verwaltungsschnittstellen ausgestattet und bieten meist hochperformante Komponenten. Sie können in den Bereichen Speicher und Rechenleistung statisch, passend zu ihrem zukünftigen Anwendungsgebiet, konfiguriert werden. Redundante Hardwarekomponenten gehören zu den Standar-

deigenschaften. Verwaltungskomponenten für die Hard- und Software sind integriert. Der Einsatz für spezielle Aufgaben ist die primäre Benutzungsbestimmung. Ein Standalone-Server bietet im Allgemeinen wenig Erweiterungsmechanismen. Dennoch ist es in einzelnen Szenarien möglich mehrere Standalone-Server zusammenzuschließen. Hierbei wird ein Verwaltungselement in Form von Hard- oder Software benötigt, damit die Gesamtheit der zusammengeschlossenen (Einzel-)Systeme wieder als ein (Gesamt-)System angesehen werden kann. Eine Darstellung der Begriffe wird in Abbildung 3.3 visualisiert.

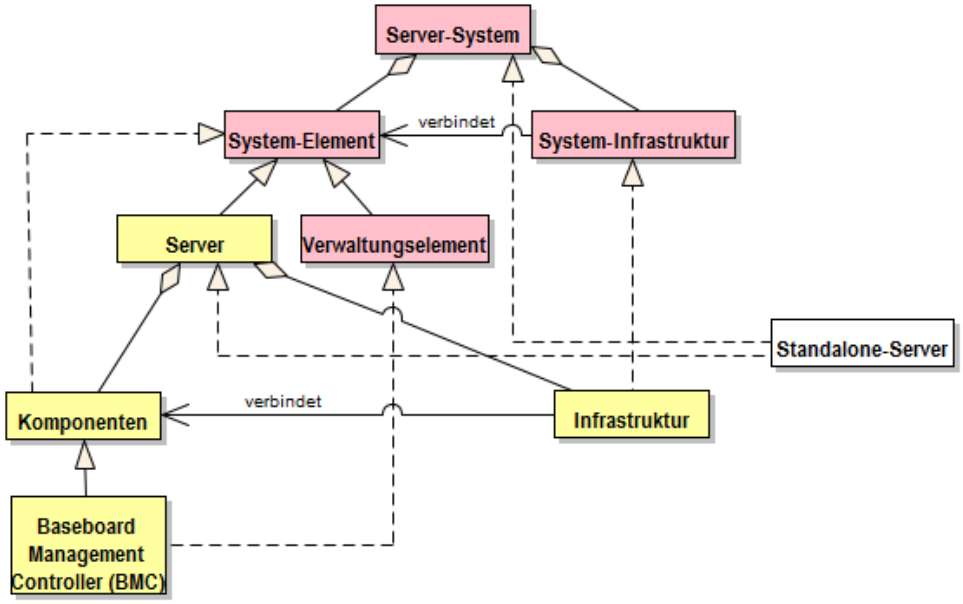


Abbildung 3.3: *Begriffe: Standalone-Server*

Abbildung auf Server-System

Ein Standalone-Server repräsentiert ein Server-System und übernimmt die Eigenschaften eines Servers. Dadurch können Komponenten als System-Elemente, die Infrastruktur als System-Infrastruktur und der BMC als Verwaltungselement angesehen werden.

3.2.2 Rack-Server-System

Beschreibung

Ein Rack-Server-System besteht aus dem Zusammenschluss einzelner Rack-Server. Der Aufbau eines einzelnen Rack-Servers ist ähnlich zu dem eines Standalone-Servers. Beim Rack-Server-System wird allerdings ein genormter

Einbaurahmen, auch Rack genannt, verwendet, damit der geordnete und dadurch Platz sparende Einbau in das Rack möglich ist. Ein Rack-Server ist mit standardisierten Verwaltungsschnittstellen ausgestattet. Für die Verwaltung des gesamten Systems wird ein spezielles Verwaltungselement vorgesehen. Die Gestaltung eines Rack-Server-Systems ist flexibel. Hierbei sind wieder unterschiedliche Konfigurationen der einzelnen Rack-Server möglich. Oft sind Netzwerk und Speicher nicht mehr Systemkomponenten des einzelnen Rack-Servers, sondern werden als eigenständige Elemente im Rack-Server-System verbaut. Weitere, spezialisierte Konfigurationsmöglichkeiten werden dadurch möglich. Der mechanische und elektrische Zusammenschluss zu einem Rack-Server-System ist eine aufwendige manuelle Aufgabe. Es muss eine logische Verwaltungsinfrastruktur durch eine manuelle Verkabelung eingerichtet werden, die nur von spezialisierten Rollen (Systemadministratoren) durchgeführt werden kann. Die Möglichkeit der beliebigen Konfiguration, angepasst an die zu erfüllenden Aufgaben, ist möglich. Eine Darstellung der Begriffe wird in Abbildung 3.4 visualisiert.

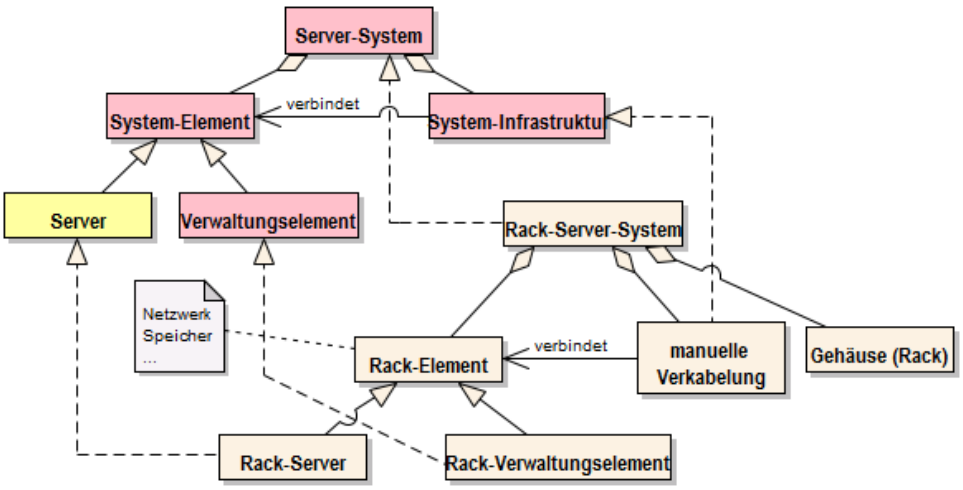


Abbildung 3.4: *Begriffe: Rack-Server*

Abbildung auf Server-System

Das Rack-Server-System ist ein Server-System, bei dem der Rack-Server die Rolle des Servers, das Rack-Verwaltungselement die Rolle des Verwaltungselements und die manuelle Verkabelung die Rolle der Infrastruktur übernimmt.

3.2.3 Modulares Server-System

Beschreibung

Der Begriff des modularen Server-Systems beschreibt ein hochflexibles, hochverfügbares und skalierbares Server-System. So weit wie möglich werden alle Teile des Gesamtsystems als einzelne Module bereit gestellt. Ein System-Chassis ist das mechanische Gehäuse, welches die Aufnahme der verschiedenen Systemmodule über Steckplätze (Slots) ermöglicht. In der Regel sind Server-, Speicher-, Netzwerk- und Verwaltungsmodule vorhanden. Das Chassis übernimmt mit seiner modularen Infrastruktur die Verkabelung der Module untereinander. Server-Module sind die einzelnen Rechnereinheiten. Netzwerk-Module sind Input/Output-Module, die den externen Netzwerkanschluss bereitstellen. Speicher-Module bieten Festplattenkapazitäten an. Verwaltungsmodule stellen eine für das Gesamtsystem aggregierte und konsolidierte Verwaltungssoftware bereit. Die Inbetriebnahme einzelner Systemmodule außerhalb eines Chassis ist nicht vorgesehen. Durch das Chassis werden vordefinierte Konfigurationsmöglichkeiten beschrieben. Je nach Bedarf können, bis zu einer hardwareseitig festgelegten Höchstmenge an Slots, mehrere unterschiedliche Module in das Chassis eingefügt werden. Beispielsweise bietet ein Primergy BX900 System von FTS gleiche Einschübe für Server- und Speicher-Module an. Auf diese Weise ist es möglich den Ausbau des Chassis entweder speicherplatz- oder rechenleistungsorientiert durchzuführen. Weiterhin ist die Stromversorgung sowie die Kühlung ebenfalls modularisiert. Diese Module werden in dieser Arbeit Versorgungsmodule genannt. Auf diese Weise können die Versorgungsmodule das gesamte System versorgen und es muss nicht jedes Systemmodul seine eigene Versorgung mit Strom und die aktive Abfuhr produzierter Abwärme sicherstellen. Das bringt Vorteile bei den Betriebskosten und dem Platzbedarf. Bei einem modularen Server-System entfällt der aufwendige, manuelle Zusammenschluss der einzelnen Systemmodule, da festgelegte Kommunikationswege mit festgelegten Schnittstellen vorgegeben sind. Ebenso ist die grundlegende Verwaltungsinfrastruktur vordefiniert. Die Inbetriebnahme und die initiale Konfiguration im Vergleich zu Rack-Server-Systemen wird erheblich vereinfacht.

Eine Darstellung der Begriffe wird in Abbildung 3.5 visualisiert.

Abbildung auf Server-System

Das modulare Server-System ist ein Server-System, bei dem das Server-Modul einen Server, das Verwaltungsmodul ein Verwaltungselement und die modulare Infrastruktur des Chassis die Infrastruktur repräsentiert. Es gibt verschiedene

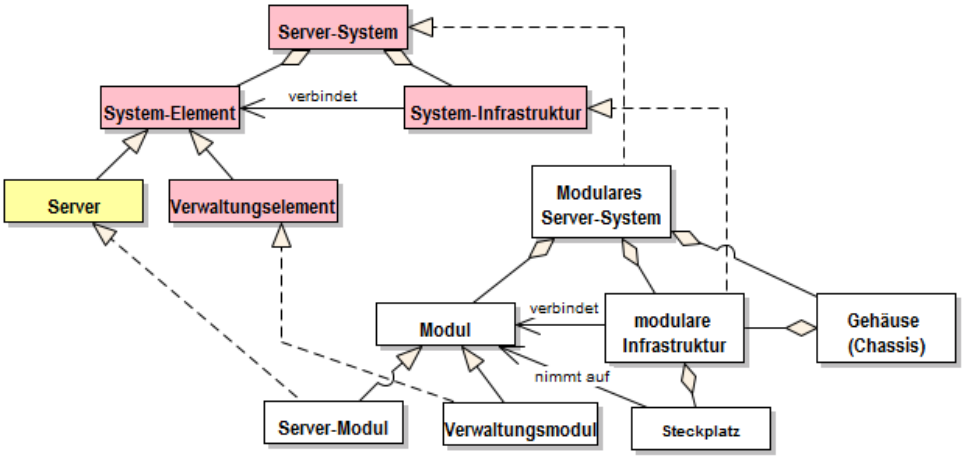


Abbildung 3.5: *Begriffe: modulares Server-System*

Ausprägungen von modularen Server-Systemen. Im Folgenden werden die beiden Varianten Blade-Server-System und Cloud-Server-System vorgestellt.

Blade-Server-System Das Blade-Server-System ist ein modulares Server-System, bei dem die Systemmodule Blades genannt werden. Die Produktpalette von Blade-Server-Systemen reicht von Systemen für kleine Betriebe, ohne IT-Abteilung, bis zu großen Betrieben, mit eigenem Rechenzentrum und IT-Abteilung. In den unterschiedlichen Betriebsumgebungen gelten verschiedene Rahmenbedingungen für die Systeme. Die kleinen Systeme sind beispielsweise für Büroumgebungen vorgesehen und das bedeutet sehr restriktive Grenzwerte im Bereich Geräuschemission. In dem Fall wird das IT-Know-how der Betreiber als gering eingeschätzt und deshalb soll eine vereinfachte Administration der Systeme möglich sein. Unterschiedliche System-Konfigurationen können nicht nur durch verschiedene Ausprägungen der Blade-Server-Systeme, sondern auch durch verschiedene Ausprägung der Blades selber erreicht werden. Hier wird auf dem Markt das Spektrum von Low-Cost bis High-End Blades abgedeckt.

Cloud-Server-System Ein modulares Server-System mit einer anderen Zielsetzung ist ein Cloud-Server-System. Während bei einem Blade-Server-System die Zielsetzung vorherrscht eine High-End-Lösung anzubieten, hat ein Cloud-Server-System den Massenmarkt im Visier, der vor allem durch Anbieter im Bereich von Cloud-Lösungen bestimmt wird. Hier steht die Eignung in virtualisierten Umge-

bungen im Vordergrund. Obwohl die Virtualisierung nicht auf der Hardwareebene umgesetzt wird, sondern auf der darüber liegenden Softwareebene, die die Hardware-as-a-Service-Funktionalität bereitstellt, gibt es spezielle Anforderungen an die Hardware. Sie muss günstig und gut skalierbar sein. Die teureren Blade-Server-Systeme eignen sich hier nicht. Die Stromversorgung und Kühlung für das Gesamtsystem bereitzustellen, bietet auch hier die Vorteile der Effizienz. Die Hardware-Konfigurationsmöglichkeiten werden eingeschränkt, dafür die Skalierbarkeit durch eine hohe Anzahl an Server-Einschüben erweitert. Einzelne (Cloud-)Server stellen nur die notwendigsten Konfigurationsmöglichkeiten bereit, dadurch wird die Infrastruktur im Cloud-System-Chassis wieder kosteneffizienter. Die Entwicklung der Server wird dadurch günstiger. Es wird nur eine Art von Server angeboten. Das reduziert den Entwicklungsaufwand und aufgrund der hohen Stückzahlen die Produktionskosten. Der momentan vorherrschende Entwicklungstrend geht in die Richtung die Cloud-Server-Systeme weiterhin zu optimieren. Das zu erreichende Ziel ist mehr Rechenleistung auf kleinerem Platz zu günstigeren Preisen anbieten zu können.

3.2.4 Rechenzentrum

Beschreibung

Die Hauptaufgabe der System-Administratoren in einem Rechenzentrum ist die Konfiguration, Verwaltung und Wartung der gesamten Hardware. Damit eine effiziente Lösung dieser Aufgaben möglich wird, werden verschiedene Server-Systeme im Rechenzentrum zu einer Verwaltungsdomäne zusammengefasst. Das geschieht in der Regel über ein autonomes Netzwerk. Ein übergeordnetes Verwaltungselement betrachtet nun nicht mehr einzelne Server, sondern gesamte Server-Systeme. Die Verwaltungsfunktionalität fokussiert hier auf Abstraktion der Server-Systeme.

Struktur

Die Abbildung 3.6 stellt die genannten Begriffe in Kombination mit der Begriffswelt der Server-Systeme dar.

Abbildung auf Server-System

Ein Rechenzentrum stellt ein Server-System dar, in dem einzelne Server-Systeme des Rechenzentrums als System-Elemente betrachtet werden. Die System-Elemente enthalten Server. Ein übergeordnetes Verwaltungselement übernimmt die Aufgabe des Verwaltungselements im Rechenzentrum. Das Netzwerk des Rechenzentrums stellt die Infrastruktur dar.

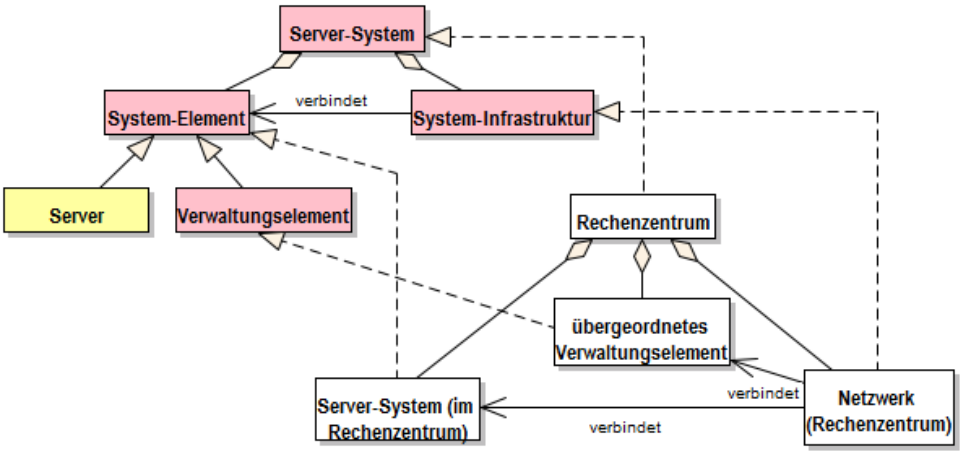


Abbildung 3.6: *Begriffe: Rechenzentrum*

Nachdem nun die Arten und Aufgaben der Server-Systeme bekannt sind, werden im folgenden Abschnitt die Grundlagen der Verwaltung von Server-Systemen beschrieben.

3.3 System Management

System Management

Für die Bewältigung der Herausforderung bei der Server-System-Entwicklung 3 *Ermittlung von Systemkonzepten in der Server-System-Entwicklung* ist es notwendig eine allgemeine Übersicht über die Verwaltungsfunktionalität zu erhalten, denn diese definiert die notwendigen Systemkonzepte. Die Gesamtheit der Verwaltungsfunktionalität wird im Bereich der Server-System-Entwicklung und Server-System-Administration *System Management* genannt.

Ursprung des System Managements

Eine frühe Festlegung von Aufgaben des System Managements ist durch das FCAPS-Modell (Fault Management, Configuration Management, Accounting Management, Performance Management, Security Management) der Internationalen Organisation für Normung (ISO) geprägt worden. Das FCAPS-Modell beschreibt die Grundfunktionen des Netzwerkmanagements. Unter Netzwerkmanagement wird die Verwaltung und Überwachung von Netzwerken mit ihrer Hardware, also auch der Server, und weiteren Elementen wie Netzwerkdienste verstanden. Das FCAPS-Modell ist ein Teil des OSI-Management-Frameworks und ist definiert im Standardisierungsdokument DIS 7498-4. Weitere allgemeine *System Management Functions* werden definiert, die allerdings in allen Aufgabengebieten angewendet werden können. Beispiele sind Log- oder Alarm-Mechanismen (vgl.: [DH08], [MS05] Seite 7ff.).

Trennung Produktiv- und Verwaltungs-umgebung

Die Produktivumgebung beschreibt Speicher und Kommunikationswege, auf denen Daten der Anwendungssoftware abgelegt oder transportiert werden, die Anwendungsseite des Server-Betriebssystems, Festplattenspeicher und deren (Produktiv-)Schnittstellen. Aus Gründen der Datenintegrität wird das System Management in einer von der Produktivumgebung logisch oder physikalisch getrennten Verwaltungsumgebung umgesetzt. Der Zugriff vom Systemadministrator auf sensible Daten des Nutzers in der Produktivumgebung wird dadurch verhindert. Hierbei stellt die physikalische Trennung der Umgebungen eine höhere Sicherheit dar als eine logische Trennung.

Architekturen

Weitere Standards definieren Architekturkonzepte für das System Management. Eine Verwaltungsarchitektur besteht aus dem Zusammenspiel verschiedener funktionaler Eigenschaften auf Basis verschiedener definierter Schnittstellen und vorgegebener Hardware-Strukturen.

Ausblick

In den folgenden Abschnitten werden die einzelnen Bausteine des System Mana-

gements beschrieben. Abschnitt 3.3.1 gibt eine Übersicht über die Verwaltungsfunktionalität. Abschnitt 3.3.2 stellt die technischen Schnittstellen zum Zugriff auf die Funktionalität vor. Abschnitt 3.3.3 erläutert Architekturen, die Funktionalität mit Daten und Schnittstellen mit Hardware zusammenbringen. Aufbauend auf den Architekturen werden Softwareschnittstellen zum Zugriff auf die Verwaltungs-Informationen und -Funktionalität definiert.

3.3.1 Verwaltungsfunktionalität

Beispiele

Die Definition, Benennung und Kategorisierung von Verwaltungsfunktionalität wird häufig von den Produktherstellern und ihren Produkten bestimmt. Es gibt nur wenige Beispiele von allgemein anerkannten Begriffen. Im Folgenden werden die Begriffe der Verwaltungsfunktionalitäten heutiger Systeme, insbesondere bei FTS, dargestellt. Eine Beschreibung der Begriffe findet sich im Anhang A.1.

- Asset Management
- Redirection
- Configuration Management
- Inventory Management / Hardware Discovery
- Cooling and Noise Management
- Deployment
- Eventing / Logging
- Fault Management
- Health and Status Management
- Time Management
- Power Management
- Redundancy Management
- User Management
- Update Management
- Localization
- Security Management
- Network Management
- Backup-Restore
- Virtualization

3.3.2 Verwaltungsschnittstellen

Zugriff

Nachdem im vorangegangen Abschnitt eine Übersicht über die mögliche Verwaltungsfunktionalität in Server-Systemen gegeben worden ist, befasst sich dieser Abschnitt mit dem technischen Zugriff auf die Funktionalität mittels Verwaltungsschnittstellen.

Benutzung

In der Verwaltungsumgebung gibt es keine Anwendungssoftware, sondern nur die Systemsoftware bestehend aus Betriebssystem und Verwaltungssoftware. Ver-

waltungsschnittstellen ermöglichen den Zugriff auf die Verwaltungsfunktionalität. Diese Funktionalität ist ausschließlich für die Benutzung durch Systemadministratoren und Verwaltungssysteme vorgesehen.

technische Eigenschaften

Technische Eigenschaften der Verwaltungsschnittstellen sind die Positionierung im System und die Restriktionen beim Zugriff. Es gibt interne Schnittstellen, die sich innerhalb des Systems befinden und externe Schnittstellen, die den Zugriff von außerhalb des Systems ermöglichen. Bei modularen Server-Systemen sind die Verwaltungsschnittstellen der Systemmodule interne Schnittstellen. Auf die internen Schnittstellen kann in der Regel nur das Verwaltungsmodul zugreifen und kapselt auf diese Weise die internen Schnittstellen. Lediglich das Verwaltungsmodul stellt externe Schnittstellen bereit. Aus Sicherheitsgründen wird der Zugriff auf die Verwaltungsschnittstellen kontrolliert. Die externen Schnittstellen werden standardmäßig mit Authorisierungs- und Authentifizierunsmaßnahmen geschützt. In Rechenzentren gewährleisten zusätzlich standardisierte Zugangskontrollen hohe Sicherheit. Bei einzelnen Server-Systemen sind teilweise mechanische Verriegelungen an den Hardware-Schnittstellen möglich.

Arten der Hardware-Schnittstellen

Folgende Arten von Hardware-Schnittstellen sind bei Server-Systemen als Grundlage für Verwaltungsschnittstellen vertreten.

Tabelle 3.3: *Arten von Schnittstellen in Server-Systemen*

Typ	Beispiele	Anwendung
Netzwerk	Ethernet	Ethernet wird in erster Linie als externe Verwaltungsschnittstelle genutzt zur Bereitstellung von Web-User-Interfaces oder höheren protokollbasierten Schnittstellen für den entfernten Verwaltungszugriff. Als interne Schnittstelle wird Ethernet für die Vernetzung von System-Modulen genutzt, die ein hohes Aufkommen an Daten haben, beispielsweise die Vernetzung verschiedener Server in einem Modularen-Server-System mit einem Verwaltungsmodul.
	Wireless LAN (WLAN) Bluetooth	Aufgrund der Schwierigkeiten der gesicherten Datenübertragung werden Funkverbindungen nicht für Verwaltungsschnittstellen genutzt.

Bussysteme	Inter-Integrated-Circuit (I^2C) System-Management-Bus (SM-Bus)	Die Bussysteme (I^2C) sowie (SM-Bus) finden Einsatz zum Zugriff auf periphere Module, ausschließlich als interne Schnittstellen innerhalb eines Server-Systems, meist auf abgeschlossener Platinen-Ebene. Aufgrund des eingeschränkten Datendurchsatzes werden hier Mikrocontroller und Sensoren mit den Verwaltungsmodulen und -komponenten verbunden.
	Serial ATA (S-ATA) Serial Attached SCSI (SAS) PCI-Express (PCIe)	Die Bussysteme S-ATA, SAS und PCIe gewährleisten einen höheren Datendurchsatz als I^2C und SM-Bus. Sie verbinden höherwertige, periphere Geräte innerhalb des Systems. S-ATA und SAS werden hauptsächlich für den Anschluss von Festplatten an den Server oder allgemeiner Storage-Module mit Server-Modulen genutzt, jedoch nicht als Verwaltungsschnittstellen. PCIe findet allerdings in speziellen Fällen Einsatz als Verwaltungsschnittstelle.
einfache Signalverbindungen	Pulsweiten-Modulation-Signal (PWM-Signal)	PWM-Signale werden zur Ansteuerung von Lüftern genutzt.
	General Purpose Input/Output GPIO	Ein GPIO ist eine konfigurierbare Signalschnittstelle, die als Eingabe oder Ausgabe konfiguriert werden kann. In Server-Systemen werden GPIOs zum Anschluss peripherer Geräte genutzt. Beispiele sind die Nutzung als Presence Signal weiterer Systemkomponenten, Ansteuerung von LEDs oder Auslesen von Sensoren.
parallele Schnittstellen	ISA, ATA, SCSI	Parallele Schnittstellen sind veraltet und werden in fast allen Fällen durch neuartige serielle Schnittstellen ersetzt. In der Regel spielen sie heutzutage für Verwaltungsschnittstellen keine Rolle.

Architektur-Konzepte

Die Hardware-Schnittstellen stellen Software-Schnittstellen zum Zugriff auf die Verwaltungsfunktionalität bereit. Die Software-Schnittstellen werden in der Systemsoftware implementiert. Es gibt zwei unterschiedliche Architekturkonzepte, In-Band- und Out-of-Band-Management.

In-Band-Management

Basiert die Verwaltungsumgebung auf derselben Infrastruktur wie die Produktivumgebung, handelt es sich um In-Band-Management. Beispielsweise stellt ein Server die Software-Schnittstellen der Verwaltungsschnittstellen über das Be-

triebssystem bereit, welches auch die Grundlage für die Anwendungssoftware darstellt. Ein Netzwerkgerät bietet administrativen Zugriff über Schnittstellen an, die auch von der Produktivumgebung genutzt werden können.

Die Tabelle 3.4 stellt die Vor- und Nachteile des In-Band-Managements dar.

Tabelle 3.4: *Vor- und Nachteile des In-Band-Managements*

Vorteile	**Nachteile**
Es bedarf keiner Definition einer separaten Systeminfrastruktur. Dies hat Kostenvorteile in der Entwicklung und im Betrieb des Server-Systems.	Mit der In-Band-Management-Architektur ist es lediglich möglich eine logische Trennung von Produktiv- und Verwaltungsumgebung zu erreichen.
Implementierte Funktionalität des Betriebssystems ist für die Systemverwaltung wiederverwendbar, beispielsweise Treiber für Hardwarekomponenten. Der Zugriff auf Konfigurationseigenschaften des Betriebssystems ist möglich. Es muss nur ein Netzwerk konfiguriert werden.	Die Verwaltungsschnittstellen sind bei dieser Architektur abhängig von einem installierten Betriebssystem. Die Ausführung des Betriebssystems ist bei Server-Systemen nicht unterbrechungsfrei. Bei einem Neustart oder Absturz des Systems ist kein entfernter Zugriff auf das System möglich. Produktiv- und Verwaltungsumgebung sind im selben Netzwerk definiert. Im Fehlerfall des Netzwerks entfällt ebenfalls der entfernte Zugriff auf die Verwaltungsfunktionalität.

Out-Of-Band-Management

Wird eine separate Infrastruktur für die Verwaltungsumgebung bereitgestellt, handelt es sich um Out-Of-Band-Management. In diesem Fall muss für ein Hardware-Interface, welches eine Verwaltungsschnittstelle darstellt, jeweils ein Controller mit Verwaltungssoftware bereit gestellt werden.

Tabelle 3.5: *Vor- und Nachteile des Out-of-Band-Managements*

Vorteile	Nachteile
Eine physikalische Trennung von Produktiv- und Verwaltungsumgebung wird erreicht. Auf Basis der separaten Infrastruktur ist keine Abhängigkeit zum Betriebssystem vorhanden. Ein unterbrechungsfreier administrativer Zugriff auf die Hardware ist gewährleistet. Beim Absturz des Betriebssystems werden Fehlerkorrekturmaßnahmen gestartet. Der Zugriff auf die Verwaltungsschnittstellen ist unabhängig vom Netzwerk der Produktivumgebung. Fehlerkorrekturmaßnahmen können während eines Ausfalls des Produktivnetzwerks ergriffen werden.	Zusätzliche Kosten für die separaten Hardwarekomponenten. Erhöhte Entwicklungskosten durch die Bereitstellung separater Systemsoftware. Ein direkter Zugriff auf Funktionalität des Betriebssystems ist nicht möglich.

Zusammenfassung

Es sind die technischen Möglichkeiten zum Zugriff auf die Verwaltungsfunktionalität vorgestellt worden. Die Brücke zwischen Hardware und Daten sowie den notwendigen Softwareschnittstellen werden beim System Management durch Verwaltungsarchitekturen gebildet. Der folgende Abschnitt gibt eine Übersicht über Verwaltungsarchitekturen und stellt die für diese Arbeit wichtigen Standards in dem Bereich vor.

3.3.3 Verwaltungsarchitekturen (CIM)

Einleitung

Das Common Information Model (CIM), das Simple Network Management Protocol (SNMP) und das Intelligent Platform Management Interface (IPMI) sind die wichtigsten Standards für das System Management. Alle drei Vertreter definieren ein Konzept und eine Architektur, die im Detail unterschiedlich sind, auf abstrakterer Betrachtungsebene jedoch viele Gemeinsamkeiten haben. Als Grundlage und hinsichtlich der Verständlichkeit der Anwendungsbeispiele dieser Arbeit beschränkt sich die Beschreibung in diesem Abschnitt auf CIM. Weitere Informa-

tionen über SNMP und IPMI sind im Anhang (siehe Abschnitt A.2.1 und A.2.2) zu finden.

Common Information Model (CIM) Heutige IT-Systeme bestehen neben Hardwareelementen aus Netzwerkinfrastrukturen, Applikationen und der Verwaltungssoftware. Während SNMP auf die Verwaltung von einzelnen Netzwerkkomponenten und IPMI auf die Verwaltung einzelner Hardwareelemente mit deren Firmware fokussiert, wird ein Standard benötigt, der allumfassend ein gesamtes IT-System verwalten kann. Für die Entwicklung eines solchen Standards ist die Distributed Management Task Force Inc. (DMTF)[3] im Jahr 1992 gegründet worden. Der wichtigste Standard der DMTF ist das Common Information Model (CIM).

Standard zur Verwaltung von IT-Systemen

CIM stellt eine gemeinsame Definition für Managementinformationen und Konfigurationseigenschaften von Systemen, Netzwerken, Anwendungen und Services bereit[4]. CIM besteht in erster Linie aus einem Datenmodell, welches System- und Informationsstrukturen von verwaltbaren Systemelementen beschreibt. Das Datenmodell wird CIM-Schema genannt und ist über ein Metamodell, das CIM-Meta-Schema, definiert. Weiterhin gibt es die CIM-Spezifikation, die die Semantik auf Basis des CIM-Meta-Schemas für das CIM-Schema und die Interaktion mit weiteren Verwaltungsarchitekturen detailliert.

CIM

Wie in der Abbildung 3.7 zu erkennen ist, ist das grundlegende Meta-Element das `CIM-NamedElement`. Vom `CIM-NamedElement` erben alle strukturbeschreibenden Elemente bis auf die Modellelementinstanzen, Werte, Datentypen und Metainformationen (`CIM-Qualifier`). In dieser Arbeit werden Instanzen von `CIM-NamedElement` auch CIM-Elemente genannt. Die Existenz von Klassen (`CIM-Class`) mit Attributen (`CIM-Property`) und Methoden (`CIM-Method`) weisen eine Ähnlichkeit zur Unified Modeling Language (UML)[Obj11c] auf. Assoziationen (`CIM-Association`) sind als Unterklasse von `CIM-Class` definiert. Ihre Assoziationsenden werden mit Referenzen (`CIM-Reference`) als Unterklasse von `CIM-Property` beschrieben.

CIM-Meta-Schema

Die Bedingung, dass eine `CIM-Association`Instanz mindestens zwei `CIM-Reference`Instanzen besitzt, die jeweils als ein Datentyp eine Referenz auf eine `CIM-Class` definieren, ist dem CIM-Meta-Schema nicht zu entnehmen. Die-

Integritätsbedinungen

3 http://www.dmtf.org, letzter Aufruf: 2015-01
4 http://dmtf.org/standards/cim, letzter Aufruf: 2015-01

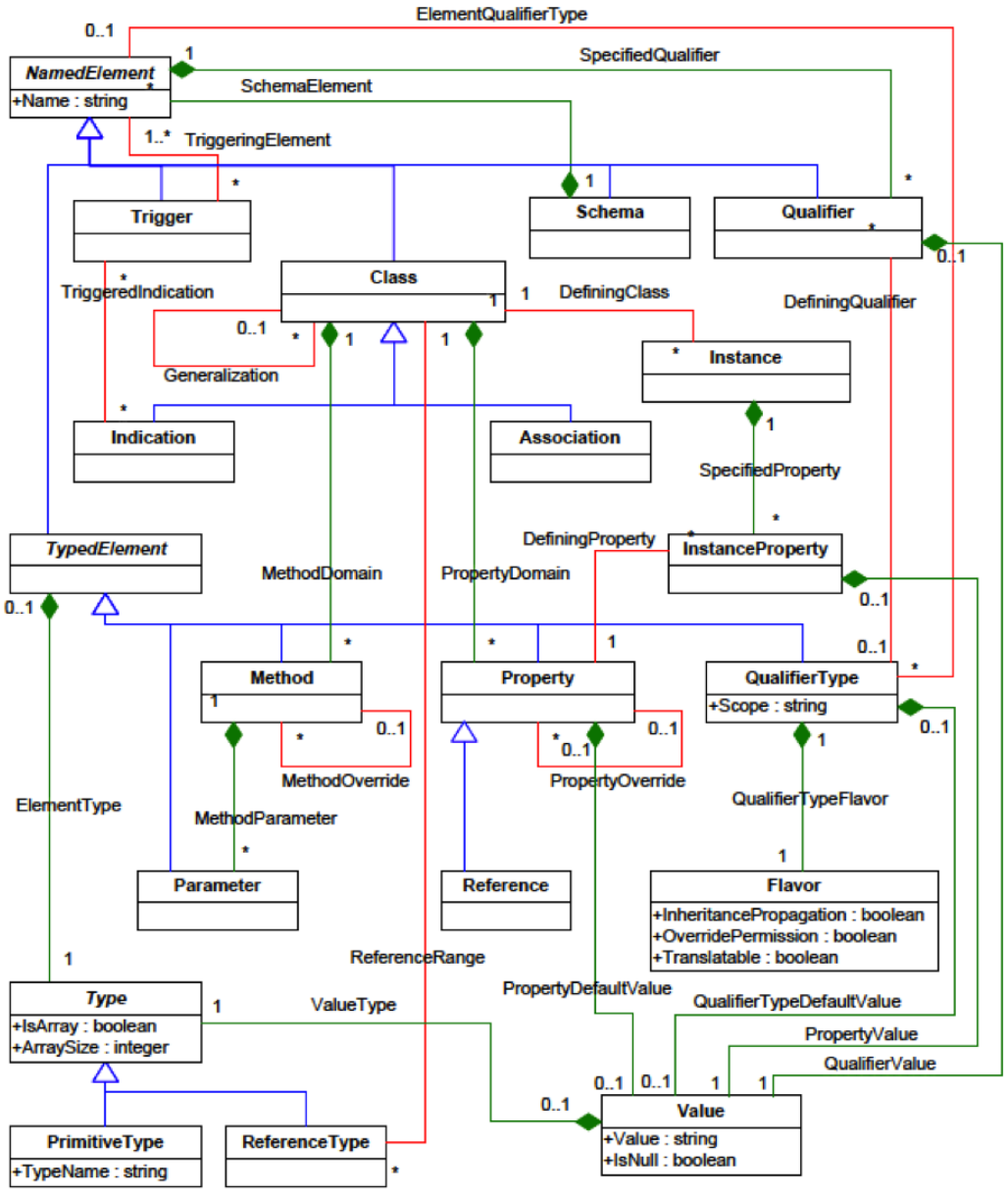


Abbildung 3.7: *CIM-Metaschema [Dis10a]*

se Informationen sind in der CIM-Spezifikation über Bedingungen in Form der Object Constraint Language (OCL) [Obj11a] beschrieben.

```
1 context Association
2 inv: self.PropertyDomain[OwningClass].OwnedProperty->
3 select( p / p.oclIsTypeOf(Reference))->size() >= 2
```

Listing 3.1: *Beispiel OCL Bedingung in der CIM-Spezifikation*

Im CIM-Schema, also einer Instanz des CIM-Meta-Schemas, übernimmt eine Instanz des Metamodellelements `CIM-Schema` die Aufgabe des Wurzelelements und aggregiert alle modellierten Elemente des CIM-Schemas.

CIM-Schema

Die Abbildung 3.8 beschreibt die Zuordnung des CIM-Meta-Schemas und des CIM-Schemas zu den von der OMG definierten Metamodellebenen der Meta Object Facility (MOF) [Obj13].

Abbildung auf die MOF Meta-Modell-Ebenen

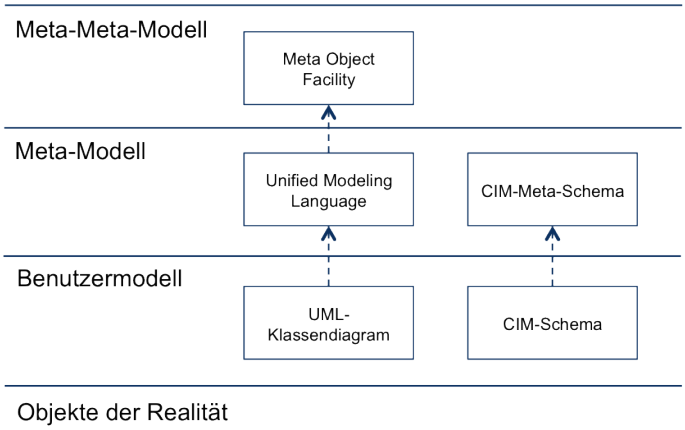


Abbildung 3.8: *Metaebenen CIM und UML*

Das CIM-Schema ist aufgrund der ähnlichen Elemente vergleichbar mit UML-Klassendiagrammen und folgt dem Konzept der Objektorientierung. Beide Sprachen benutzen den Begriff Klasse, welcher im Meta-Modell definiert wird. Instanzen der Klasse werden auf der Benutzermodellebene definiert. In beiden Sprachen können Objekte (`UML-Object` und `CIM-Instance`) der Klassen ebenfalls auf Benutzermodellebene dargestellt werden. Sich entsprechende Modelle befinden sich auf derselben Metamodellebene (siehe Abbildung 3.8).

Vergleich CIM und UML

CIM ist integriert in dem Web based Enterprise Management Standard (WBEM)[5] der DMTF. WBEM beschreibt eine Sammlung von Standards für den entfernten

WBEM

5 http://www.dmtf.org/standards/wbem, letzter Aufruf: 2015-01

Zugriff auf die über CIM beschriebene Verwaltungsfunktionalität. Darunter fallen unter anderem eine Abbildung von CIM auf ein XML-Format (xmlCIM) [Dis09b], Zugriffsprotokolle (CIM-XML u.a.) [Dis09a] sowie die Abfragesprache CIM Query Language (CQL) [Dis07a].

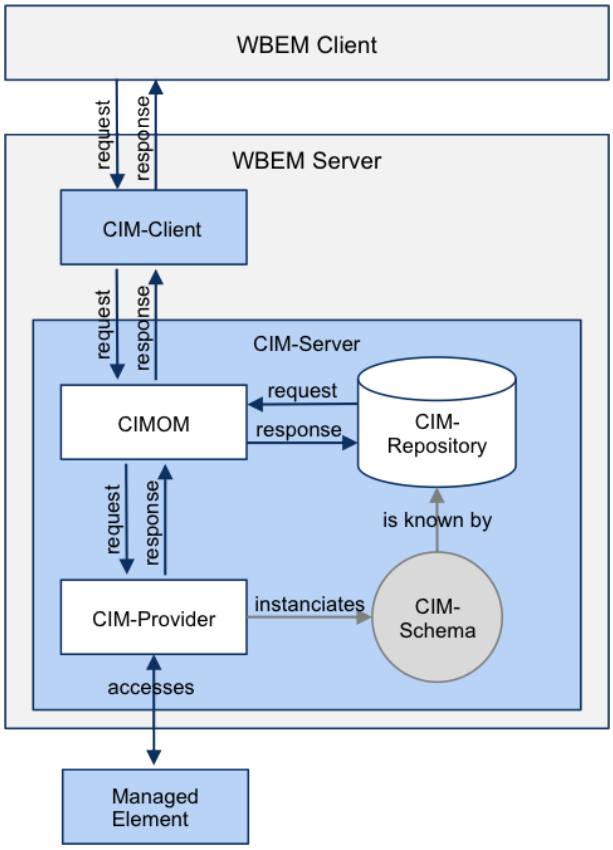


Abbildung 3.9: *Standard WBEM- / CIM-Architektur*

WBEM-Architektur

Die WBEM-Spezifikationen beschreiben implizit eine Standardarchitektur, wie sie in Abbildung 3.9 dargestellt ist. Der *WBEM-Client* repräsentiert die Management-Software, die der Systemadministrator nutzt, um auf den *WBEM-Server* zuzugreifen. Der WBEM-Server implementiert die CIM-Funktionalität. Der WBEM-Server wird in einem Server-System integriert, als eigenständiges Modul oder Komponente, oder auch extern vom Server-System umgesetzt. Der WBEM-Server muss über Zugriff auf die zu verwaltenden Elemente, den *Managed Element*s, verfügen. Managed Elements können beliebige Elemente sein, die über das CIM-Schema beschrieben werden. Der WBEM-Server besteht aus *CIM-Client*s. CIM-Clients implementieren die einzelnen Zugriffprotokolle. Sie nehmen Anfragen der Management Clients entgegen und übersetzen die Anfrage in CIM-

XML oder proprietäre Protokolle, die der Common Information Model Object Manager (*CIMOM*) versteht. Der CIMOM ist die zentrale Einheit zur Bearbeitung der Anfragen. Der CIMOM hat Zugriff auf das *CIM-Repository*. Im CIM-Repository sind sämtliche CIM-Schema-Elemente registriert, für die eine Implementierung im zu verwaltenden System vorhanden ist. Auf Basis der vom CIM-Client eingehenden Anfragen schaut der CIMOM nach, ob das angefragte CIM-Element im CIM-Repository bekannt ist. Wenn das CIM-Element im CIM-Repository registriert ist, wird eine Referenz auf den entsprechenden *CIM-Provider* zurückgeliefert. Der CIM-Provider ist die Implementierung des registrierten CIM-Elements (vergleiche mit [Hob04][6]). Der CIMOM übersetzt die Anfrage und ruft den CIM-Provider auf. Dieser erstellt die Modellinstanzen der CIM-Elemente konform zu den im CIM-Schema definierten strukturellen Eigenschaften. Dazu greift der CIM-Provider auf systemspezifische Schnittstellen der Managed Elements zu. Die vom CIM-Provider erstellte Modellinstanz wird als Antwort an den CIMOM geliefert. Der CIMOM übersetzt die Antwort für den CIM-Client, der danach die Antwort über das entsprechende Protokoll an den WBEM-Client sendet, welcher die Anfrage gestellt hat.

CIM basierte Schnittstellen

Unterschiedliche standardisierte Schnittstellen sind im Umfeld des WBEM-Servers zu nennen. Die CIM-Clients stellen die unterschiedlichen Protokolle für den Zugriff auf den WBEM-Server bereit. Hier ist der Zugriff über CIM-XML, über das Smash Command Line Protocol (SM-CLP) [Dis07c] oder Web Services (for) Management (WS-MAN) [Dis10b] spezifiziert. CIM-XML ist das grundlegende Protokoll des WBEM-Standards. CIM-XML wird abhängig von der Implementierung auch zur Kommunikation zwischen CIM-Client und CIM-Server genutzt, meist aber durch durchsatzstärkere proprietäre Protokolle ersetzt. SM-CLP ist ein Protokoll, welches als CIM-Client umgesetzt wird. Für den Fall des entfernten Zugriffs muss Telnet oder das Secure Shell (SSH) genutzt werden. WS-Man basiert auf dem Simple Object Access Protocol (SOAP) und präsentiert sich deshalb als passender Partner für den Datenaustausch über das Hypertext Transport Protocol (HTTP). Der größte Teil heutiger IT-Strukturen basiert auf HTTP. Für Kommunikation zwischen CIMOM und CIM-Provider wird teilweise die standardisierte Schnittstelle Common Manageability Programming Interface (CMPI) genutzt.

6 Der Autor Chris Hobbs beschreibt den CIM-Provider außerhalb des WBEM-Servers. Ich habe mich an der Stelle bewusst für die Darstellung des CIM-Providers innerhalb des WBEM-Servers entschieden. Die Registrierung des CIM-Schemas im CIM-Repository bedingt die Existenz eines CIM-Providers. Mir sind keine Architekturen bekannt, die eine Trennung vorsehen. Ansonsten ist kann fälschlicherweise angenommen werden, ein Provider wird von einem Managed Element umgesetzt.

Das Core Modell

Das allgemeingültige CIM-Schema ist umfangreich. Es ist unterteilt in Core Modell und diverse Common Modelle. Das Core Modell umfasst unter anderen die in der folgenden Aufzählung zusammengefassten Informationen.

- Qualifier: Das Qualifier Konzept bietet die Möglichkeit sämtliche CIM-Elemente mit Metadaten zu beschreiben. Getypt ist ein Qualifier über das CIM-Meta-Schema-Element `CIM-Qualifier`. Das Core Modell definiert die möglichen Datentypen der Qualifier.
- Basisklassen: Die Basisklassen definieren unterschiedliche Oberklassen und bilden so eine Kategorisierung in verwaltbare Elemente, logische Elemente, Softwarelemente, Systemelemente und weitere. Ebenfalls werden die Basis-Assoziationen wie Abhängigkeit oder Komponente definiert. Diese Oberklassen definieren so grundsätzliche Eigenschaften sowie eine vordefinierte Semantik aller von den Basisklassen abgeleiteten Klassen.
- Physikalische Elemente: Beschreibt grundsätzliche Eigenschaften physikalischer Elemente wie beispielsweise Herstellerinformationen.
- Software-Identitäten: Beschreibt grundsätzliche Eigenschaften von Softwareelementen wie beispielsweise Versions- und Releasenummern.
- Devices: Beschreibt die logischen Funktionalitäten von Hardwareelementen.
- Collections: Bietet die Basisklassen für die Zusammenfassung und Aggregation von CIM-Elementen.
- Redundanz: Definiert die Basisklassen für die Beschreibung von redundanten Systemelementen und deren Beziehungen.
- Statistics: Basisklasse für die Beschreibung von statistischen Daten und Metriken.
- Capabilities: Definiert die strukturellen Informationen zur Beschreibung beliebiger CIM-Elemente, die explizit, also nicht über Meta-Informationen, präsentiert werden.
- Power Management: Beschreibt die grundlegenden Elemente und Eigenschaften zur Definition und Änderung von Betriebszuständen der CIM-Elemente.

Die Common Modelle

Weitere vordefinierte und detaillierende Eigenschaften werden durch so genannte Common Modelle, wie beispielsweise Applications Model, Database Model, Event Model und weitere, beschrieben.

Profile / Verwaltungsdomänen

Eine Strukturierung des CIM-Schemas wird durch Profile erreicht. Während das CIM-Schema die strukturellen Informationen der CIM-Klassen mit ihren At-

tributen und Methodensignaturen beschreibt und die Qualifier-Informationen den strukturellen Elementen Metainformation hinzufügen, werden den Elementen abhängig vom Auftreten in einer Verwaltungsdomäne zusätzliche semantische Eigenschaften zugesprochen. Diese werden in CIM-Profildokumenten beschrieben. Beispiele sind das Base Server Profile zur Beschreibung eines einzelnen Servers, das Modular Systems Profile für die Beschreibung eines Modularen-Server-Systems. Die folgende Auflistung beschreibt eine Auswahl der Verwaltungsdomänen, für die CIM-Profildokumente vorhanden sind. Die Verwaltungsdomänen sind den Begriffen der Verwaltungsfunktionalität aus dem Abschnitt 3.3.1 zugeordnet. Dadurch wird die Vielfalt der über CIM verwaltbaren Aspekte dargestellt.

- Inventory Management
  - Base Server Profile
  - Modular Systems Profile
  - Service Processor Profile
  - Fan Profile
  - Sensors Profile
  - CPU Profile
  - System Memory Profile
  - Power Supply Profile
  - Battery Profile
- Network Management
  - DHCP Client Profile
  - Command Line Protocol Profile
  - Ethernet Port Profile
  - Telnet Service Profile
  - SSH Service Profile
  - IP Interface Profile
  - DNS Client Profile
- Logging
  - Record Log Profile
- Update Management
  - Software Identity Profile
  - Software Update Profile
- User Management, Security Management
  - Simple Identity Management Profile
  - Role Based Authorization
- Power Management
  - Boot Control Profile
  - Power State Management Profile
  - Power Utilization Management Profile
- Redirection
  - Text Console Redirection
  - USB Redirection Profile
  - KVM Redirection Profile
- Eventing / Logging
  - Record Log Profile
  - Platform Watchdog Profile
  - Indications Profile
- Virtualization
  - Virtual System Profile
  - Platform Watchdog Profile
  - Virtual Ethernet Switch Profile

Ausweitung der Verwaltungs-domänen

Weitere Organisationen zur Standardisierung von Verwaltungsinformationen nutzen CIM und weiten die Definitionen, und dementsprechend die Verwaltungsdomänen, aus. Beispielsweise gibt es die Storage Network Industry Association (SNIA)[7]. Sie definieren die Storage Management Initiative Specification (SMI-S)[8]. Das ist ein Standard für die Hersteller übergreifende Verwaltung von Storage Area Networks (SAN). Die Definition von Eigenschaften der SANs wird über diverse CIM-Profildokumente gewährleistet.

OEM-Erweiterungen des CIM-Schemas

Erweiterungen des CIM-Schemas sind erwünscht. Der CIM-Standard beschreibt den Begriff des Erweiterungsschemas. Damit ist es einem OEM möglich die strukturellen Informationen des CIM-Schemas nach eigenen Bedarf zu erweitern. Am folgenden Beispiel (siehe Abbildung 3.10) soll die Notwendigkeit und Möglichkeit der Erweiterung beschrieben werden.

Beispiel für eine OEM-Erweiterung

Wir betrachten die Verwaltungsfunktionalität *Asset Management* (siehe Abschnitt 3.3.1). Hier wird eine OEM-spezifische Anforderung formuliert. Jedem Server-System soll eine eindeutige Inventarnummer zugeordnet werden. Die Inventarnummer soll an die Hardware gebunden sein. Sie soll über CIM-basierte Schnittstellen gesetzt und ausgelesen werden. Im Sinne einer Machbarkeitsstudie wird das CIM-Schema analysiert und die Definition eines Attributs gesucht, welches die Inventarnummer beschreibt. In der Abbildung 3.10 ist die Klasse CIM_Chassis mit ihren Attributen (Property) sowie ihren Oberklassen mit deren Attributen und Methoden (siehe CIM_PhysicalPackage) dargestellt. CIM_Chassis beschreibt die Hardwarerepräsentation eines Server-Systems. Im Sinne der CIM-Spezifikation und des CIM-Profilspezifikationsdokuments Physical Asset Profile, welches die passende Verwaltungsdomäne darstellt, eignet sich kein eigenes oder vererbtes Attribut für die Abbildung der Inventarnummer. Weiterhin lässt sich im CIM-Schema auch keine zum CIM_Chassis zugehörige Klasse finden, die eine Inventarnummer der Hardware beschreibt. In diesem Beispiel erweitern wir die Klasse CIM_Chassis um die Klasse OEM_Chassis. In der Klasse OEM_Chassis definieren wir nun ein passendes Attribut mit dem Namen AssetTag. Die Inventarnummer ist nun im Erweiterungsschema beschrieben.

Erweiterungs-möglichkeiten

Es darf generell keine Anpassung an einer Klasse des CIM-Schemas gemacht werden. Durch die Möglichkeit der Spezialisierung einer Klasse über die Generalisierungsbeziehung ist es möglich der abgeleiteten Klasse Attribute und Me-

7 http://www.snia.org/, Webseite, letzter Aufruf: 2015-01

8 http://www.snia.org/tech_activities/standards/curr_standards/smi, Webseite, letzter Aufruf: 2015-01

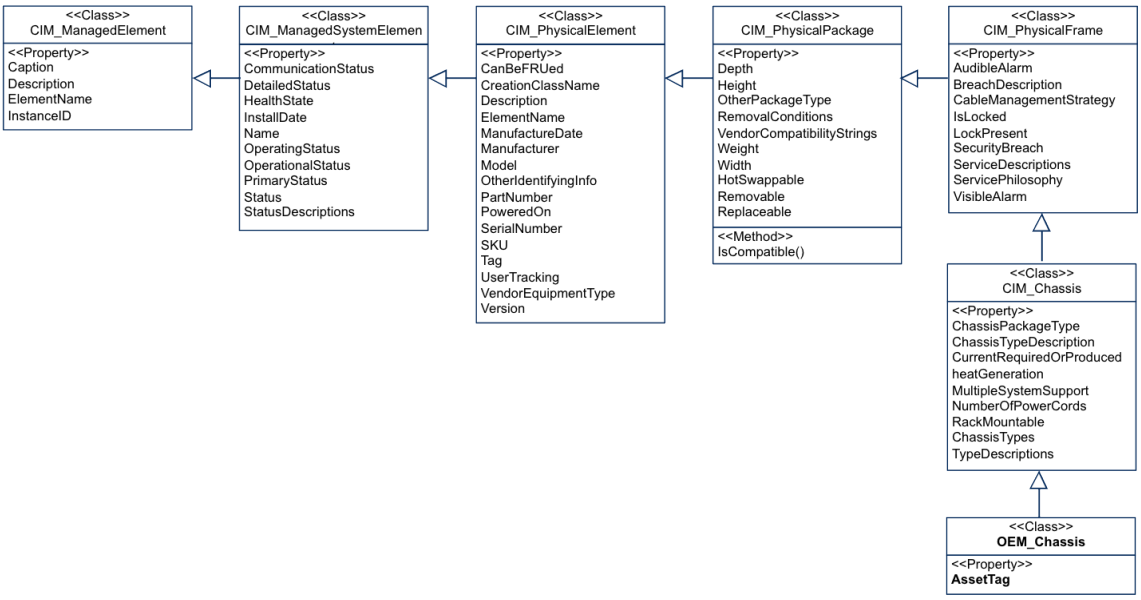


Abbildung 3.10: *Beispiel einer OEM-Erweiterung des CIM-Schemas v 2.24.1*

thoden hinzuzufügen. Die abgeleitete Klasse stellt eine Spezialisierung der Oberklasse dar. Wie im obigen Beispiel basiert die Spezialisierung meist auf individuellen Anforderungen eines OEMs, um weitere strukturelle Eigenschaften eines Elements einer Verwaltungsdomäne zu definieren. Es ist ebenfalls möglich neue Verwaltungsdomänen zu definieren, indem Klassen und Assoziationen mit der Semantik der neuen Verwaltungsdomäne erstellt werden. Während neue Klassen, Attribute und Methoden erstellt werden können, ist weiterhin lediglich eine Einschränkung der Wertebereiche vorhandener Attribute erlaubt. Eine Erweiterung verletzt den Standard.

MOF

Die Erweiterungen im Erweiterungsschema müssen im CIM-Repository abgelegt werden. Dafür wird das Model Object Format (MOF)[9] genutzt. Das MOF ist über eine Grammatik in Augmented-Backus-Naur-Form definiert. Heute verfügbare CIMOMs können die strukturellen Informationen des CIM-Schemas und des Erweiterungsschemas im MOF verarbeiten. Die Standard-CIM-Klassen werden von der DMTF in MOF-Dateien bereitgestellt. Die Klassen des Extension-Schemas müssen vom OEM ins MOF überführt werden.

formale Strukturbeschreibung

Die strukturellen Eigenschaften der CIM-basierten Verwaltungsfunktionalität können formal, getypt über das CIM-Meta-Schema und zusätzlichen Integritätsbe-

9 Das Model Object Format (MOF) sollte nicht mit dem Meta Object Facility (ebenfalls MOF) verwechselt werden.

dinungen durch OCL-Constraints, beschrieben werden. CIM bietet jedoch keinen Formalismus zur Verhaltensbeschreibung.

intrinsische Funktionalität

CIM unterscheidet zwischen intrinsischer und extrinsischer Funktionalität. Intrinsische Funktionalität umfasst alle Standardfunktionen, die auf dem CIM-Schema beziehungsweise auf Instanzen von `CIM-Class` und `CIM-Association` ausgeführt werden können. Das umfasst:

- die Aufzählung aller Instanzen einer CIM-Klasse
- das Abfragen einzelner Instanzen einer CIM-Klasse
- das Erstellen von Instanzen einer CIM-Klasse
- das Löschen von Instanzen einer CIM-Klasse
- die Modifikation von Instanzen einer CIM-Klasse

extrinsische Funktionalität

Die extrinsische Funktionalität umfasst alle definierten Methoden der CIM-Klassen und CIM-Assoziationen (beispielsweise die Methode *IsCompatible()* der Klasse CIM_PhysicalPackage aus Abbildung 3.10).

nicht formale Verhaltensbeschreibung

Die Verhaltensbeschreibung der intrinsischen Funktionalität wird in der CIM-Spezifikation in textueller Form beschrieben. Die CIM-Profilspezifikationsdokumente detaillieren das intrinsische und beschreiben das extrinsische Verhalten ebenfalls in textueller Form. Es werden Regeln und Bedingungen für die Instanziierung einzelner Klassen und Assoziationen, sowie für die Ausführung der intrinsischen und extrinsischen Methoden vorgegeben.

OEM-Spezifikation

Die Implementierung der Funktionalität und der konkreten Anwendung für ein bestimmtes System muss vom OEM spezifiziert werden. An dieser Stelle gibt die DMTF mit der Spezifikation [Dis11] und der Dokumentvorlage [Dis07b] eine Struktur der Spezifikation vor, jedoch keinerlei Anweisung zur formalen Verhaltensbeschreibung.

Vor- und Nachteile des OO-Paradigmas

Der objektorientierte Ansatz von CIM bietet die Möglichkeit der strukturierten und formalen Beschreibung der strukturellen Verwaltungsinformationen von IT-Systemen. Darauf aufbauend kann eine elegante Form der Erweiterungen des CIM-Schemas mit dem Vererbungsparadigma etabliert werden. Die Verwaltungsfunktionalität beim Out-of-Band-Management wird jedoch meist von Systemkomponenten und -modulen umgesetzt, die den typischen Restriktionen eingebetteter Systeme, wenig Speicher und Rechenkapazitäten, unterliegen. Somit muss ei-

ne geeignete Umsetzung der Objektorientierung in hardwarenahen Programmiersprachen nachgebaut werden. Des Weiteren ist das Paradigma der Objektorientierung bei den Entwicklern von Out-of-Band-Verwaltungssoftware wenig bekannt. In dem Bereich ist meistens wenig bis kein Vorwissen durch die Anwendung von höheren *objektorientierten* Programmiersprachen vorhanden.

fehlende Konzepte, Methoden und Werkzeuge

Bis auf die von der DMTF bereitgestellten Spezifikationsdokumente sind wenig zusätzliche Informationen verfügbar. Eine dem Standard konforme Implementierung CIM-basierter Schnittstellen im Bereich Out-of-Band-Management ist momentan marktrelevant und dadurch werden gewonnene Erkenntnisse der Spezifikations-, Implementierungs- und Testaufgaben als gut zu sicherndes Geschäftseigentum angesehen. Die In-Band-Management-Implementierung Windows Remote Management (WinRM) stellt die Microsoft-Implementierung von CIM und WS-Man dar. Die Basisfunktionalität wird von jedem aktuellen Windows Betriebssystem ausgeliefert, ist also etabliert und weit verbreitet, aber auch hier fehlen veröffentlichte Entwicklungsmethoden und verfügbare Entwicklungswerkzeuge. Einige Open-Source-Werkzeuge sind aus dem SBLIM-Projekt[10] entstanden. Wobei der Small Footprint CIMOM-Broker (SFCB) und der WBEM-Client wbemcli, ein Kommandozeilenprogrogramm zur entfernten Abfrage von CIM-Informationen über dem CIM-XML-Protokoll, nennenswert sind. In der Forschung gibt es nur zu vernachlässigende Aktivitäten die CIM betreffen. Jedoch zeigen die Entwicklungsaktivitäten Bedarf nach Entwicklungskonzepten und -werkzeugen. Bei der Spezifikation und beim Testen sind hohe Automatisierungspotenziale erkennbar. Bei der Entwicklung von CIM-Extension-Schemas können Modellierungswerkzeuge viel Unterstützung leisten.

kein UML

Die Ähnlichkeiten zur UML sind erkennbar, aber die Tatsache, dass das CIM-Meta-Schema nicht über die Meta Object Facility definiert und auch strukturell Unterschiede vorhanden sind, begünstigt die mangelnde Werkzeugunterstützung. Wären beide Voraussetzungen gegeben, wäre es möglich CIM als UML-Erweiterung zu definieren und bestehende UML-Modellierungswerkzeuge um CIM-Funktionalität zu erweitern. Bestrebungen zum Abgleich mit der UML haben stattgefunden, indem beispielsweise eine Abbildung vom CIM-Meta-Schema auf das UML-Meta-Modell beschrieben worden ist [Dis09c]. Bei der stetigen Weiterentwicklung der CIM-Spezifikation ist die Abbildung für aktuelle Versionen des CIMs nicht mehr gültig.

10 http://sourceforge.net/apps/mediawiki/sblim/index.php?title=Main_Page, Webseite, letzter Aufruf: 2015-01

schnelle Weiterentwicklung

Weiterentwicklung ist eine Voraussetzung zur Verbesserung, jedoch auch ein Hindernis bei der Entwicklung. Im Jahr 2011 sind 4 Versionen des CIM-Schemas veröffentlicht worden. Da es sich hier um Erweiterungen handelt und die Abwärtskompatibilität gewährleistet ist, sind diese Änderungen nicht direkt schädlich. Ein Jahr Entwicklungszeit für die Fertigstellung einer markttauglichen CIM-Implementierung ist realistisch, das Ergebnis demnach aber schon wieder veraltet. Es stellt sich immer die Frage nach einer Aktualisierung des CIM-Schemas, der Unterstützung der neueren Version eines CIM-Profildokumentes oder der Anwendung der neuesten Spezifikationsvorlagen.

3.4 Strukturelle Eigenschaften der Verwaltungssoftware

Aus den Grundlagen, die in den vorangegangenen Abschnitten beschrieben worden sind, können nun strukturelle Eigenschaften für die Entwicklung der Verwaltungssoftware abgeleitet werden.

Verteilung der Verwaltungssoftware

Es gibt die unterschiedlichen Varianten von Verwaltungssoftware, entsprechend für die verschiedenen Arten von Server-Systemen. Im Folgenden werden drei Varianten vorgestellt.

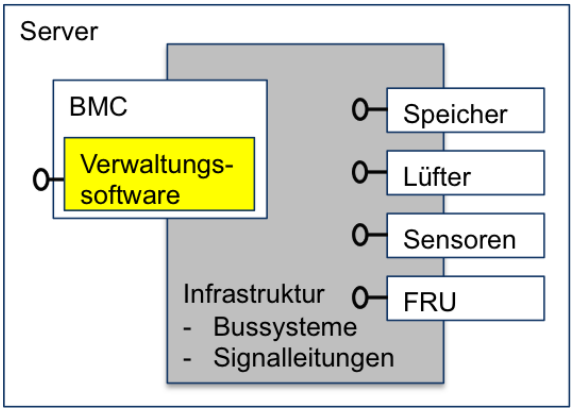


Abbildung 3.11: *Verwaltungssoftware im einzelnen Server*

Variante 1

Die erste Variante ist die Firmware eines Servers, die eine Out-Of-Band-Schnittstelle zur Verwaltung des einzelnen Servers bedient. Die Verwaltungssoftware eines einzelnen Servers wird im BMC umgesetzt. Der BMC greift über

Bussysteme und Signalleitungen auf Serverkomponenten zu (siehe Abbildung 3.11).

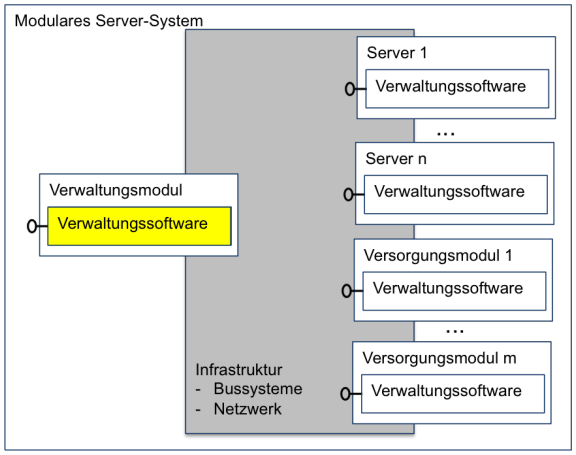


Abbildung 3.12: *Verwaltungssoftware im modularen Server-System*

Variante 2

Die zweite Variante ist die Verwaltungssoftware, die in einem Verwaltungsmodul im modularen Server-System ausgeführt wird. Wiederum wird über eine definierte Infrastruktur auf die zu verwaltenden weiteren Module zugegriffen. In diesem Fall sind Netzwerkverbindungen und Bussysteme vorherrschend (siehe Abbildung 3.12).

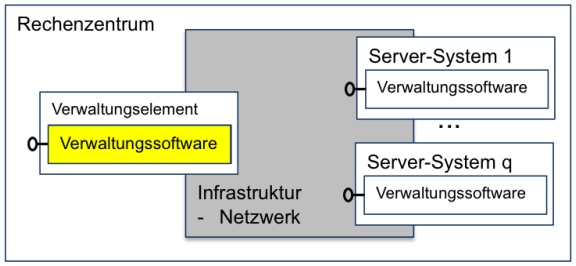


Abbildung 3.13: *Verwaltungssoftware im Rechenzentrum*

Variante 3

Eine dritte Variante ist ein komplexes Management-System für ein Rechenzentrum. Im Rechenzentrum übernimmt ein übergeordnetes Verwaltungselement die Aufgabe, komplexe Verwaltungssoftware auszuführen. Die Infrastruktur umfasst hier fast ausschließlich Netzwerkverbindungen (siehe Abbildung 3.13).

Gemeinsamkeiten der Struktur

Obwohl sich die Varianten und die darauf aufbauenden Infrastrukturen unterscheiden, gibt es gemeinsame Strukturen bei der Entwicklung der Verwaltungs-

software. Die Infrastruktur, an der die Verwaltungssoftware angeschlossen ist, muss von der Verwaltungssoftware gekannt werden, ebenso muss die Identifikation der verbundenen Elemente möglich sein. Die Spezifikation der Verwaltungsschnittstellen der zu verwaltenden Elemente muss vorliegen. Die mögliche Funktionalität der Verwaltungssoftware wird durch die zu verwaltenden Elemente geprägt. Zusammenfassend finden sich in allen Server-Systemen die folgenden Strukturelemente wieder:

- Verwaltungselement mit Verwaltungssoftware
- Infrastruktur
- Zu verwaltende Elemente mit internen Verwaltungsschnittstellen

Gemeinsamkeiten Verwaltungssoftware

Strukturelle Eigenschaften der Verwaltungssoftware selber sind ebenfalls vorhanden. Betrachten wir wieder SNMP, IPMI oder CIM, erkennen wir in allen Fällen eine strukturelle Definition der Verwaltungsinformationen in Form einer MIB, diversen SDRs oder des CIM-Schemas. In allen Fällen muss die Definition der Daten OEM-spezifisch erweiterbar sein. Weiterhin definieren alle Standards eine Architektur und legen die Zugriffsprotokolle auf die Verwaltungsdaten fest (vgl. Abbildung 3.14).

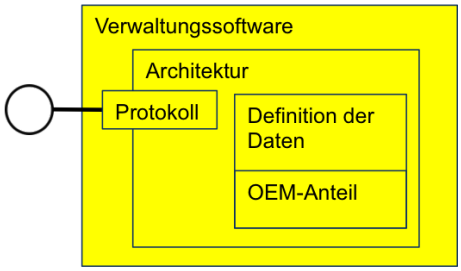


Abbildung 3.14: *Strukturelle Elemente der Verwaltungssoftware*

Ausblick

In dem folgenden Abschnitt wird unter anderem die Ist-Methode der Spezifikation der Server-System-Architektur ermittelt. Die hier identifizierten Strukturelemente sind hilfreich, wenn die Verbesserungs-Analyse durchgeführt wird. Die Ist-Methode muss daraufhin untersucht werden, inwieweit die notwendigen strukturellen Elemente eines Server-Systems betrachtet und berücksichtigt werden.

3.5 Anwendungsbeispiele

Aufgaben

In den verschiedenen Projektphasen des Forschungs- und Entwicklungsprojekts bei Fujitsu Technology Solutions GmbH ist die Entwicklung von modularen Server-Systemen begleitet worden. Dieser Abschnitt gibt eine Übersicht über die Aufgaben beim Projektpartner und definiert die in dieser Arbeit betrachteten Anwendungsbeispiele.

MMB

Die erste Aufgabe war die Verbesserung der Spezifikation für das Verwaltungsmodul des Blade-Server-Systems BX900 mit dem Namen Management-Blade (MMB). Das Problem bestand in der schlechten Qualität der Verwaltungsfunktionalität des MMBs, die von einer Zuliefererfirma implementiert worden ist. Unter anderem war eine Schwachstelle die von FTS gelieferte Spezifikation. Sie war ungenau und unvollständig. Sie ließ Freiheiten bei der Interpretation zu, die von der Zuliefererfirma in Teilen auch fehlinterpretiert worden ist. Dieser Umstand ist durch sprachliche Schwierigkeiten und kulturelle Unterschiede begünstigt worden. Beim Testen und der Fehlerbehebung entstanden weitere Schwierigkeiten, denn was ungenau spezifiziert ist, ist ebenso nur ungenau testbar. Die Klärung der Spezifikation, die Qualitätsanalyse, die Fehlerbehebung sowie die Feststellung der Zuständigkeit für Fehler und deren Behebung verursachte einen erheblichen, für die Kosten relevanten, Aufwand. Die Erreichung des Zieltermins der Freigabe des MMBs ist gefährdet worden. In einer Neuentwicklung des MMBs sollten die Schwächen in der Spezifikation durch die Einführung von formalen und semi-formalen Sprachen und der Erstellung einer abstrakten Systemsicht reduziert werden. Die Benutzung der Sprachen (UML, SysML und DSLs) verbessert die Eindeutigkeit der Spezifikationsinhalte. Die abstrakte Systemsicht ermöglicht eine bis dahin nicht zur Verfügung stehende Übersicht des Gesamtsystems. Dadurch wird ein besseres Verständnis über im System vorherrschende Verwaltungsschnittstellen und deren Funktionalität erreicht, da die Verwaltungsschnittstellen schließlich vom MMB benutzt werden.

BX400

Die zweite Aufgabe im Forschungs- und Entwicklungsprojekt war die Unterstützung der Spezifikation des Blade-Systems BX400. Das BX400 ist ein neuartiges Blade-Server-System für den Small-to-Medium-Business- und Small-to-Medium-Enterprise-Markt (SMB-, SME-Markt). Die neuen Anforderungen an die Spezifikationsdokumente, abstrakte Systembeschreibung mit Darstellung der Abhängigkeiten, sollten durch die Unterstützung der SysML und der Erstellung eines neu-

en Spezifikationsdokuments, der Systemarchitektur-Spezifikation, erfüllt werden. Die zweite Aufgabe ermöglichte eine gesamtheitliche Systemsicht zu entwickeln, nicht nur aus der Perspektive des Verwaltungsmoduls, wie in der ersten Aufgabe.

MMB-NXT

Die dritte Aufgabe im Forschungs- und Entwicklungsprojekt war die Spezifikation des Management-Blades Next Generation (MMB-NXT). Aufgrund der Zuliefererthematik ist beim Projektpartner eine Eigenentwicklung des Verwaltungselements für Blade-Server-Systeme angestrebt worden. Das MMB-NXT sollte erstmals CIM-basierte Schnittstellen bereitstellen.

MMP und MMP-CX

Die vierte Aufgabe war die Überführung der Kenntnisse aus dem MMB- und MMB-NXT-Projekt in eine Referenzarchitektur für Verwaltungselemente mit dem Namen *Management Platform (MMP)* und einer Spezialisierung der Referenzarchitektur für das Verwaltungselement des CX1000 Cloud-Server-Systems mit dem *Namen Management Platform CX (MMP-CX)*. Die Anwendung und Weiterentwicklung der Systemsicht mit Spezifikation der gesamten Systemarchitektur und der Verbesserung des Entwicklungsprozesses für die Entwicklung CIM-basierter Schnittstellen stand im Vordergrund.

iRMC S4

Die fünfte Aufgabe war die Unterstützung der Spezifikation und Implementierung des integrated Remote Management Controllers in der vierten Version (iRMC S4). Der iRMC S4 ist eine Eigenentwicklung eines Baseboard Management Controllers (BMC) für verschiedene Server von Fujitsu Technology Solutions GmbH. Der iRMC S4 wird in Standalone Servern, Rack-Servern und Blade-Servern eingesetzt. Der iRMC S4 ist als erster BMC von Fujitsu Technology Solutions GmbH mit CIM-basierten Schnittstellen ausgestattet worden. Die Erkenntnisse aus den früheren Aufgaben, insbesondere die Entwicklungsmethodik, sollten wiederverwendet und weiter verbessert werden.

Detaillierte Informationen der Aufgaben können in den Abschlussberichten [Spi08], [Spi09], [Spi10], [Spi11], [Spi12] nachgelesen werden.

Anwendungsbeispiele

Durch die Bearbeitung der 5 Aufgaben konnten die 5 Herausforderungen bei der Server-System-Entwicklung identifiziert werden, aus denen die Teilaufgabe der Dissertation *Durchführung einer Methodenweiterentwicklung für die Server-System-Entwicklung* resultiert (siehe Abschnitt 3.1). Für die Dokumentation der Lösung der Teilaufgabe der Dissertation sind zwei Anwendungsbeispiele aus den Aufgaben des Projekts ausgewählt worden.

Anwendungsbeispiel 1
Darstellung der Methodenweiterentwicklung für die Spezifikation der Systemarchitektur von Server-Systemen.

Anwendungsbeispiel 2
Darstellung der Methodenweiterentwicklung für die Entwicklung CIM-basierter Schnittstellen in Server-Systemen.

3.6 Lösungsbaustein: Ermittlung Ist-Methode

Ist-Methode

Nachdem die Eigenschaften der Entwicklungsdomäne Server-System-Entwicklung bekannt sind, betrachten wir die Erstellung der Ist-Methode an den zwei Anwendungsbeispielen aus dem Kontext des Forschungs- und Entwicklungsprojekts bei Fujitsu Technology Solutions GmbH.

Vorgehen

Es soll die Aktivität *Ermittlung Ist-Methode* aus dem Vorgehen der Methodenweiterentwicklung (siehe Abschnitt 2.3.1) genutzt werden. Sie beschreibt das Methodenmetamodell als Eingabe-Artefakt. Zusätzlich können weitere Hilfsmittel die Aktivität unterstützen, um letztendlich die Ist-Methode zu definieren. Die komplexe Aktivität *Ermittlung Ist-Methode* ist bisher ungenügend detailliert worden und bietet dem Methodenentwickler wenig Unterstützung. Er weiß, dass eine Ist-Methode definiert werden soll, braucht aber weitere Handlungsanweisungen.

Die Aufgabe dieses Abschnitts wird in Abbildung 3.15 skizziert. Die allgemeine

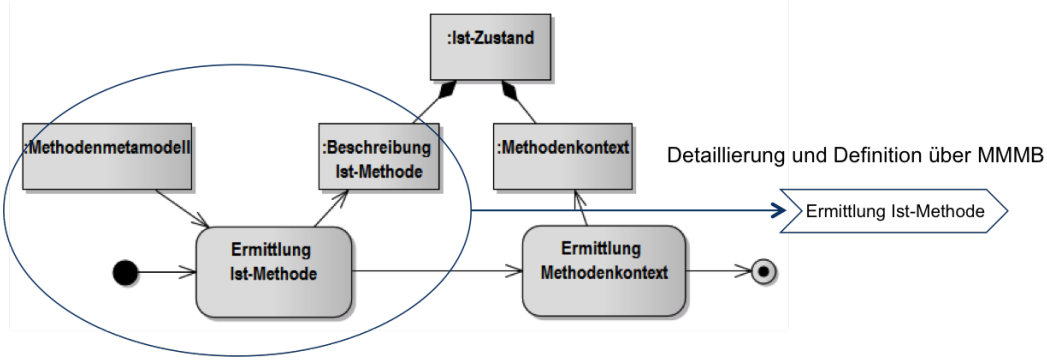


Abbildung 3.15: *Detaillierung Ermittlung Ist-Methode*

Aufbau

Aktivität *Ermittlung Ist-Methode* wird detailliert. Die detaillierten Handlungsan-

weisungen sind die Ergebnisse der Projektarbeit bei Fujitsu Technology Solutions GmbH. Die Detaillierung wird mittels der Sprache MMMB beschrieben. Das geschieht in Abschnitt 3.6.1. In den zwei weiteren Absätzen wird die detaillierte Aktivität *Ermittlung Ist-Methode* zum einen auf die System-Architektur-Spezifikation (Abschnitt 3.6.2) und zum anderen auf die CIM-Entwicklung (Abschnitt 3.6.3) angewendet.

3.6.1 Detaillierung der Aktivität *Ermittlung Ist-Methode*

Die Aktivität *Ermittlung Ist-Methode* ist Bestandteil der zusammengesetzten Aktivität *Ist-Analyse*, die aus Gründen der Übersicht in Abbildung 3.16 dargestellt wird.

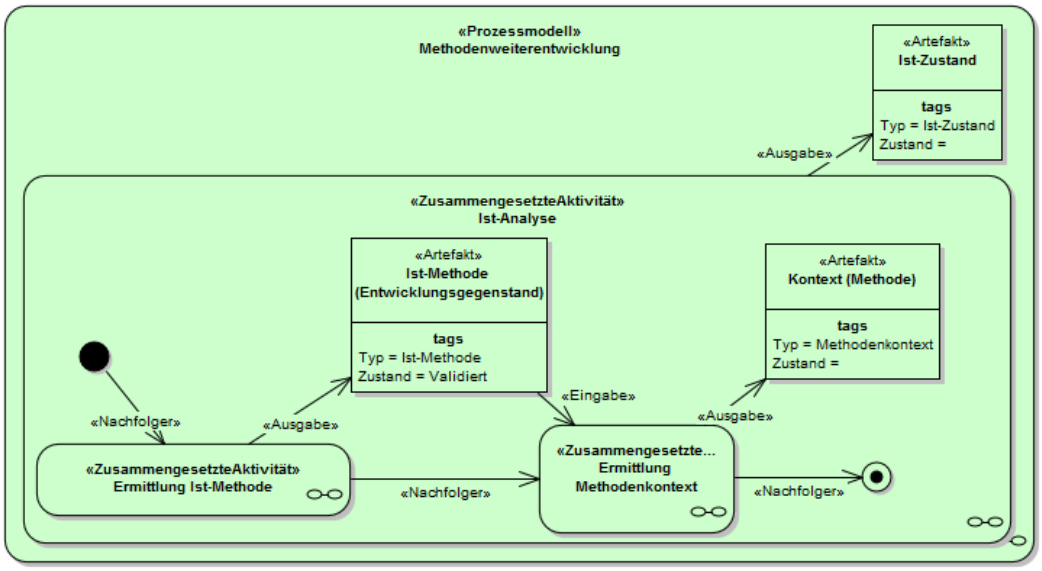


Abbildung 3.16: *Übersicht der Ist-Analyse*

Ist-Analyse

Die Ist-Analyse besteht aus den zusammengesetzten Aktivitäten *Emittlung Ist-Methode* und *Ermittlung Methodenkontext*. In diesem Abschnitt wird jedoch nur die *Ermittlung Ist-Methode* detailliert erläutert. In der folgenden Abbildung 3.17 werden ihre Details dargestellt.

Informationsgrundlage

Möchte der Methodenentwickler eine Ist-Methode beschreiben, benötigt er eine Informationsbasis, die notwendige Informationen über bestehende Methoden-

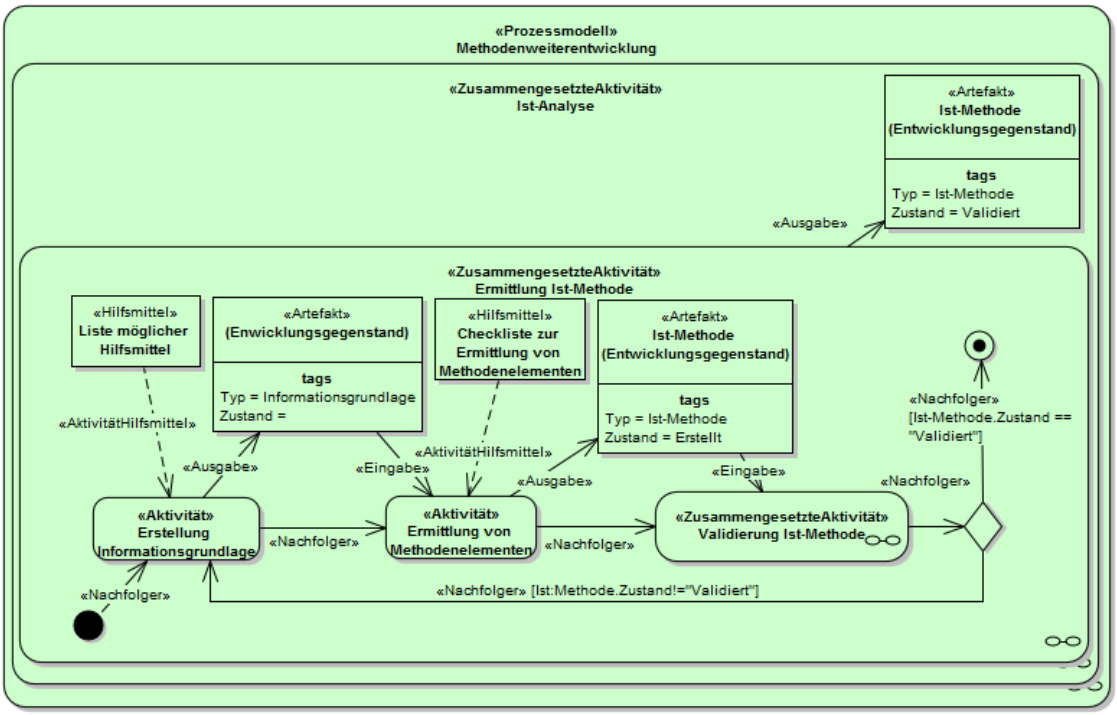


Abbildung 3.17: *Erstellung der Ist-Methode mit MetaMe++*

elemente bereitstellt. Die Informationsgrundlage muss erstellt werden. Sie besteht aus den unterschiedlichsten Hilfsmitteln. Eventuell sind explizite methodische Vorgaben vorhanden, beispielsweise in Form von Methodenhandbüchern. Es kommt allerdings auch vor, dass es unzureichend explizite Informationen zur Erstellung der Ist-Methode gibt. Weiterhin muss beobachtet werden, ob das tatsächliche methodische Vorgehen den expliziten Vorgaben entspricht. In beiden Fällen ist es notwendig implizite methodische Informationen zu ermitteln. Diese sind schwieriger zu ermitteln. Die impliziten Informationen müssen vom Methodenentwickler explizit gemacht werden. Die folgende Tabelle 3.7 summiert mögliche Hilfsmittel, die zur Erstellung der Informationsgrundlage genutzt werden können. Zusätzlich wird beschrieben, welche Elemente der Methode mit dem Hilfsmittel gefunden werden können.

Tabelle 3.7: *Mögliche Hilfsmittel zur Erstellung der Informationsgrundlage*

Hilfsmittel (Informationselement)	**Beschreibung**
Hilfsmittel zur Identifizierung expliziter methodischer Vorgaben	

Spezifikations-dokumente	Spezifikationsdokumente geben direkt Hinweise über zu erstellende Artefakte der Entwicklung. In der Regel wird ein Spezifikationsdokument direkt als Artefakt der Entwicklung angesehen. Ein Spezifikationsdokument stellt aber auch den Zusammenschluss der Beschreibung mehrerer Artefakte dar. Beispielsweise beschreibt eine Softwarearchitektur die verschiedenen Artefakte, Komponenten und Schnittstellen. Das Vorhandensein eines Spezifikationsdokuments ist bedingt durch die Notwendigkeit bei der Entwicklung oder als Vorgabe durch ein methodisches Vorgehen.
Methoden-dokumente	Methodendokumente, wie beispielsweise Methodenhandbücher, beschreiben explizit eine zu befolgende Methode. Der Inhalt eines Methodenhandbuchs ist von Methodenexperten entwickelt worden und das Ergebnis einer Analyse der zu bewältigenden Aufgaben. Alle Elemente einer Methode können beschrieben sein und die für das Methodenweiterentwicklungsprojekt relevanten Informationen müssen in die Beschreibung der Ist-Methode überführt werden.
Anleitungen	Anleitungen sind unter den verschiedensten Begriffen bekannt. Einige Beispiele sind Guidelines, How-To's oder Manuals. Eine Anleitung ist in der Regel eine weniger formale Beschreibung als ein Methodenhandbuch. Einfache Aktivitäten werden beschrieben. Schwierig oder aufwendig herzuleitende Vorgehensweisen werden meist nach einer erfolgreichen Durchführung dokumentiert. Eine Bewertung oder Analyse des dokumentierten Vorgehens ist nicht vorhanden. Eine Anleitung ist eine pragmatische Wissenskonsolidierung. Hinweise auf alle Elemente einer Methode können vorhanden sein.
Source Code	In der Softwareentwicklung ist der Source-Code das Endprodukt der Entwicklung. Der Source Code muss auf verschiedene Aspekte hin überprüft werden. Der Source Code folgt beispielsweise einer Struktur, eventuell sogar auch einer Architektur. Die Organisation des Source-Codes muss durch Softwarekomponenten erfolgen. Eine solche Softwarekomponente ist eventuell ein Hinweis auf ein Artefakt, welches im Vorfeld spezifiziert werden muss. In diesem Fall ist Reverse-Engineering die durchzuführende Tätigkeit. Ian Sommerville beschreibt mit dem Begriff Reverse-Engineering ebenfalls die Aufgabe der Verbesserung des Verständnisses über Design, Funktionalität und Spezifikation (siehe [Som07]).
Organigramme des Unternehmens	Organigramme des Unternehmens beschreiben deren organisatorische Struktur. Die Abteilungen mit Mitarbeitern, Leitern und Verantwortlichen werden dargestellt. Organigramme sind bei der Identifikation von Rollen nützlich.

(Software-) Werkzeuge	Bei der Entwicklung von (Software-)Systemen werden die unterschiedlichsten Werkzeuge benutzt. Sie helfen den Designern und Entwicklern bei ihrer täglichen Arbeit. Die benutzten Werkzeuge selber sind Hilfsmittel für die Durchführung von Aktivitäten und Prozessen. Durch eine Analyse der Werkzeuge ist auch die durchzuführende Aktivität ermittelbar.
Meetings	Meetings sind organisatorische Mittel zur Planung von Projekten oder Besprechung von Fragestellungen. Die Fragestellungen können Hinweise auf wichtige oder wiederkehrende Probleme geben für deren Lösung methodische Vorgaben bestehen. Weiterhin können angefertigte Protokolle der Meetings konkrete Aufgaben (Aktivitäten) und Verantwortlichkeiten (Rollen) dokumentieren.
informelle Informationen	Informelle Informationen sind beispielsweise Notizen, E-Mails, Tabellen oder Abbildungen. Sie entstehen während der Entwicklung eines Produkts, sind aber informell in ihrer Form. Das heißt, sie entstehen ad-hoc zur Klärung und Abgleich von auftretenden Fragestellungen oder aber zur Wissensdokumentation und -verbreitung. Diese informellen Informationselemente beschreiben wiederum Aktivitäten, Artefakte, Rollen und werden oft selber als Hilfsmittel benutzt. Sie sind sehr nützlich, da sie aus dem Grund der Notwendigkeit entstehen, sonst würde sich keiner den zusätzlichen Aufwand machen, diese Informationen zu erstellen.
Hilfsmittel zur Ermittlung impliziter Informationen	
Workshops / Interviews	Die Durchführung von Workshops und Interviews dient der Ermittlung von tatsächlichen Arbeitsabläufen. Das Ziel muss sein zu verstehen, wie tatsächlich gearbeitet wird.
Nachvollziehbarkeit der Entwicklung	Die Nachvollziehbarkeit der Ermittlung bedeutet die einzelnen Schritte in der Entwicklung des (Software-)Systems zu verstehen. Sind die Informationen nicht über explizite Informationen oder Workshops und Interviews zu erkennen, ist die Mitarbeit im Entwicklungsprojekt sinnvoll.

«Aktivität» Erstellung Informationsgrundlage

Es wird mit Hilfe der *Liste möglicher Hilfsmittel* die Informationsgrundlage erstellt. Für jedes Element in der Liste wird geschaut, ob im Methodenweiterentwicklungsprojekt die Möglichkeit besteht Zugang zu einem solchen Informationselement zu bekommen. Wichtig ist Zugriff auf explizite Informationen zu erlangen und die Protokollierung der ermittelten Informationen aus impliziten Quellen.

Der Zugriff auf die Informationsgrundlage muss während des gesamten Projekts der Methodenweiterentwicklung gewährleistet sein.

Ermittlung von Methodenelementen

In einem weiteren Schritt muss der Methodenentwickler nach bestehenden Elementen einer Methode suchen. Das Methodenmetamodell, welches auch die Grundlage der Sprache MMMB darstellt, beschreibt alle notwendigen Elemente einer Methode. Weiterhin können genau diese Elemente auch mit MMMB beschrieben werden. Auf Basis der Elemente des Methodenmetamodells ist eine Checkliste zur Ermittlung der möglichen Methodenelemente erstellt worden. Diese Checkliste ist vom Methodenentwickler auf Basis der Informationsgrundlage abzuarbeiten. Sie enthält die Anweisungen für jedes Methodenelement mit entsprechender Beschreibung zur Definition der Ist-Methode mit MMMB. Die folgende Tabelle 3.8 beschreibt die bis dato entwickelte Checkliste zur Ermittlung von Methodenelementen.

Tabelle 3.8: *Checkliste zur Ermittlung von Methodenelementen*

Identifiziere Methodenelement	**Anweisung zur Erstellung der Beschreibung der Ist-Methode**
Produktmodellelemente	
Artefakttyp	Artefakttypen definieren den Typ aller Zwischen- oder Endprodukte der Produktentwicklung. Erstelle für jedes Zwischen- oder Endprodukt ein Element «Artefakttyp» im «Produktmodell» der Ist-Methode.
Zustände von Artefakten	Jeder Artefakttyp ist um eine Zustandsbeschreibung erweiterbar, die die möglichen Zustände einer Instanz des Artefakttyps (Artefakt) beschreibt. Beispielsweise sind *erstellt*, *analysiert* und *implementiert* die erlaubten Zustände einer Anforderung.
Artefakttyp-verfeinerung	Es gibt komplexe und strukturierte Artefakttypen. Komplexe Artefakttypen können aus verfeinerten Artefakttypen bestehen. Dieser strukturelle Zusammenhang kann über die Beziehung «Artefakttypverfeinerung» beschrieben werden, indem der komplexe Artefakttyp mit dem verfeinerten Artefakttyp verbunden wird.
Fachliche Verfeinerung	Zwei Artefakttypen können nicht nur durch eine strukturelle, sondern auch durch fachliche Verfeinerung miteinander in Beziehung stehen. Wird dieser Zusammenhang in der Informationsgrundlage entdeckt, sind beide Artefakttypen mit der Beziehung «FachlicheVerfeinerung» miteinander zu verbinden.
Abhängigkeiten	Es können neben der strukturellen und fachlichen Verfeinerung unbestimmte Abhängigkeiten zwischen Artefakttypen bestehen. Diese Abhängigkeit wird dargestellt, indem beide Artefakttypen mit der «Abhängigkeit»-Beziehung verbunden werden.
Prozessmodellelemente	

Artefakt	Ein Artefakt ist die konkrete Instanz eines im Produktmodell definierten Artefakttyps im Prozessmodell. Für ein konkretes Zwischen- oder Endprodukt wird ein «Artefakt» im Prozessmodell erstellt und einem Artefakttyp aus dem Produktmodell zugewiesen. Gibt es keinen entsprechenden Artefakttyp, muss er erstellt werden. Ein Artefakt darf einen bestimmten Zustand einnehmen, der in der Zustandsbeschreibung des Artefakttyps beschrieben wird.
Aktivitäten	Aktivitäten sind alle durchgeführten Tätigkeiten der Produktentwicklung. Eine atomare, nicht strukturierbare Tätigkeit ist eine Aktivität. Erstelle für jede Aktivität ein Element «Aktivität» im Prozessmodell der Ist-Methode. Alle weiteren strukturierbaren Tätigkeiten sind zusammengesetzte Aktivitäten. Erstelle für jede strukturierbare Aktivität ein Element «ZusammengesetzteAktivät» im Prozessmodell der Ist-Methode.
Eingabe Artefakt	Für die Ermittlung von Eingabeartefakten müssen die notwendigen Artefakte für die Durchführung einer Aktivität bestimmt werden. Das entsprechende Artefakt und die «Aktivität» müssen mit einer «Eingabe»-Beziehung verbunden werden.
Ausgabe Artefakt	Für die Ermittlung von Ausgabeartefakten müssen alle Produkte, die bei Durchführung einer Aktivität entstehen, betrachtet werden. Das entsprechende Artefakt und die «Aktivität» müssen mit einer «Ausgabe»-Beziehung verbunden werden.
Nachfolger	Mit Hilfe der «Nachfolger»-Beziehung werden Reihenfolgen der Aktivitäten definiert. Sind offensichtliche Reihenfolgen von Aktivitäten erkennbar, sollten diese spezifiziert werden. In der Regel ergeben sich die Reihenfolgen der Aktivitäten von selbst. In den meisten Fällen werden sie durch die Eingabe- und Ausgabe-Artefakte bestimmt (Objektfluss). Die Überspezifikation der Reihenfolge ist nicht sinnvoll. Gibt es eine Menge von Aktivitäten, die praktisch parallel ausgeführt werden können, muss nicht notwendigerweise eine Reihenfolge vorgegeben werden, da so jeder Durchführer eine für ihn sinnvolle Reihenfolge wählen kann. Gibt es jedoch Erkenntnisse, die eine Reihenfolge für sinnvoller als eine andere erachten, ist es von Vorteil, eine solche Vorgabe zu spezifizieren. Die beiden «Aktivitäten» werden mit einer «Nachfolger»-Beziehung miteinander verbunden.

Phasen	Phasen in der Entwicklung werden häufig durch Meilensteine, als Ende einer Phase, festgelegt. Phasen können jedoch auch durch Übergabe von Zwischen- oder Endprodukten zwischen verschiedenen Organisationselementen oder Auftraggebern und Auftragnehmern definiert werden. Wird beispielsweise ein Pflichtenheft übergeben, ist die Phase *Produktdefinition* abgeschlossen. Gibt es Methodenhandbücher, muss überprüft werden, inwieweit Entwicklungsphasen vorgegeben sind. Sind keine Entwicklungsphasen ermittelbar, macht es Sinn, welche zu definieren, beispielsweise nach RUP, V-Modell, Wasserfallmodell oder ähnlichen Vorgaben. Für jede identifizierte Phase wird ein Element «Phase» im Prozessmodell erstellt. Die «Aktivitäten» müssen den Phasen zugeordnet werden.
Hilfsmittel	Die Suche nach Hilfsmitteln beginnt mit der Suche nach Dokumenten, die Hilfestellungen enthalten. How-To's, Guidelines, Benutzungshandbücher, Vorlagen, Whitepaper, Wikis, Onlinehilfen sind die gängigsten Ressourcen. Werkzeuge die direkt im Entwicklungsprozess eingesetzt werden, wie IDE's, Parser, Testwerkzeuge und weitere, oder aber auch Werkzeuge, die indirekt zur Durchführung von Aktivitäten genutzt werden, wie Text-, Tabellen und Zeichenwerkzeuge, stellen Hilfsmittel dar. Für jedes Hilfsmittel wird ein Element «Hilfsmittel» im Prozessmodell erstellt und mit der Beziehung «ProzessmodellHilfsmittel» an die entsprechende Aktivität angefügt.
Rollen	Rollen beschreiben bestimmte Kompetenzen bei der Produktentwicklung, beispielsweise Produktmanager, Systemarchitekt oder Programmierer. Jede Rolle ist von einer oder mehreren Personen zu erfüllen, die die definierte Kompetenz mit sich bringen. Für jede identifizierte Kompetenz wird ein Element «Rolle» im Prozessmodell erstellt.
Teilnehmer	Ist eine Rolle als Teilnehmer bei einer Aktivität notwendig, wird die Rolle über die «Teilnehmer»-Beziehung mit der Aktivität verbunden.
Durchführer	Wird eine Rolle als Durchführer, das heißt als Verantwortlicher, bei einer Aktivität benötigt, wird die Rolle über die «Durchführer»-Beziehung mit der Aktivität verbunden.
Meilensteine	Meilensteine definieren hervorgehobenen Zustände im Entwicklungsprozess. In der Regel sind die Übergänge von einer Entwicklungsphase in eine weitere durch Meilensteine definiert. Ein Meilenstein beschreibt in der Definition von MMMB den Zusammenschluss eines oder mehrerer Artefakte in einem definierten Zustand. Für jeden Meilenstein muss ein Element «Meilenstein» im Prozessmodell erstellt werden.
Ergebnisse	Die aggregierten Artefakte eines Meilensteins müssen über die «Ergebnis»-Beziehung mit dem Meilenstein verbunden werden.

«Aktivität» Ermittlung von Methodenelementen

Eingabe: Informationsgrundlage
Die *Checkliste zur Ermittlung von Methodenelementen* wird auf der Informationsgrundlage angewendet.
Jedes identifizierte Methodenelement wird mit der Sprache MMMB beschrieben.
Ausgabe: Ist-Methode

Überprüfung der Ist-Methode

Ist mit Hilfe der Sprache MMMB eine Ist-Methode definiert, muss diese auf Validität hin überprüft werden. Das geschieht in der ZusammengesetztenAktivität *Validierung Ist-Methode*. Zwei Überprüfungen können durchgeführt werden, wie in Abbildung 3.18 dargestellt wird.

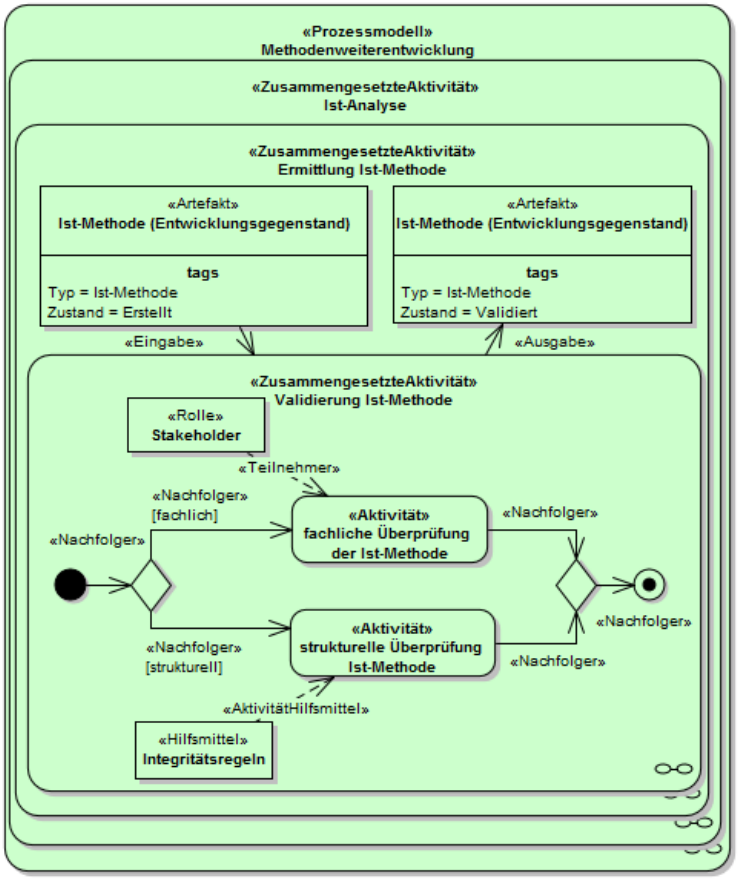


Abbildung 3.18: *Validierung der Ist-Methode mit MetaMe++*

fachlich

Zum einen ist eine *fachliche Überprüfung der Ist-Methode* mit Hilfe der Stake-

holder der Methode möglich. Stakeholder sind die Personen, die an der Methode interessiert sind. In erster Linie sind das die Anwender.

«Aktivität» fachliche Überprüfung der Ist-Methode

Die Ist-Methode wird nach Fehlinterpretationen, die bei der Auswertung der Informationsgrundlage entstanden sein müssen, überprüft. Die Aufgabe wird mittels Reviews oder Workshops mit den Stakeholdern der Methodenweiterentwicklung durchgeführt.

strukturell

Zum anderen ist es möglich eine *strukturelle Überprüfung der Ist-Methode* mit Hilfe von Integritätsregeln durchzuführen, um die konstruktiven und strukturellen Fehler in der Ist-Methode zu finden. Eine Übersicht über die Integritätsregeln gibt die folgende Tabelle 3.9.

Tabelle 3.9: *Integritätsregeln zur Überprüfung der Struktur der Ist-Methode*

ID	Regel
01	Jedes «Artefakt» im Prozessmodell ist über einen «Artefakttyp» im Produktmodell getypt.
02	Jeder genutzte Zustand eines «Artefakt»-Elements ist im Zustandsmodell des «Artefakttyp»-Elements definiert.
03	Zwei Artefakte, deren Artefakttypen im Produktmodell mit einer fachlichen Verfeinerung verbunden sind, müssen im Prozessmodell in einem Objektfluss liegen, bei dem das abstrakte Artefakt ein Vorgänger vom spezialisierten Artefakt ist.
04	Artefakte im Prozessmodell müssen entweder als Eingabe oder Ausgabe mit einer Aktivität verbunden sein.
05	Es darf keine unendliche Schleife über «Nachfolger»-Beziehungen im Prozessmodell definiert werden.
06	Sind «Phasen» im Prozessmodell definiert, muss jede «Aktivität» einer Phase zugeordnet sein.
07	Jedes im Prozessmodell vorhandene «Hilfsmittel»-Element muss über eine «ProzessmodellHilfsmittel»-Beziehung mit einer «Aktivität» verbunden sein.
08	Jedes im Prozessmodell vorhandene «Rolle»-Element muss über eine «Teilnehmer»- oder «Durchführer»-Beziehung mit einer «Aktivität» verbunden sein.
09	Jedes im Prozessmodell vorhandene «Meilenstein»-Element muss über eine «Ergebnis»-Beziehung mit mindestens einem «Artefakt»-Element verbunden sein. Das «Artefakt»-Element muss einen Zustand definieren.

«Aktivität» strukturelle Überprüfung Ist-Methode

Überprüfung der Integritätsregeln auf dem Modell der Ist-Methode

Korrektur

Fehler aus beiden Aktivitäten müssen korrigiert werden. Dafür sind Iterationen vorgesehen, die eine Anpassung und Erweiterung der Informationsgrundlage sowie die wiederholte Anwendung der Checkliste erlauben. Auf diese Weise können weitere Informationen bei der Ermittlung der Ist-Methode berücksichtigt sowie die bestehende Ist-Methode angepasst werden.

Abschluss

Sind alle Beteiligten der Meinung, die Ist-Methode entspricht den aktuellen Stand, es stehen keine offene Fragen im Raum und die Integritätsregeln gelten, wird der Status der Ist-Methode von *Erstellt* auf *Validiert* geändert. Die Ermittlung der Ist-Methode ist abgeschlossen.

Die für die «ZusammengesetzteAktivität» *Ermittlung Ist-Methode* relevanten «Artefakttypen» sind in der Abbildung 3.19 zusammengefasst.

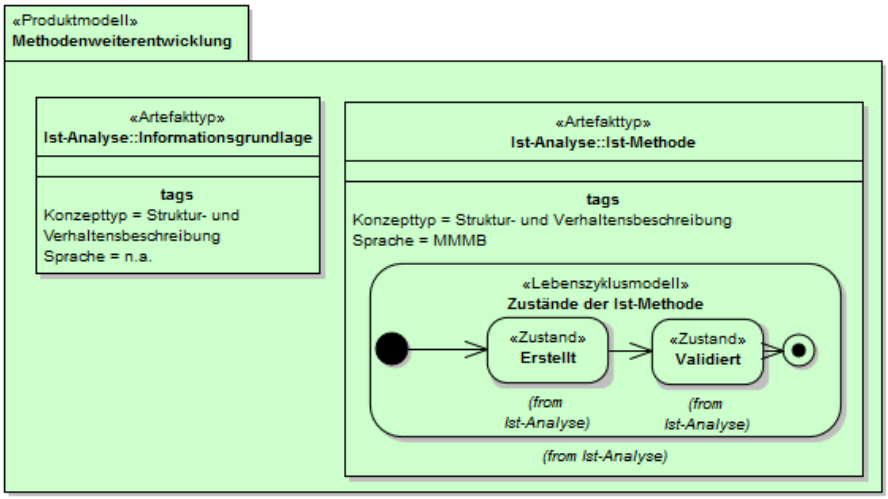


Abbildung 3.19: *Erstellung der Ist-Methode mit MetaMe++*

Informationsgrundlage

Die Informationsgrundlage wird aufgrund ihrer vielfältigen Ausprägungen nicht näher definiert. Sie besteht aus den unterschiedlichsten Informationselementen. Der «Artefakttyp» Ist-Methode wird über die Sprache MMMB definiert. Weiterhin gibt das Lebenszyklusmodell die beiden Zustände *Erstellt* und *Validiert* an.

Ermittlung Ist-Methode

Die bisherige Definition der detaillierten Aktivität *Ermittlung Ist-Methode* mit ihren Hilfsmitteln ist nun hinreichend für die exemplarische Anwendung auf die Anwendungsfälle der Server-System-Entwicklung bei FTS. Die Ergebnisse werden in den folgenden Abschnitten vorgestellt.

3.6.2 Ist-Methode für das Anwendungsbeispiel der System-Architektur-Spezifikation

Beispiele der Systementwicklung

Die Ergebnisse der Ist-Methodenerstellung werden in diesem Abschnitt exemplarisch für die Server-System-Entwicklung vorgestellt. Das Anwendungsbeispiel befasst sich mit der Entwicklung eines Blade-Server-Systems. Konkret sind die Erkenntnisse der Entwicklung der Systeme BX 900 und BX 400 betrachtet worden.

Informationsgrundlage

Die Informationsgrundlage besteht aus einer Reihe an Dokumenten. Insbesondere die Inhalte der von FTS bereitgestellten Dokumente *Methodenhandbuch - Hardware* und *Methodenhandbuch - Software* haben die initialen Informationen geliefert. Sie definieren eine Reihe von Meilensteinen und Prozessschritten zur Erstellung der geforderten Artefakte für einen Meilenstein. Weiterhin werden die Rollen genau spezifiziert. Im Forschungs- und Entwicklungsprojekt ist durch die aktive Mitarbeit im Projekt, Zugriff auf das Wissen der Mitarbeiter vor Ort und der Validierung der initial modellierten Ist-Methode jedoch festgestellt worden, dass eigentlich nur die Meilensteine fest eingehalten werden, die ein Artefakt hervorbringen, welches für das Management sichtbar ist. Wie der Entwicklungsprozess zur Erstellung der Artefakte, die für das Management sichtbar sind, durchgeführt wird, obliegt der Entwicklung und wird von Mal zu Mal unterschiedlich organisiert. Aufgrund der Verzahnung von Hardware- und Softwareentwicklung verschwimmt die Grenze zwischen den beiden Entwicklungen.

Prozessmodell

Die komplexen Abläufe während der Entwicklung lassen sich auf das in Abbildung 3.20 vorgestellte Prozessmodell reduzieren.

Phasen

Der Entwicklungsprozess definiert die Phasen *Analyse/Planung* und *Entwurf/Design*. Es folgt danach die Phase *Realisierung/Implementierung*, diese ist jedoch für die System-Architektur-Spezifikation nicht relevant, da dort bereits mit der konkreten HW-Entwicklung bzw. der Software-Programmierung begonnen wird.

Analyse/Planung

In der Phase *Analyse/Planung* werden Anforderungen gesammelt. Das geschieht

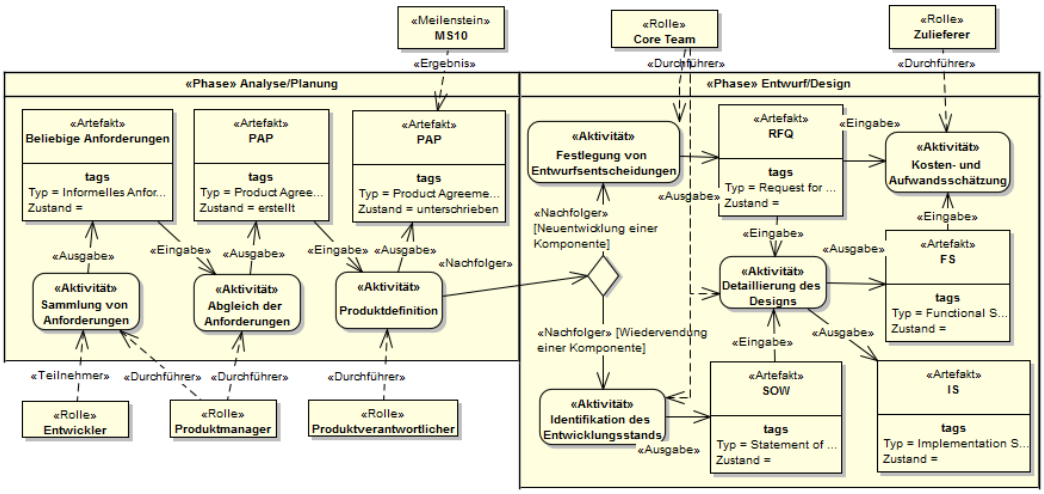


Abbildung 3.20: *Ist-Methode der System-Architektur-Spezifikation – Prozessmodell*

unter der Leitung des Produktmanagers mit Hilfe der Entwickler. Diese Anforderungen werden recht rudimetär in beliebiger Art und Weise dokumentiert. Excel-Tabellen, Word-Dokumente und Power-Point-Folien sind die Werkzeuge der Wahl, die für die informelle ungeregelte Sammlung von Informationen genutzt werden. Alle diese Informationen werden als Anforderungen angesehen. In einem weiteren Schritt sammelt der Produktmanager die für ihn relevanten Anforderungen für das zu entwickelnde Produkt und verfasst ein Product Agreement Paper. Dieses Dokument wird in der Aktivität *Produktdefinition* von den Produktverantwortlichen diskutiert, begutachtet und angepasst bis alle Produktverantwortlichen das Ergebnis vertreten können. Danach wird das Product Agreement Paper von allen Produktverantwortlichen unterzeichnet. Das Product Agreement Paper wird auch Produktvereinbarung genannt. Dieses Dokument bildet die Grundlage für die weitere Entwicklung und definiert den Meilenstein MS25. Der MS25 wird auch als Übergang von der Phase *Analyse/Planung* zur Phase *Entwurf/Design* betrachtet.

Das Product Agreement Paper (PAP) enthält eine inhaltlich formlose Definition des Gesamtsystems. Eine formelle Gesamtarchitektur wird nicht definiert. Es werden lediglich die Module des Systems aufgezählt und eine Beschreibung derer Schnittstellen und Funktionalität angegeben. Organisatorisch wird für jedes Modul ein Core Team gebildet. Da ein Modul Hard- und Software-Aspekte enthält, ist das Core Team ein Team interdisziplinärer Kompetenzen aus den verschiedenen Entwicklungsabteilungen. Für jedes Modul ist zu ermitteln, ob es bereits vorhanden ist oder ob es eine Weiterentwicklung eines bestehenden Mo- PAP

duls ist. In diesem Fall wird dessen Entwicklungsstand in einem Statement-of-Work-Dokument festgehalten. Muss ein Modul neu entwickelt werden, wird ein Request-for-Quote-Dokument angefertigt. Beide Dokumente sind ebenfalls informeller Natur und enthalten Beschreibungen in Form natürlicher Sprache und skizzenhafter Abbildungen. Bei bestehenden Modulen wird für bestimmte Aspekte des funktionalen Verhaltens eine Implementation Specification angefertigt, die die Inhalte des Statement-of-Work-Dokuments verfeinern. In der Regel wird ein neu zu entwickelndes Modul von Zulieferer-Firmen erstellt. Reicht einem Zulieferer das Request-for-Quote-Dokument nicht aus, um eine fundierte Kosten- und Aufwandsschätzung zu erstellen, werden die ungenügend spezifizierten Aspekte des Request-for-Quote-Dokuments in einer Functional Specification detailliert.

Letztendlich besteht die bisherige System Architektur Spezifikation aus einer Menge an verschiedenen Spezifikationsdokumenten. Teilweise werden die Inhalte neu erstellt oder aus alten Dokumenten von vorangegangenen Systementwicklungen kopiert.

Produktmodell

In der Abbildung 3.21 werden die grundlegenden Artefakttypen für das beschriebene Prozessmodell erläutert.

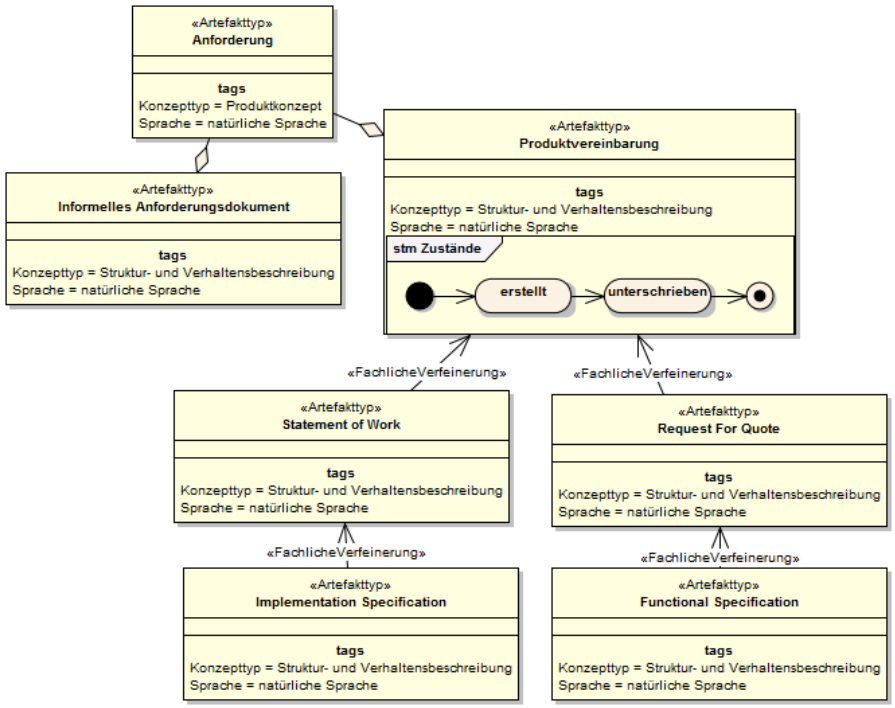


Abbildung 3.21: *Ist-Methode der System-Architektur-Spezifikation – Produktmodell*

Anforderung

Ein Artefakttyp ist eine Anforderung. Sie besteht aus einer informellen Beschreibung in natürlicher Sprache. Das macht es schwierig weitere Strukturinformationen im Produktmodell zu definieren. Anforderungen sind zum einen in den informellen Anforderungsdokumenten enthalten, denen ebenso wenig eine strukturelle Form zugrunde liegt, und zum anderen im Product Agreement Paper oder auch Produktvereinbarung. Dieses Dokument hat eine einzuhaltende äußere Form, jedoch sind die für die Spezifikation relevanten Inhalte ad-hoc strukturiert. Deshalb wird hier als Typ der Sprache die natürliche Sprache angegeben. Für das Prozessmodell sind die beiden Zustände *erstellt* und *unterschrieben* wichtig. Auf diese Weise wird auch der Meilenstein MS25 definiert. Die weiteren Dokumente *Statement-of-Work* und *Request-for-Quote* stellen jeweils fachliche Verfeinerungen der Produktvereinbarung dar. Ebenso wie das die Implementation Specification und Functional Specification für die Dokumente Statement-of-Work und Request-for-Quote tun. Diesen Dokumenten war es nicht möglich Zustände zuzuordnen, da sie in der Regel nie fertig sind. Es gibt in den seltensten Fällen einen Dokument-Lebenszyklus oder gar eine dokumentierte Versionierung zur Nachverfolgung der Dokumente. Sogar der Ablageort der Dokumente ist nicht definiert. Es passiert, dass ein Dokument entgegen der Vermutung gar nicht vorhanden ist oder vorhandene Dokumente schwer bis gar nicht gefunden werden.

Fazit

Das in diesem Abschnitt vorgestellte Prozess- und Produktmodell dokumentiert die Ist-Methode in dem für die System Architektur Spezifikation relevanten Ausschnitt der bestehenden Entwicklungsmethode der Server-System-Entwicklung bei FTS.

3.6.3 Ist-Methode für das Anwendungsbeispiel der CIM-Entwicklung

Ist-Zustand

Die Ergebnisse der Ist-Methodenerstellung werden in diesem Abschnitt exemplarisch für die CIM-Entwicklung vorgestellt. Das Anwendungsbeispiel befasst sich mit der Entwicklung CIM-basierter Schnittstellen. Der Ist-Zustand ist der Zustand bevor die Integration von CIM bei der Entwicklung des MMB-NXTs stattgefunden hat. Die CIM-Entwicklung war zu dem Zeitpunkt noch Neuland für FTS.

Informationsgrundlage

Die Informationsgrundlage besteht lediglich aus den von der DMTF bereitgestellten Spezifikationsdokumenten. Es gab weiterhin keine Kenntnisse für die CIM-Entwicklung bei FTS. Insbesondere [Dis10a] lieferte die Grundlage für die

Ermittlung der Prozessschritte und Artefakttypen, die für die Bereitstellung von CIM-basierten Schnittstellen notwendig sind.

technische Grundlage

Die technische Grundlage bestand in der verfügbaren Implementierung aus dem SBLIM-Projekt[11]. Dort ist eine Implementierung des CIM-Servers als HTTP-Server inklusive eines CIMOM mit dem Namen *Small Footprint CIM Broker* (SFCB) bereitgestellt worden. Weiterhin sind die Schnittstellen zum CIM-Repository sowie zu den CIM-Providern vorhanden. Der Zugriff auf die CIM-Provider ist standardisiert über das Common Manageability Programming Interface (CMPI). Abschließend wird ein CIM-XML-basierter CIM-Client zum Senden, Dekodieren und Empfangen von XML-basierten HTTP-Nachrichten zur Verfügung gestellt (siehe auch Abschnitt 3.3.3).

Entwicklungsaufgabe

Die Entwicklungsaufgabe lag in der Implementierung der CIM-Provider für das MMB-NXT Verwaltungsmodul. Die für die Entwicklung der CIM-Provider notwendigen Artefakttypen werden in der Abbildung 3.22 vorgestellt.

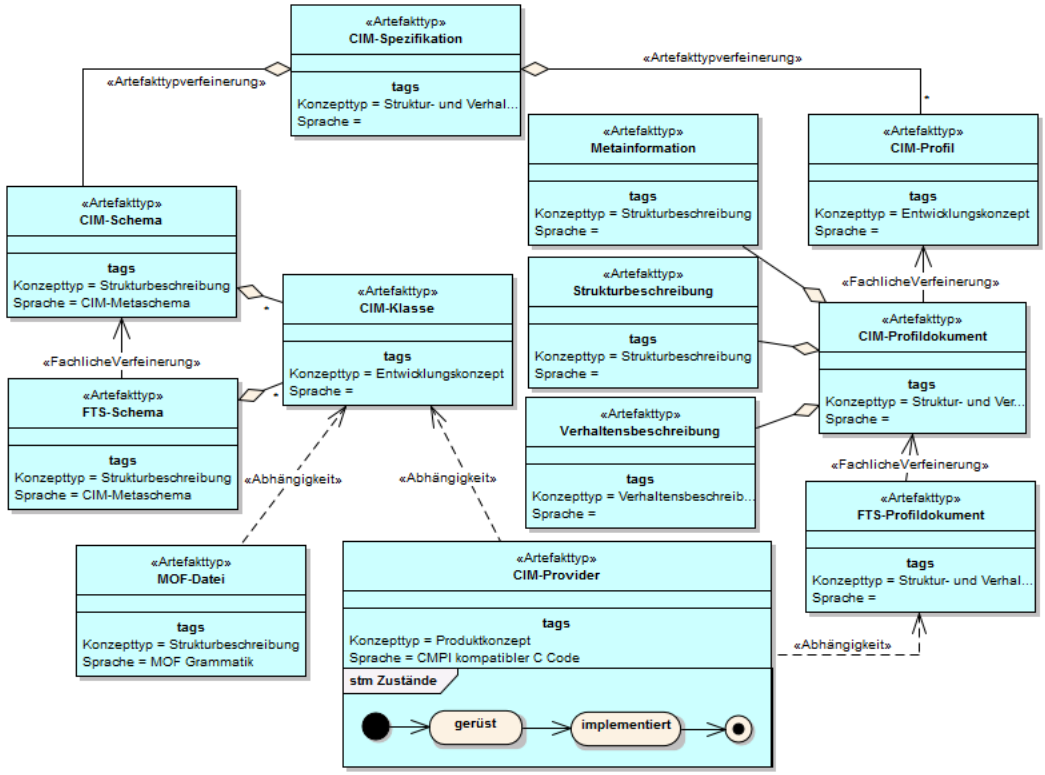


Abbildung 3.22: *Ist-Methode der CIM-Entwicklung – Produktmodell*

11 http://sblim.wiki.sourceforge.net/, letzter Aufruf: 2015-01

grundlegende Spezifikationsinhalte

Die CIM-Provider müssen auf Basis der CIM-Spezifikation erstellt werden. Die CIM-Spezifikation besteht zum einen aus den strukturellen Informationen des CIM-Schemas und zum anderen aus einer Menge an CIM-Profilen. Das CIM-Schema besteht aus einer Menge an CIM-Klassen. Eine CIM-Klasse stellt die strukturelle Spezifikation eines Informationselements dar. Es ist definiert über das CIM-Metaschema. Die Spezifikation wird mit einer formalen Grammatik erstellt und das Ergebnis ist eine MOF-Datei, welche eine maschinenlesbare Beschreibung der CIM-Klasse ist und vom CIMOM und dem CIM-Repository verwaltet werden kann. Ein CIM-Profil ist der logische Zusammenschluss verschiedener CIM-Klassen zu einer Verwaltungsdomäne. Es ist ein logisches Konstrukt zur Strukturierung der Spezifikation. Ein CIM-Profil wird mit einem CIM-Profildokument beschrieben. Das CIM-Profildokument enthält eine informelle Spezifikation. Erstens werden Metainformationen des CIM-Profils zur Verwaltung, Versionierung und zur Abgrenzung zu anderen CIM-Profilen definiert. Zweitens werden die strukturellen Informationen der zum CIM-Profil angehörigen CIM-Klassen dargestellt. Das ist eine informelle Repräsentation der Inhalte der MOF-Dateien. Drittens wird eine Verhaltensbeschreibung für die CIM-Provider angefügt, die beschreibt, wie eine CIM-Klasse und deren Inhalte zur Laufzeit instanziiert werden.

spezialisierte Spezifikationsinhalte

Die von der DMTF bereitgestellte Spezifikation ist allgemein gehalten und beschreibt die Struktur des CIM-Schemas sowie Regeln, die von einem CIM-Provider eingehalten werden müssen, damit eine dem Standard konforme Implementierung vorherrscht. Unterschiedliche Systeme, die über CIM-basierte Schnittstellen verwaltet werden sollen, bringen allerdings auch unterschiedliche Voraussetzungen zur Implementierung der CIM-Provider und der möglichen Funktionalität mit sich. Darum muss ein CIM-Profil für das jeweilige System spezialisiert werden. Diese Spezialisierung wird in diesem Fall mit einem FTS-Profildokument beschrieben. Das FTS-Profildokument detailliert die Vorgaben des CIM-Profildokuments für das spezifische System. Bei dieser Spezialisierung ist es zusätzlich erlaubt das CIM-Schema zu erweitern. Das geschieht mittels dem FTS-Schema, welches dann auch neue CIM-Klassen und MOF-Dateien hervorbringt.

Entwicklungsprozess

Der Weg von der CIM-Spezifikation zum CIM-Provider wird als Prozessmodell in der Abbildung 3.23 dargestellt. Für die Entwicklung von Verwaltungsfunktionalität auf Basis des CIMs müssen CIM-Provider programmiert werden. In einem ersten Schritt muss überlegt werden, welche Informationen über CIM beschrieben werden und welche CIM-Provider bereit gestellt werden sollen. Die DMTF stellt eine Menge an CIM-Profilen bereit. In der ersten Aktivität *Auswahl CIM-*

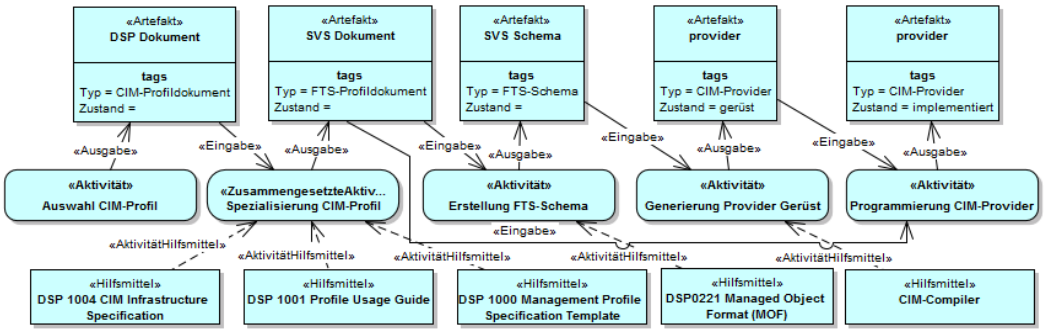


Abbildung 3.23: *Ist-Methode der CIM-Entwicklung – Prozessmodell*

Profil wird die Auswahl eines CIM-Profildokuments beschrieben. Dazu muss das CIM-Profildokument ausgewählt werden, welches die benötigten Verwaltungsinformationen beschreibt. Die DMTF stellt alle CIM-Profildokumente auf ihrer Webseite[12] bereit. Die CIM-Profildokumente haben eine Kennung zwischen DSP1000 und DSP1999. In der Zusammengesetzten Aktivität *Spezialisierung CIM-Profil* wird das CIM-Profil für ein FTS-spezifisches System fachlich verfeinert. Die fachliche Verfeinerung umfasst unterschiedliche Aktivitäten: Erstellung eines FTS-Profildokuments mit Definition der Metainformationen sowie der Struktur- und Verhaltensbeschreibung. Das Dokument DSP1004 liefert eine Beschreibung der fachlichen Grundlagen (siehe [Dis10a]), der DSP1000 Profile Usage Guide (siehe [Dis11]) eine Beschreibung zur Erstellung eines Profildokuments und DSP1001 eine Dokumentvorlage für das Profildokument (siehe [Dis07b]). Alle drei Dokumente sind benötigte Hilfsmittel bei der Erstellung des FTS-Profildokuments. Mittels des FTS-Profildokuments wird das FTS-Schema erstellt. Das geschieht in der Aktivität *Erstellung FTS-Schema*. An dieser Stelle müssen die MOF-Dateien für die FTS-spezifischen Klassen, die im FTS-Profildokument spezifiziert worden sind, bereitgestellt werden. Dabei hilft die MOF-Spezifikation DSP0221(siehe [Dis13]). Mit Nutzung von frei verfügbaren CIM-Compilern können die MOF-Dateien auf korrekte Syntax überprüft werden. Eine weitere Funktionalität der CIM-Compiler ist die Möglichkeit CIM-Provider-Gerüste mit CMPI-Schnittstelle zu generieren. Dadurch wird der C-Code erstellt, der CMPI-konform die Zugriffsfunktionen auf die Funktionalität der CIM-Provider enthält. In der letzten Aktivität *Programmierung CIM-Provider* muss dem CIM-Provider Gerüst die Funktionalität hinzugefügt werden, welche im FTS-Profildokument spezifiziert worden ist.

Zusammenfassung

Der vorgestellte Entwicklungsprozess erstellt das FTS-Profildokument, welches

12 www.dmtf.org, letzer Aufruf: 2013-12

als Benutzerspezifikation, aber auch Entwicklungsspezifikation genutzt wird. Die MOF-Dateien werden erstellt, die für CIM-Repository benutzt werden. Die CIM-Provider werden programmiert. Mit der Integration dieser Artefakte in die technische Umgebung wird die Verwaltungsfunktionalität über CIM-basierte Schnittstellen bereit gestellt.

3.7 Zusammenfassung

Zusammenfassung

Dieses Kapitel hat die Herausforderungen bei der Server-System-Entwicklung vorgestellt (siehe Abschnitt 3.1). Die einzelnen Eigenschaften der Architektur der Systeme (Abschnitt 3.2) sind herausgearbeitet und in Begriffsmodelle überführt worden (siehe Abbildung 3.1 und 3.2). Die Grundlagen der Systemverwaltung mit der entsprechenden Funktionalität, den Schnittstellen und den dazugehörigen Softwarearchitekturen sind erläutert worden (Abschnitt 3.3). Auf Basis der Erkenntnisse konnten allgemeine strukturelle Eigenschaften identifiziert werden (Abschnitt 3.4). Diese allgemeinen Ergebnisse stellen die Grundlage der Methodenweiterentwicklung in dem Bereich der Server-System-Entwicklung dar, wobei in diesem Kapitel die Ist-Methoden der beiden Anwendungsbeispiele exemplarisch aufgezeigt worden sind.

4 Situationsbeschreibung

Aufgabe

Der Methodenentwickler steht bei der Methodenweiterentwicklung vor der Aufgabe der Ermittlung des Ist-Zustands. Er hat aus dem vorangegangen Kapitel gelernt die vorhandene Ist-Methode zu beschreiben. Nun muss er zusätzlich den Methodenkontext definieren, der die Rahmenbedingungen beschreibt, in welchem die Ist-Methode genutzt wird. Bei dieser Aufgabe besteht nun das Problem der Identifizierung der Rahmenbedingungen, die einen Einfluss auf sein Methodenweiterentwicklungsprojekt haben.

Notwendigkeit Methodenkontext

Bei der Methodenweiterentwicklung soll eine Verbesserung der Methode durch Weiterentwicklung stattfinden. Das bedeutet, es gibt Schwächen in der Ist-Methode, die behoben werden sollen. Ein Teil der Schwächen besteht darin, dass die Ist-Methode nicht passend für ihren Methodenkontext konzipiert ist. Der Methodenkontext muss demnach bekannt sein, damit bei der Methodenweiterentwicklung die vorhandenen Schwächen erkannt werden können. Erst dann ist es möglich die Schwächen zu beheben und eine Soll-Methode passend zu dem Methodenkontext zu definieren.

Berücksichtigung bei der MWE

Das Vorgehen der Methodenweiterentwicklung sieht in der Phase *Ist-Analyse* die Aufgabe vor, den Ist-Zustand des Methodenweiterentwicklungsprojekts mit Ist-Methode (siehe Abschnitt 3.6) und Methodenkontext zu beschreiben. Der Methodenkontext definiert die zu berücksichtigenden Einflussfaktoren auf die Methodenweiterentwicklung. Sind die Einflussfaktoren bekannt, können in einem weiteren Schritt entsprechende Methodenanforderungen auf Basis der zu berücksichtigenden Einflussfaktoren erstellt werden (siehe Kapitel 5). Dieses Kapitel thematisiert die dafür notwendige Grundlage und detailliert die Aktivität *Ermittlung Methodenkontext*.

Methodenkontext = Situation

Das Forschungsgebiet Situation Method Engineering (SME) beschreibt die Berücksichtigung einer Situation als Notwendigkeit bei der Entwicklung von situationsgerechten Entwicklungsmethoden (vgl. Abschnitt 2.2). Eine Situation wird

über Situationsfaktoren beschrieben. Es gibt Situationsfaktoren, die während der Entwicklung einen Einfluss auf die Effektivität und Einhaltung der Qualität bei der Erstellung des Produkts haben. Der Begriff Methodenkontext ist im Abgleich zum Forschungsgebiet Situational Method Engineering als Synonym zum Begriff Situation zu betrachten. Die Einflussfaktoren entsprechen den Situationsfaktoren. Der strukturelle Zusammenhang zwischen Situation und Situationsfaktoren wird in Abbildung 4.1 dargestellt.

Abbildung 4.1: *Definition: Situationsfaktoren*

fehlende Übersicht über Situationsfaktoren

Ein Problem bei der Ermittlung der Situation ist bedingt durch das Fehlen einer definierten Menge an möglichen Situationsfaktoren. In der Literatur gibt es heterogene Quellen, die Situationsfaktoren identifizieren oder definieren.

zu berücksichtigende Situationsfaktoren

SME beschreibt die Aufgabe, eine Entwicklungsmethode an gegebenen Situationsfaktoren anzupassen. Ein Beispiel für Situationsfaktoren ist die Auflistung der 17 *contingency factors* von [SH96]. Die weiter reichende Literatur über SME bringt jedoch nur eine geringe Anzahl von tatsächlich identifizierten möglichen Situationsfaktoren mit sich (vgl. mit [BKKW07]). Die folgenden drei Feststellungen führen zu Schwierigkeiten bei der Ermittlung einer vollständigen Menge an möglichen Situationsfaktoren.

- In dem Bereich des *Assembly-Based* SME-Vorgehens wird erstens das Vorhandensein von Methodenanforderungen vorausgesetzt, bei denen Situationsfaktoren bereits beachtet worden sind.
- Zweitens wird der Fall einer bestehenden Methodenbasis skizziert und davon ausgegangen, dass ein Methodenfragment Metadaten bezüglich des möglichen Einsatzgebiets mit sich bringt (bspw. [RR01], [HS06], [MR06])). Ein Methodenfragment beschreibt bereits, in welcher Situation es eingesetzt werden kann. Aus mangelndem Zugriff auf solche Methodenbaukästen, die Methodenfragmente beschreiben, ist es schwierig auf mögliche Situationsfaktoren zu schließen.
- Die dritte Schwierigkeit ist der Umstand, dass die bestehende SME-Literatur keine umfassende Auflistung möglicher Situationsfaktoren bereitstellt. Es

werden lediglich Hinweise darauf gegeben, dass die Situationsfaktoren oder Methodenanforderungen erhoben werden sollen, jedoch fehlen detaillierte Anleitungen dazu.

Zuerst bleibt demnach, aus Gründen einer fehlenden Gesamtübersicht, die Aufgabe die bestehende Literatur nach möglichen Situationsfaktoren zu durchsuchen.

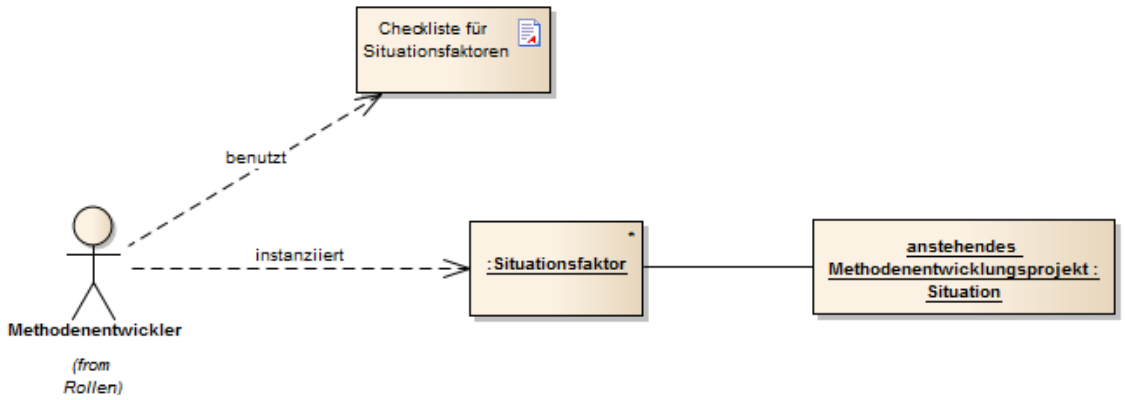


Abbildung 4.2: *Identifizierung von gültigen Situationsfaktoren auf Basis einer Checkliste möglicher Situationsfaktoren*

Checkliste möglicher Situationsfaktoren

Eine vorliegende, umfassende Checkliste von möglichen Situationsfaktoren ermöglicht die Identifikation gültiger Situationsfaktoren für das zu bearbeitende Methodenentwicklungsprojekt. Mit Hilfe der Checkliste möglicher Situationsfaktoren wird ein einfaches Vorgehen zur Ermittlung gültiger Situationsfaktoren definiert. Der Methodenentwickler schaut Eintrag für Eintrag in die Checkliste und überprüft für jeden möglichen Situationsfaktor, ob der Situationsfaktor für sein vorliegendes Methodenweiterentwicklungsprojekt gültig ist. Auf diese Weise ist es nicht möglich Situationsfaktoren zu *vergessen* (siehe Abbildung 4.2).

Die in diesem Kapitel verfeinerte Aufgabe wird demnach wie folgt beschrieben.

Teilaufgabe der Dissertation 6
Bereitstellung einer konsolidierten Checkliste möglicher Situationsfaktoren zur Ermittlung der gültigen Situationsfaktoren im Methodenweiterentwicklungsprojekt.

Struktur des Kapitels

Zur Lösung der Aufgabe wird wie folgt vorgegangen. Der folgende Abschnitt

4.1 diskutiert den Stand der Forschung und stellt eine Übersicht über die bekannte Literatur vor, die relevante Aussagen über Situationsfaktoren trifft. Der Abschnitt 4.2 stellt dar, wie Situationsfaktoren bewertet werden können. Aufgrund der Heterogenität der Situationsfaktoren fokussiert diese Arbeit jedoch lediglich auf das Vorhandensein eines Situationsfaktors. Auf Basis der Vorgaben aus der Literatur wird in Abschnitt 4.3 eine Übersicht über mögliche Situationsfaktoren vorgestellt. Das methodische Vorgehen wird im Lösungsbaustein *Ermittlung Methodenkontext* beschrieben und beispielhaft an den Anwendungsbeispielen dieser Arbeit, aus dem zugrunde liegenden Forschungs- und Entwicklungsprojekt, durchgeführt. Die Ergebnisse werden in Abschnitt 4.4.1 vorgestellt. Das Kapitel endet mit einer Zusammenfassung in Abschnitt 4.5.

4.1 Situationsfaktoren in der Literatur

Dieser Abschnitt gibt eine kurze Erläuterung der genutzten Quellen, damit der Leser eine Übersicht über den Kontext erhält, in dem die Diskussion über die Situationsfaktoren stattfindet.

[HSR10]: Der Artikel stellt eine State-of-the-Art-Übersicht dar und ist eine Einstiegslektüre in das Thema SME. In dem Artikel wird ein Technologiewechsel als möglicher Treiber zur Methodenanpassung beschrieben, aber auch von speziellen Anwendungsgebieten gesprochen. Es werden Web Services, Agile Entwicklung, Produktlinien ebenfalls wie der Bereich Unternehmensführung, Risikomanagement und Einhaltung von Normen unter dem Begriff *Technologiewechsel* vorgestellt. Es werden die speziellen Anwendungsgebiete Web Content Management Systeme, die Entwicklung dynamischer Systeme oder Requirements Engineering vorgestellt mit ihren Besonderheiten, die zu unterschiedlichen Methoden führen. Verschiedene Domänen bringen demnach eigene spezielle Konzepte mit sich, wobei sich einige Domänen Konzepte teilen können, wie beispielsweise die Anwendung von Requirements Engineering in den unterschiedlichen Domänen.

[HS05]: Dieser Übersichtsartikel erwähnt die Situationsfaktoren Sicherheit, Ausfallsicherheit, Real-Zeit, Kritikalitätssicherheit, Reifegrad der Organisation, Projektgröße sowie die Fertigkeiten der Entwickler im allgemeinen SME Kontext.

[CS05]: Bei der Auswahl von Methodenelementen wird in dieser Arbeit die Berücksichtigung von Auswahlkriterien beschrieben. Es wird erwähnt, dass zusammengesetzte Methodenfragmente Vertragseigenschaften berücksichtigen sollen. Es werden aber auch Interessen und Wünsche von Stakeholdern als Einflussfaktoren auf eine Methode genannt. Als weiterer Hinweis wird die besondere Berücksichtigung von domänenspezifischen Faktoren hervorgehoben.

[Coc00]: Der Artikel beschreibt den Begriff *The Big-M methodology*, also die Big-M-Methode. In der Definition von Big-M-Elementen wird Folgendes beschrieben. Entwickler mit bestimmten Fertigkeiten und Persönlichkeiten erfüllen Rollen im Projekt und arbeiten in verschiedenen Teams. Sie benutzen Techniken, um Arbeitsprodukte zu erstellen, die bestimmten Standards und Qualitätseigenschaften genügen. Die Techniken basieren auf Fertigkeiten und Werkzeugen, wobei die Werkzeuge wiederum die Standards etablieren. Die Teams beteiligen sich an den Aktivitäten, die im Entwicklungsprozess anfallen. Jede Aktivität passiert Meilensteine, die den Projektfortschritt beschreiben. Die Elemente werden durch Team-Werte bestimmt, die mit den Persönlichkeitswerten der einzelnen Entwickler und dem Entwicklungsprozess abgeglichen werden müssen. Die vorgestellte Definition setzt die Begriffe bereits in Relation zueinander (siehe Abbildung 4.3).

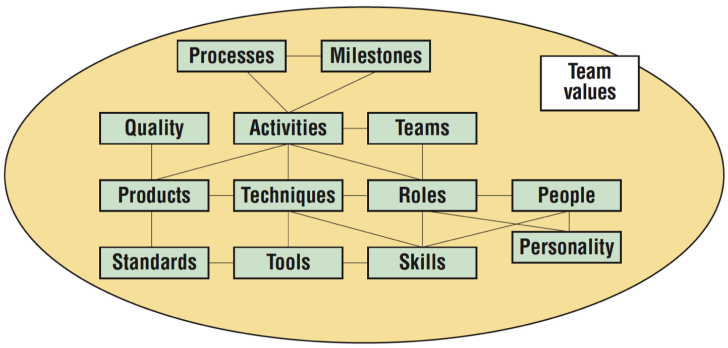


Abbildung 4.3: *Big-M-Methode [Coc00]*

[KA04]: Die Darstellung einer Methode zur Konfiguration von Methoden (engl. Method for Method Configuration (MMC)) steht im Vordergrund des Artikels. Ein wesentliches Konzept der Methode ist die Definition von Charakteristiken. *A Characteristic is a delimited part of a development situation, focusing on a certain*

problem or aspect which the method configuration aim to solve or handle. Die Methode erhebt zweierlei Arten von Charakteristiken, zum einen *Development Characteristics* und zum anderen *Project Characteristics*. Diese Einteilung ist ähnlich zu der detaillierteren Unterteilung der Situationsfaktoren in Domänen-, Organisations- und Projekteigenschaften. Die *Development Characteristics* sollen aus dem Bestand durchgeführter und geplanter Projekte ermittelt werden. Die *Project Characteristics* stellen eine konkrete Charakterisierung eines durchzuführenden Projekts dar.

[KDS07]: Die Autoren bauen mit ihrer Arbeit auf dem Vorhandensein einer Methodenbasis, die Methodenelemente bereitstellt, auf. Bei der Entwicklung oder Anpassung einer Methode müssen geeignete Methodenelemente aus der Methodenbasis ausgewählt werden. Damit der Auswahlprozess verbessert wird, muss eine Topologie von Projektcharakteristiken dargestellt werden. Die Projektcharakteristiken basieren auf der Anpassung der 17 *contingency factors* von [SH96], dem *Reuse Frame*-Konzept von [MR06] und den eigenen Erkenntnissen. Initial werden von den Autoren die Dimensionen *organizational*, *human*, *application domain* und *development strategy* zur Kategorisierung der Projektcharakteristiken vorgestellt.

[Rup09]: Dieses Buch über Requirements Engineering beschreibt im 15ten Kapitel eine Übersicht über nicht-funktionale Anforderungen. In dieser Übersicht gibt es einen Abschnitt über Anforderungen an durchzuführenden Tätigkeiten. Beispielsweise sollen nicht-funktionale Anforderungen beachtet werden, die beschreiben, *welche Tätigkeiten im Rahmen des Analyseprozesses durchzuführen sind* [Rup09]. Die nicht-funktionalen Anforderungen werden für das Entwicklungsziel definiert, haben jedoch Auswirkungen auf die Entwicklungsmethode zur Erreichung des Entwicklungsziels. Das Vorhandensein solcher Anforderungen muss dem Methodenentwickler bekannt sein, denn er ist derjenige, der diese Anforderungen in der Entwicklungsmethode berücksichtigen muss. Die Gliederung der nicht-funktionalen Anforderungen gibt weiterhin Hinweise auf Tätigkeiten, die im Rahmen der folgenden Prozessbestandteile vorgegeben werden können:

- Architektur- und Design-Prozess
- Implementierungsprozess
- Test-Prozess (Testvorbereitung, Durchführung und Dokumentation)

- Systeminstallationsprozess
- Auslieferungsprozess
- Wartungsprozess
- Prozess zur Erstellung von Benutzerinformationen und Schulungen
- Prozess zur Erstellung von Benutzungshandbüchern und Hilfesystemen
- Prozess zur Erstellung von Schulungs- und Lernunterlagen
- Prozess zur Erstellung von Administrationsdokumentation

Folgende Methodenelemente und -fragmente können durch nicht-funktionale Anforderungen vorgegeben sein:

- Anforderungsmanagement
- Projektmanagement
- Rollen
- Werkzeuge
- Dokumentation
- Qualitätsmanagement
- Konfigurationsmanagement
- Änderungsmanagement
- Risikomanagement
- Dokumentationsmanagement

Folgende Richtlinien sind ebenfalls zu berücksichtigen:

- Kommunikationsrichtlinien
- Durchführung von Meetings
- Standards

Bestehen solche nicht-funktionalen Anforderungen, sind sie als Situationsfaktoren zu betrachten, da sie die Methodenentwicklung beeinflussen.

[Bad11]: Die Autorin untersucht Taktiken der Einflussnahme auf das Entwicklungsziel von Auftragnehmern und Auftraggebern nach Vertragsabschluss. Dabei wird die Auftragnehmer-Auftraggeber-Beziehung charakterisiert. Beim Kundentyp wird zwischen Privat und Öffentlich unterschieden. Auch die Art des Entwicklungsprozesses wird thematisiert. Die Vertragsgestaltung wird auf Bezahlmodalitäten, Änderungswünsche, Vertragsgegenstand, Klarheit von Spezifikationsdo-

kumenten, Technische Unsicherheiten, Kostenausgleich und Einflussnahme des Kunden auf die Design-Phase charakterisiert.

[BWBM08]: Die Autoren monieren die fehlende Berücksichtigung von Situationsfaktoren bei der Optimierung von Software-Prozessen. Ihr Ansatz basiert auf der passenden Auswahl von Methodenfragmenten abhängig von dem Methodenkontext, welcher über Situationsfaktoren beschrieben wird. Die Identifikation der Situationsfaktoren ist mit einer empirischen Studie erarbeitet worden. Dabei sind Literaturrecherchen durchgeführt worden. Daraus resultierte eine initiale Liste von Situationsfaktoren, die im Feld durch Interviews und Analysen von Dokumenten untersucht worden ist. Ein wichtiger Aspekt bei der Studie ist die Bewertung des Einflusses eines Situationsfaktors auf die Veränderung im Prozess. Umso höher der Einfluss, umso eher muss ein Prozess bei sich veränderten Situationsfaktoren geändert werden. Bei der Studie ist die Liste bereits von unzuverlässigen Ergebnissen bereinigt worden. Das Ergebnis ist eine Liste von 27 Situationsfaktoren strukturiert in 5 Kategorien. Die 5 Kategorien lauten *Geschäftsbereich Charakteristiken*, *Kundencharakteristiken*, *Marktcharakteristiken*, *Produktcharakteristiken* und *Stakeholdercharakteristiken*. Jedem Situationsfaktor ist ein empirisch ermitteltes Gewicht zugeordnet. Neben [KDS07] wird eine der umfangreichsten Übersichten an Situationsfaktoren angeboten.

4.2 Bewertung der Situationsfaktoren

Bewertung

Die Bewertung von Situationsfaktoren im Hinblick auf ihre Auswirkungen auf eine Entwicklungsmethode ist schwierig. Bei der Bewertung muss der Methodenentwickler für einen möglichen Situationsfaktor überprüfen, ob er gültig für sein Methodenweiterentwicklungsprojekt ist.

Situation

Es ist möglich eine Situation über Situationsfaktoren zu charakterisieren. Bei der Erhebung von Situationsfaktoren beginnt das erste Problem. Beispielsweise erkennt [Bad11] Situationsfaktoren, definiert für sie jedoch keine Werte. Ebenso wenig tut dies [KA04], betrachtet seine erhobenen Charakteristika (Situationsfaktoren) aber schon als Enumeration Types. In [KDS07] werden mögliche Werte angegeben, teilweise aber auch eine weitere Typisierung. Werte reichen von Skalen von *low*, *normal*, *high* und *weak*, *medium*, *strong* über numerische Werte bis zu

vordefinierten Werten, beispielsweise *outsourcing*, *iterative*, *prototyping* und weitere für den Situationsfaktor *Development Strategy*. Es gibt also keine einheitliche Definition der Werte von Situationsfaktoren.

Werte

Die Definition der Werte für Situationsfaktoren führt zum zweiten Problem, der Interpretation der Werte. [KA04] erstellt beispielsweise eine Entscheidungsmatrix, der Prinzipien wie *Eine größere Gruppe, benötigt eine größere Methode* zugrunde liegen. Aus den Prinzipien abgeleitete Dimensionen sind *Personen*, *Kritikalität* und *Priorität*. Auf Basis der Entscheidungsmatrix können Erkenntnisse für charakterisierte Projekte ermittelt werden. Die im vorigen Abschnitt identifizierte Literatur enthält jedoch weitere Situationsfaktoren, die nicht zu einer von [Coc00] definierten Dimensionen passen, beispielsweise die von [Bad11] erkannten Vertragseigenschaften. Bisher gibt es keine verlässlichen Aussagen zur Gestaltung der Entwicklungsmethode abhängig von relevanten Vertragsgegenständen.

Interpretation

Bei der Interpretation der Werte von Situationsfaktoren hat sich im Forschungs- und Entwicklungsprojekt herausgestellt, dass die Bewertung von Situationsfaktoren subjektiv ist. Beispiele sind in der folgenden Aufzählung angegeben.

- Ein Entwicklungsteam mit 8 Mitarbeitern ist für die eine große Firma klein, für eine kleine Firma aber als groß zu bewerten.
- Die Einschätzung über Erfahrung der Projektmitarbeiter wird nicht in Bezug zu der zu erledigenden Aufgabe gesetzt. Ein langjähriger Mitarbeiter, dem viel Erfahrung im Bereich der Server-System-Entwicklung zugesagt wird, kennt neu einzusetzende Techniken und Paradigmen weniger gut als ein neuer Mitarbeiter.
- Die Entwicklung standardisierter Verwaltungsschnittstellen wird von einer Person als positiv bewertet, da viel Wissen über die Entwicklungstätigkeiten in Form von bereitgestellten Spezifikationen den Entwicklungsaufwand senkt. Eine zweite Person macht eine negative Bewertung, da er langfristig erhöhten Aufwand bei der Erhaltung der Standardkonformität befürchtet.

Schwierigkeiten

Passend zu dieser Schwierigkeit der Subjektivität räumt [KA04] ein, Eigenschaften des Endprodukts und des Entwicklungsteams falsch bewertet zu haben dass und sich daraus eine Ungleichmäßigkeit beim Arbeitsaufwand einzelnen Entwickler entwickelt hat.

gültige SF

Auch wenn die Wertebelegung für Situationsfaktoren nicht definiert ist und die anschließende Bewertung der Situationsfaktoren subjektiv ist, soll gezeigt werden, dass eine Charakterisierung der Situation sinnvoll ist. Das wird in Kapitel 5 gezeigt, wo auf Basis von gültigen Situationsfaktoren Anforderungen an die Methode erstellt werden. Es muss also möglich sein zu ermitteln, ob ein möglicher Situationsfaktor für die Methodenweiterentwicklung gültig ist, ohne ihn vorab bewerten zu können. Dafür muss definiert werden, was *gültig* bedeutet.

Beispiel

Nehmen wir als Beispiel den Situationsfaktor *Anzahl der Mitarbeiter in der Entwicklung*. Wir identifizieren 8 Mitarbeiter im Entwicklungsprojekt. Wir wissen, dass die Anzahl einen Einfluss auf die zu erstellende Soll-Methode hat. Die Zahl 8 kann aus Gründen der Subjektivität nicht als groß oder klein bewertet werden. Zur Bewertung der Gültigkeit muss nun untersucht werden, ob die Anzahl eine Herausforderung im Entwicklungsprojekt darstellt. Beispielsweise bei der Projektorganisation, Kommunikation oder Aufgabenverteilung. Sind Schwierigkeiten aus ähnlichen Projekten bekannt, besteht eine Erwartungshaltung beim Methodenentwickler, oder sind Schwierigkeiten bei der Nutzung der Ist-Methode bezüglich der Anzahl der Mitarbeiter erkannt worden, stellt das Charakteristikum eine Herausforderung dar. In diesem Fall ist der Situationsfaktor gültig, denn bei der Methodenweiterentwicklung soll die Herausforderung gelöst werden.

Allgemein gilt: Stellt ein möglicher Situationsfaktor eine Herausforderung dar, ist er gültig für das Methodenweiterentwicklungsprojekt.

Bewertung

Diese Erkenntnis hat Auswirkungen auf die Aufgabe der Bereitstellung einer Checkliste von möglichen Situationsfaktoren. Ein Situationsfaktor muss so definiert sein, dass seine Gültigkeit bewertet werden kann (auch wenn diese Aufgabe wiederum subjektiv ist).

Schema der SF

In der einfachsten Form muss die Checkliste eine *ja/nein*-Bewertung ermöglichen. Das bedeutet, dass die aus der Literatur ermittelten Situationsfaktoren nicht eins zu eins übernommen werden können, sondern umformuliert werden müssen, damit sie in folgendes Schema passen.

Tabelle 4.1: *Bewertungsschema*

Situationsfaktor	**gültig**	**Notizen**
SF 1	ja/nein	kurze Begründung

Die Spalte *Notizen* dient zur Argumentation für oder gegen die Gültigkeit des Situationsfaktors, damit das subjektive Empfinden begründet, diskutiert oder abgeglichen werden kann. Weiterhin kann eine informelle Wertevergabe hinterlegt werden.

4.3 Situationsfaktoren

Dieser Abschnitt schlägt eine Kategorisierung für Situationsfaktoren vor. Es werden die in der Literatur vorhandenen Informationen über Situationsfaktoren gesammelt, in konkrete Situationsfaktoren überführt und Kategorien zugeordnet.

mögliche Kategorien

Das zugrunde liegende Vorgehen aus Abschnitt 2.3 beschreibt implizit eine mögliche Kategorisierung von Situationsfaktoren. Der Methodenkontext ist definiert über Domänen-, Organisations- und Projekteigenschaften (siehe Abbildung 2.4). Die Eigenschaften können auf den Methodenkontext sowie auf den Methodenentwicklungskontext wirken, wobei der Methodenkontext in den unterschiedlichsten Domänen liegen kann, aber beim Methodenentwicklungskontext die Domäne offensichtlich auf *Methodenentwicklung* oder spezieller *SME* festgelegt ist (vgl.: Abschnitt 2.3).

Ausblick

Im Folgenden werden die Erkenntnisse aus den Literaturquellen konsolidiert. Die Literaturquellen identifizieren Situationsfaktoren, die verschiedene Kategorien und Unterkategorien definieren. Jede Kategorie wird in einem eigenen Abschnitt beschrieben.

4.3.1 Eigenschaften der Entwicklungsdomäne

Anwendungsgebiete

[HSR10] nennt die speziellen Anwendungsgebiete Web Content Management Systeme, die Entwicklung dynamischer Systeme oder Requirements Engineering. [CS05] und [KDS07] heben ebenfalls die besondere Berücksichtigung von domänenspezifischen Situationsfaktoren hervor.

Systemklassen

Erstens kann hier eine Unterteilung in die verschiedenen Systemklassen wie Datenbanksysteme, Dynamische Systeme, Eingebettete Systeme, Selbst-Adaptive Systeme, Realzeit-Systeme, GUI-Systeme und weitere verstanden werden.

Qualitäts-eigenschaften

Zweitens gibt es die Charakterisierung der Entwicklungsdomäne durch Qualitätseigenschaften. [KDS07] beschreibt die Qualitätseigenschaften Formalität, Beziehungen, Abhängigkeiten, Komplexität, Wiederholbarkeit, Variabilität und variable Artefakte. Die Eigenschaften Sicherheit, Ausfallsicherheit, Real-Zeit und Kritikalitätssicherheit werden aus [HS05] abgeleitet.

domänenspezifische Arbeitsprodukte

Drittens bringen die verschiedenen Domänen, wie in der Literaturquelle [HSR10] beschrieben wird, eigene spezielle Arbeitsprodukte mit sich. Einige Arbeitsprodukte können in mehreren Domänen vorkommen. Eine Zuordnung zu einzelnen Domänen ist nicht sinnvoll, wie das Beispiel Anforderungen und Anforderungspezifikationen zeigt. Die unterschiedlichsten Ausprägungen von Anforderungen können in den verschiedensten Domänen gefunden werden. Aus den genannten Gründen werden die Arbeitsprodukte als Unterkategorie der Kategorie *Eigenschaften der Entwicklungsdomäne* definiert. Eine vollständige Erhebung von Arbeitsprodukten für jede Domäne kann an dieser Stelle nicht geleistet werden und muss nach Bedarf von Domänenexperten ermittelt werden.

Techniken und Paradigmen

Viertens unterliegt jede Domäne den angewendeten Techniken und Paradigmen, beispielsweise beschreibt der in dem Artikel [HSR10] die Anwendungsgebiete Web Services, und Produktlinien explizit. Die Diskussion über die Anwendungstechnologie aus [KDS07] nennt Datenbanken, verteilte Entwicklung, GUI-basierte Entwicklung. In dieser Arbeit werden die Paradigmen modellbasierte Entwicklung, komponentenbasierte Entwicklung, Abstraktion und Separation-of-Concerns thematisiert. Diese Paradigmen werden ebenfalls in die Liste der Situationsfaktoren aufgenommen.

Hardware

Wir befinden uns thematisch bei der Softwareentwicklung. Durch die Anwendungsbeispiele aus dem Forschungs- und Entwicklungsprojekt ist erkannt worden, dass die Hardware-Entwicklung, insbesondere bei der Server-System-Entwicklung, ebenfalls eine besondere Charakterisierung notwendig macht. Daher muss fünftens die Hardware-Entwicklung betrachtet werden. Hier sind die identifizierten Situationsfaktoren Mechanik, Elektrotechnik und Thermodynamik.

Definition Domäne

In dieser Arbeit werden die Eigenschaften der Domäne durch die fünf Unterkategorien *Systemklasse*, *Qualitätseigenschaften*, *Arbeitsprodukte*, *Techniken und Paradigmen* und *Hardware-Entwicklung* definiert.

Tabelle 4.2: *Kategorie: Eigenschaften der Domäne*

Unterkategorie	**möglicher Situationsfaktor**	**gültig**
Systemklasse	Datenbanksystem	□
	Dynamisches System	□
	Eingebettetes System	□
	Selbst-Adaptives System	□
	Real-Zeit-Sytem	□
	GUI-basierte Systeme	□
	Web Content Management Systeme	□
Qualitätseigenschaften	Formalität	□
	Beziehungen	□
	Abhängigkeiten	□
	Komplexität	□
	Wiederholbarkeit	□
	Variabilität	□
	Variable Artefakte	□
	Sicherheit	□
	Ausfallsicherheit	□
	Real-Zeit	□
	Kritikalitätssicherheit	□
Wichtige Arbeitsprodukte der Domäne	*müssen vom Domänenexperten definiert werden*	
Techniken und Paradigmen	SOA	□
	Produktlinien	□
	modellbasierte Entwicklung	□
	komponentenbasierte Entwicklung	□
	Abstraktion	□
	Separation-of-Concerns	□
Hardware-Entwicklung	Mechanik	□
	Elektrotechnik	□
	Thermodynamik	□

4.3.2 Eigenschaften der Organisation

Die Dimension *organizational*, die in [KDS07] genannt wird, bestätigt das Vorhandensein der Kategorie *Eigenschaften der Organisation*. Die konkreten Charakteristiken wie Management Unterstützung, Wichtigkeit des Entwicklungsvorhabens, Auswirkungen, Zeitdruck, Ressourcenverfügbarkeit und Grad der Innovati-

on sind zu berücksichtigen. Die genannte Größe des Entwicklungsvorhabens wird ebenfalls den Organisationseigenschaften zugeordnet. Hiermit wird ein Entwicklungsvorhaben gemeint, welches sich über verschiedene Projekte erstreckt. Ansonsten muss der Situationsfaktor *Projektgröße* der Eigenschaften des Projekts betrachtet werden. In [HSR10] wird der Bereich Unternehmensführung und Risikomanagement als spezielles Anwendungsgebiet dargestellt, welche ebenfalls Charakteristika der Organisation bestimmen. [BWBM08] nennt explizit *Unternehmensstrategie* als Situationsfaktor, der ebenfalls der Kategorie Eigenschaften der Organisation zugeordnet wird.

Es werden keine Unterkategorien definiert.

Tabelle 4.3: *Kategorie: Eigenschaften der Organisation*

Unterkategorie	möglicher Situationsfaktor	gültig
	Unternehmensführung	☐
	Risikomanagement	☐
	Managementunterstützung	☐
	Wichtigkeit des Entwicklungsvorhabens	☐
	Auswirkungen	☐
	Zeitdruck	☐
	Ressourcenverfügbarkeit	☐
	Größe des Entwicklungsvorhabens	☐
	Grad der Innovation	☐
	Reifegrad	☐

4.3.3 Eigenschaften des Projekts

In [HS05] wird explizit die Projektgröße als Einflussfaktor auf eine Methode genannt. [Coc00] beschreibt ebenfalls die Projektgröße, aber auch die Problemgröße, Projektpriorität, Projektkritikalität und Anzahl der Mitarbeiter. [BWBM08] beschreibt in seiner Kategorie Geschäftsbereich neben dem Situationsfaktor *Größe des Entwicklungsteams* noch die *Größe des Business Teams* und trennt somit die Anzahl der Mitarbeiter in zwei Bereiche. Diese Eigenschaften werden als Eigenschaften des Projekts definiert. Im Forschungs- und Entwicklungsprojekt tritt die Situation auf, dass die Organisation über verschiedene Standorte, mit Zulieferfirmen sogar international, verteilt ist. Aufgrund der interdisziplinären Entwicklung sind verschiedene Standorte an einem Projekt beteiligt. Diese verteilte Entwicklung stellt ebenfalls Anforderungen an den Entwicklungsprozess. Die Eigen-

schaft verteilte Entwicklung wird als Situationsfaktor definiert. Es werden keine Unterkategorien definiert.

Tabelle 4.4: *Kategorie: Eigenschaften des Projekts*

Unterkategorie	möglicher Situationsfaktor	gültig
	Projektgröße	□
	Problemgröße	□
	Projektkritikalität	□
	Projektpriorität	□
	Anzahl der Mitarbeiter in der Entwicklung	□
	Anzahl der Mitarbeiter im Business-Team	□
	Anzahl der Stakeholder	□
	Benutzerbeteiligung	□
	verteilte Entwicklung	□

4.3.4 Menschliche Eigenschaften

Das Vorhandensein von Aspekten der Persönlichkeit und Team-Werte, wie in [Coc00] beschrieben wird, hat zweierlei Bedeutung. Einerseits sind die Persönlichkeiten und Team-Werte im (Projekt-) Team vorhanden, andererseits sind Persönlichkeiten und Team-Werte Eigenschaften für bestimmte Rollen. Für die erste Erkenntnis werden menschliche Eigenschaften als eigenständige Kategorie mit aufgenommen, da es jeweils eine Zuweisung zur Organisation oder zum Projektteam geben kann. Für die zweite Erkenntnis gilt, dass es menschliche Eigenschaften gibt, die Rahmenbedingung von Methodenelementen sein können. In [KDS07] wird die Dimension *human* genannt und ist ein weiteres Indiz für die Sinnhaftigkeit der Kategorie *Menschliche Eigenschaften*. Hier werden die konkreten Eigenschaften wie Widerstände, Konflikte, Expertise (Wissen, Erfahrung, und Fertigkeiten), Klarheit, Stabilität, Benutzerbeteiligung und Anzahl der Stakeholder genannt, wobei die letzten beiden Faktoren aber den Eigenschaften des Projekts (siehe 4.3.3) zugewiesen werden. Hier beziehen sich die Eigenschaften Widerstände, Konflikte, Klarheit und Stabilität jedoch auf die Einigung auf eindeutige Anforderungen unter den Stakeholdern. Die Eigenschaften der Expertise beziehen sich konkret auf die Eigenschaften der Entwickler. Es werden die beiden Unterkategorien *Eigenschaften der Stakeholder* und *Eigenschaften der Entwickler* gebildet.

Zu der Eigenschaft Expertise äußern sich auch [HS05] und [Coc00]. Sie nennen explizit Fertigkeiten der Entwickler als Einflussfaktoren. Die Eigenschaften der Expertise Wissen, Erfahrung und Fertigkeiten wird um die Eigenschaft Kompetenz erweitert.

Tabelle 4.5: *Kategorie: Menschliche Eigenschaften*

Unterkategorie	möglicher Situationsfaktor	gültig
Eigenschaften der Stakeholder	Widerstände	□
	Konflikte	□
	Klarheit	□
	Stabilität	□
	Wünsche	□
	Interessen	□
Eigenschaften der Entwickler	Wissen	□
	Erfahrung	□
	Fertigkeiten	□
	Kompetenz	□

4.3.5 Eigenschaften der Methodenelemente

Es gibt Eigenschaften, die Auswirkungen auf einzelne Elemente einer Methode haben.

Beispielsweise wird in [HSR10] die Einhaltung von Normen und Standards thematisiert. Sicherlich kann das Vorhaben der Einhaltung von Normen und Standards ein Unternehmensvorsatz sein, letztendlich müssen aber die Arbeitsprodukte innerhalb der Organisation den Normen und Standards gerecht werden. Auch wenn hier beispielsweise an die ISO 9001 *Qualitätsmanagement* gedacht wird, gelten letztendlich die eingehaltenen Anforderungen des Managementsystems. Normen und Standards sind demnach Eigenschaften von Arbeitsprodukten. Arbeitsprodukte sind Elemente der Methode. Hierzu passt auch die Berücksichtigung von nicht-funktionalen Anforderungen bezüglich *Standards* nach [Rup09]. Auch [BWBM08] berücksichtigt rechtliche Einflüsse, die hier ebenfalls mit in die Betrachtung einfließen.

[CS05] berücksichtigt bei den Auswahlkriterien von Methodenelementen Vertragseigenschaften von zusammengesetzten Methodenfragmenten. In dieser Ar-

beit wird die Meinung vertreten, dass eine Methode oder Teile der Methode ausschließlich zur Erstellung von Arbeitsprodukten und letztendlich zur Erstellung des Entwicklungsziels dienen. Demnach muss ein Arbeitsprodukt gemäß den Vertragseigenschaften erstellt werden. Wie auch Normen und Standards sind Vertragseigenschaften Eigenschaften von Arbeitsprodukten.

Es gibt ebenso Eigenschaften für die Benutzung von Werkzeugen, wie in der Diskussion in [KA04] herausgestellt wird, die zu berücksichtigen sind. In dem Artikel hatte das Vorhandensein einer Entwicklungsumgebung Einfluss auf die Methode. Allgemeiner formuliert, die Situationsfaktoren sind das Vorhandensein, die Anschaffung oder die Entwicklung von Entwicklungswerkzeugen. [BWBM08] beschreibt auch den *Reifegrad der Entwicklungsumgebung* als Situationsfaktor.

Ebenfalls werden neben den Rahmenbedingungen der Arbeitsprodukte und Werkzeuge, die Rahmenbedingungen der weiteren Methodenelemente Rollen, Aktivität, (Entwicklungs-)Prozess und Meilensteine (siehe Abbildung 2.1) mit als Unterpunkt aufgenommen.

Eine Teilmethode ist nach dem Methodenmetamodell in 2.1 durch die Kompositionsbeziehung der Methodenklasse zur Methodenklasse definierbar. [Rup09] fordert die Auseinandersetzung mit nicht-funktionalen Anforderungen für das Anforderungsmanagement, Projektmanagement, Qualitätsmanagement, Konfigurationsmanagement, Änderungsmanagement, Risikomanagement und Dokumentationsmanagement. Diese Themengebiete können in dieser Arbeit als komplette Methode oder Teile einer übergeordneten Methode verstanden werden. Das Herunterbrechen der nicht-funktionalen Anforderungen von der Teilmethodenebene auf einzelne detailliertere Methodenbestandteile, wie zum Beispiel Arbeitsprodukte, durch den unterschiedlichen Abstraktionsgrad, ist schwierig. Die Teilmethode wird deswegen als Unterkategorie eingeführt. In der Anforderungsanalyse für die zu entwickelnde Methode muss dann eine entsprechende Detaillierung, der hier als Situationsfaktoren aufgenommen Anforderungen, durch den Methodenentwickler durchgeführt werden.

Im Bezug zu den Anwendungsfällen dieser Arbeit, bei denen insbesondere die Analyse- und Entwurfsphase eines übergeordneten Entwicklungsprozesses betrachtet wird, ist es sinnvoll ebenfalls die Anforderungen an die Teilmethoden einer Systementwicklungsmethode – Analyse, Entwurf, Realisierung und Test – als Situationsfaktoren zu definieren.

[Rup09] beschreibt insbesondere nicht-funktionale Anforderungen an Prozessbestandteile. Die konkret genannten Prozessbestandteile werden als Eigenschaften der Unterkategorie *Prozesse* der Kategorie *Rahmenbedingungen für Methodenelemente* aufgenommen. Die zu berücksichtigenden Richtlinien *Kommunikationsrichtlinien* und *Durchführung von Meetings* beziehen sich ebenfalls auf Prozesse. Anforderungen an Rollen und Anforderungen an Werkzeuge werden jeweils dem Methodenelementen Rollen und Werkzeuge zugewiesen. Anforderungen an Dokumente wird der Unterkategorie Arbeitsprodukte hinzugefügt.

Tabelle 4.6: *Kategorie: Eigenschaften der Methodenelemente*

Unterkategorie	**möglicher Situationsfaktor**	**gültig**
Arbeitsprodukt (beliebiges Element der Methode)	Normen und Standards	☐
	Vertragseigenschaften	☐
	Dokumentation	☐
	Rechtliche Grundlagen	☐
Prozess	Anforderungen am Architektur- und Design-Prozess	☐
	Anforderungen am Implementierungsprozess	☐
	Anforderungen am Test-Prozess	☐
	Anforderungen am Systeminstallationsprozess	☐
	Anforderungen am Auslieferungsprozess	☐
	Anforderungen am Wartungsprozess	☐
	Anforderungen am Prozess zur Erstellung von Benutzerinformationen und Schulungen	☐
	Anforderungen am Prozess zur Erstellung von Benutzungshandbüchern und Hilfesystemen	☐
	Anforderungen am Prozess zur Erstellung von Schulungs- und Lernunterlagen	☐
	Anforderungen am Prozess zur Erstellung von Administrationsdokumenten	☐
Werkzeuge	Anforderungen an Werkzeuge	☐
	Reifegrad von Werkzeugen	☐
Rollen	Anforderungen an Rollen	☐
Aktivitäten	Anforderungen an Aktivitäten	☐
Meilensteine	Anforderungen an Meilensteine	☐
(Teil-)Methoden	Anforderungen an das Anforderungsmanagement	☐
	Anforderungen an das Projektmanagement	☐
	Anforderungen an das Qualitätsmanagement	☐
	Anforderungen an das Konfigurationsmanagement	☐

Anforderungen an das Änderungsmanagement	□
Anforderungen an das Risikomanagement	□
Anforderungen an das Dokumentationsmanagement	□
Anforderungen an die Analyse	□
Anforderungen an den Entwurf	□
Anforderungen an die Realisierung	□
Anforderungen an das Testen	□

4.3.6 Eigenschaften des Entwicklungsvorhabens

[KDS07] beschreibt die Dimension *development strategy*, die Charakteristiken der Art des Entwicklungsvorhabens beschreibt. Es werden die Typen Wiederverwendungstrategie, Entwicklungsstrategie, Realisierungsstrategie sowie Einführungstrategie mit den möglichen Ausprägungen dargestellt.

In [KA04] wird das Evaluierungsbeispiel, die Entwicklung eines Personalverwaltungssystems auf Basis eines bestehenden Systems, diskutiert. Auf Basis der Diskussion lassen sich einige Aspekte ermitteln, die Einfluss auf die Methode haben und abhängig vom geplanten Entwicklungsvorhaben sind. Die Analyse verschiedener Situationen ist relevant, da sie eine unterschiedliche Entwicklungsmethode hervorbringen würden. Hier passen die Erkenntnisse in den meisten Fällen zu den in [KDS07] genannten Strategien, werden aber um den Punkt *Art des Entwicklungsvorhabens* erweitert.

Die in der Dimensionsdefinition verwendeten konkreten Ausprägungen sind schwierig in die deutsche Sprache zu übersetzen und zu stark durch die festgelegten Werte eingeschränkt. Aus diesem Grund wird eine allgemeinere Beschreibung in der folgenden Liste auf Basis der Quelle [KA04] und weiteren allgemeinen Kenntnissen vorgestellt.

- Art des Entwicklungsvorhabens: Neuentwicklung, Funktionale Anpassung, Funktionale Erweiterung, Technologiewechsel, Integration verschiedener Systeme, Produktlinienentwicklung
- Wiederverwendung: Code, Funktionselemente, Schnittstellen
- Entwicklungsstrategien: Outsourcing, Entwicklung von Prototypen, Iterativ und Phasen-basiert

- Realisierungsstrategien: komplett, iterativ, nebenläufig oder überlappend
- Einführungsstrategien: Top-Down, Bottom-up, Evolutionär, Inkrementell, Parallel, Ersetzung

[BWBM08] nennt in seiner Kategorie *Geschäftsbereiche* die Entwicklungsphilosophie als Situationsfaktor mit den Ausprägungen *Agil*, *Iterativ* und *Wasserfall*. Das passt zu den Entwicklungsstrategien von [KA04]. Die Entwicklungsstrategien werden um den Situationsfaktor *Agil* erweitert. Der Situationsfaktor *Iterativ* ist bereits vorhanden und *Wasserfall* wird durch den Situationsfaktor *Phasen-basiert* bereits beschrieben.

Die Kategorie *Eigenschaften des Entwicklungsvorhaben* mit den unterschiedlichen Ausprägungen für die Kategorisierung von Situationsfaktoren erscheint relevant und wird in die Betrachtung mit aufgenommen.

Tabelle 4.7: *Kategorie: Eigenschaften des Entwicklungsvorhabens*

Unterkategorie	**möglicher Situationsfaktor**	**gültig**
Wiederverwendungsstrategie	Code	☐
	Funktionselemente	☐
	Schnittstellen	☐
Realisierungsstrategie	komplett	☐
	iterativ	☐
	nebenläufig	☐
	überlappend	☐
Art des Entwicklungs-	Neuentwicklung	☐
vorhabens	Funktionale Anpassung	☐
	Funktionale Erweiterung	☐
	Technologiewechsel	☐
	Integration verschiedener Systeme	☐
	Produktlinienentwicklung	☐
Einführungsstrategie	Top-Down	☐
	Bottom-up	☐
	Evolutionär	☐
	Inkrementell	☐
	Parallel	☐
	Ersetzung	☐
Entwicklungsstrategie	Outsourcing	☐
	Entwicklung von Prototypen	☐
	Iterativ	☐
	Phasen-basiert	☐

Agil	□

4.3.7 Eigenschaften der Kunden

[BWBM08] nennt unter Kundencharakterisiken die Situationsfaktoren Kundenloyalität, Kundenzufriedenheit, Kundenvariabilität, Kundenanzahl, Anzahl von Endbenutzern und Kundentyp. Die Kategorie Eigenschaften der Kunden wird mit den genannten Situationsfaktoren gebildet und definiert keine Unterkategorien.

Tabelle 4.8: *Kategorie: Eigenschaften der Kunden*

Unterkategorie	möglicher Situationsfaktor	gültig
	Kundenloyalität	□
	Kundenzufriedenheit	□
	Kundenvariabilität	□
	Kundenanzahl	□
	Anzahl von Endbenutzern	□
	Kundentyp	□

4.3.8 Auftraggeber- und Auftragnehmer-Beziehungen

[Bad11] thematisiert die Einflüsse durch verschiedene Auftraggeber und Auftragnehmer-Beziehungen. Eine entsprechende Kategorie wird in die Checkliste mit aufgenommen. Die Unterkategorie *Vertragsgestaltung* mit den in [Bad11] genannten Eigenschaften wird erstellt.

[BWBM08] beschreibt den Punkt der Einflussnahme des Kunden auf das Produkt, welcher ebenso von [Bad11] unter *Einflussnahme des Kunden auf die Design-Phase* beschrieben wird und unter der Kategorie *Vertragsgestaltung* auffindbar ist. Weiterhin nennt [BWBM08] die *Einflussnahme des Entwicklungspartners* als Situationsfaktor. Dafür wird die Annahme getroffen, dass es Kooperationen gibt, die nicht Gegenstand eines Produktvertrags sind. Darunter fallen beispielsweise Kooperationsprojekte. Eine entsprechende Unterkategorie wird gebildet.

Aus der Projekterfahrung bei FTS ist zu berichten, dass eine Zusammenarbeit mit Auftraggebern und Auftragnehmern stark von der geschäftlichen Beziehung abhängt. Ist diese gut, kommt es vor, dass außervertragliche Aufgaben übernommen werden, ein gemeinsames Problembewusstsein entsteht und beide Parteien gut zusammenarbeiten. Ist die Geschäftsbeziehung schlecht, sind nur Vertragsbestandteile einforderbar. Es ist zu beobachten, dass umso größer der Wirtschaftswert für ein Unternehmen in einer Auftraggeber- und Auftragnehmer-Beziehung ist, desto besser sind die Geschäftsbeziehungen. Die Unterkategorie Geschäftsbeziehungen mit den Eigenschaften Auftragnehmer, Auftraggeber wird erstellt. Die Zulieferersituation bei FTS ist im Grunde eine Auftraggeber- Auftragnehmersituation, hat jedoch eine Besonderheit. Mehrere Teilaspekte des Gesamtsystems werden vom Zulieferer erstellt. Durch diese Gegebenheit sind Abhängigkeiten des Entwicklungsprozesses mit durchzuführenden Abnahmetests sowie Abhängigkeiten bei der Systemintegration zu berücksichtigen. Aus diesem Grund wird der Situationsfaktor *Zulieferersituation* definiert.

Tabelle 4.9: *Kategorie: Auftraggeber- und Auftragnehmer-Beziehungen*

Unterkategorie	**möglicher Situationsfaktor**	**gültig**
Vertragsgestaltung	Bezahlmodalitäten	□
	Änderungswünsche	□
	Vertragsgegenstand	□
	Klarheit der Spezifikationsdokumente	□
	Technische Unsicherheiten	□
	Kostenausgleich	□
	Einflussnahme des Kunden auf die Design-Phase	□
Geschäftsbeziehung	Auftraggeber	□
	Auftragnehmer	□
Kooperationseigenschaften	Einflussnahme des Entwicklungspartners	□
	Zulieferersituation	□

4.3.9 Markteigenschaften

Insbesondere [BWBM08] beschreibt eine Reihe von allgemeineren Situationsfaktoren. Seine Marktcharakteristiken beschreiben beispielsweise den Situationsfaktor *Standard-Dominanz*, bei dem der Markt beispielsweise Standards bei den Produkten bezüglich Schnittstellen, Daten, Protokollen und weitere Produktelemente

fordert. Der Markt kann bewertet werden, indem die *Variabilität der Feature Requests* identifiziert wird. *Marktgröße* und *Marktwachstum* sind weitere Situationsfaktoren. Jedoch ist zu überprüfen, inwieweit allgemeinere Situationsfaktoren durch detailliertere Situationsfaktoren abgedeckt werden. Beispielsweise kann die Variabilität der Feature Requests durch ein geeignetes definiertes Änderungsmanagement schon berücksichtigt sein. Jedoch wird aus Gründen der Vollständigkeit dieser Situationsfaktor von [BWBM08] auch an dieser Stelle übernommen.

Ein Begriff der im Kontext dieser Arbeit aufgetreten ist, ist *Technologie getrieben*. Er umfasst die Eigenschaft, dass das Entwicklungsprojekt stark von den neuesten und vom Markt geforderten Technologien beeinflusst wird. Da der Ursprung jedoch mehr aus den Markteigenschaften hervor geht, findet er auch an dieser Stelle Berücksichtigung.

Im Gegensatz zur Technologiegetriebenheit gibt es die SchnittstellenKompatibilität. Das bedeutet, dass bestehende, sogar veraltete, Schnittstellen in neuen Systemen Berücksichtigung finden müssen, da Bestandskunden auf diese so genannten *legacy*-Schnittstellen bestehen.

Tabelle 4.10: *Kategorie: Markteigenschaften*

Unterkategorie	möglicher Situationsfaktor	gültig
	Standard-Dominanz	□
	Variabilität der Feature Requests	□
	Marktgröße	□
	Marktwachstum	□
	Technologie getrieben	□
	Schnittstellen-Kompatibilität	□

4.3.10 Produkteigenschaften

Unter *Produktcharakteristiken* beschreibt [BWBM08] die *Anzahl neuer Anforderungen*, *Produkt-Lebenszyklus*, *Produktgröße*, *Produktalter*, *Softwareplattform* und *Produkttoleranz*. Für diese Produkteigenschaften wird eine Kategorie definiert.

Ein besonderes Charakteristikum der Server-Systeme ist, dass sie hoch konfigurierbar sind. Der Aufbau sowie das Verhalten beim Endkunden sind durch eine

große Menge an Konfigurationseigenschaften gekennzeichnet. Diese Konfigurationseigenschaften sind bereits bei der Entwicklung besonders zu berücksichtigen, da sie interdisziplinär sind und dafür das Gesamtsystem betrachtet werden muss. Den Produkteigenschaften wird der Situationsfaktor *konfigurierbar* hinzugefügt.

Tabelle 4.11: *Kategorie: Produkteigenschaften*

Unterkategorie	möglicher Situationsfaktor	gültig
	Anzahl neuer Anforderungen	□
	Produktlebenszyklus	□
	Produktgröße	□
	Produkttoleranz	□
	Softwareplattform	□
	Produktalter	□
	konfigurierbar	□

4.3.11 Methodenentwicklungskontext

[Coc00] wie auch [CS05] nennen eindeutig die Interessen und Wünsche der Stakeholder des Methodenentwicklungsprojekts als Einflussfaktoren auf die Methode. Weiterhin wird dargelegt, dass durchaus die persönlichen Erfahrungen eines Methodenentwicklers Auswirkungen auf die zu erstellende Methode haben werden. Hier wird nun der Einfluss vom Methodenentwicklungskontext erkannt. Das ist der Kontext, in dem sich das Methodenentwicklungsprojekt befindet und nicht der Kontext, in dem sich die zur erstellende Methode befinden wird.

Der Methodenentwicklungskontext ist ebenso wie der Methodenkontext durch bekannte Situationsfaktoren beschreibbar. Jedoch wird nur eine Teilmenge der erarbeiteten Situationsfaktoren zu betrachten sein. Beispielsweise ist in dem Fall die Domäne *Methodenentwicklung* festgelegt, in der Arbeitsprodukte der Domäne eine Rolle spielen, aber keine Systemklassen.

Darüber hinaus sind jedoch die verfügbaren Ressourcen Geld, Zeit, Know-how und Mitarbeiter festzulegen. Diese Ressourcen haben direkten Einfluss auf das Methodenweiterentwicklungsprojekt und legen letztendlich die Möglichkeiten bei der Methodenweiterentwicklung fest. Diese Eigenschaften werden bereits in der Kategorie *Eigenschaften des Projekts* definiert.

Demnach referenziert die Kategorie *Methodenentwicklungskontext* lediglich bestehende Kategorien, die nicht nur für den Methodenkontext, sondern auch für den Methodenentwicklungskontext gelten. Die folgende Tabelle definiert die Kategorie Methodenentwicklungskontext.

Zur Vollständigkeit werden die Interessen und Wünsche der Stakeholder den *Menschlichen Eigenschaften* hinzugefügt (siehe Abschnitt 4.3.4). Diese Kategorie gilt ebenfalls für den Methodenentwicklungskontext. Auf diese Weise werden die Erkenntnisse von [Coc00] und [CS05] für den Methodenentwicklungskontext berücksichtigt.

Tabelle 4.12: *Kategorie: Methodenentwicklungskontext*

Wiederverw. Kategorien
Eigenschaften des Projekts (siehe Abschnitt 4.3.3)
Menschliche Eigenschaften (siehe Abschnitt 4.3.4)
Eigenschaften des Entwicklungsvorhabens (siehe Abschnitt 4.3.6)
In dem Fall von zwei Parteien: Auftraggeber- und Auftragnehmer-Beziehung (siehe Abschnitt 4.3.8)

4.4 Lösungsbaustein: Ermittlung Methodenkontext

Im vorigen Abschnitt haben wir mögliche Situationsfaktoren kennen gelernt. Es bleibt nun die Aufgabe, die Aktivität *Ermittlung Methodenkontext* formal zu definieren (siehe Abbildung 4.4).

Ermittlung Methodenkontext

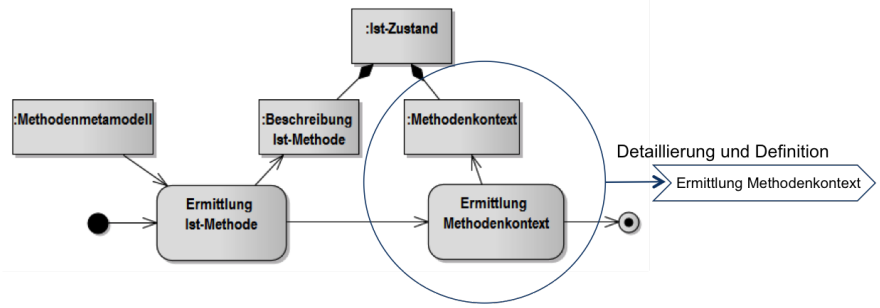


Abbildung 4.4: *Detaillierung: Ermittlung Methodenkontext*

Die formale Definition der Aktivität mit Hilfe der Sprache MMMB wird in Ab-

Erweiterung der Checkliste

schnitt 4.4.1 dargestellt. Die Aktivität definiert die Tätigkeit auf Basis der Checkliste möglicher Situationsfaktoren die gültigen zu ermitteln. In einem ersten Schritt muss die Checkliste möglicher Situationsfaktoren um domänenspezifische Arbeitsprodukte erweitert werden. Dieser Sachverhalt wird in Abschnitt 4.4.2 dargestellt. Danach wird die Aktivität exemplarisch für die Anwendungsfälle des Forschungs- und Entwicklungsprojekts durchgeführt. Einige Kategorien der Situationsfaktoren sind gültig für die gesamte Organisation, andere für das spezifische Produkt der Entwicklung. Die für FTS und für die Anwendungsbeispiele gültigen Situationsfaktoren werden in Abschnitt 4.4.3 vorgestellt.

4.4.1 Detaillierung der Aktivität *Ermittlung Methodenkontext*

Die detaillierte Aktivität *Ermittlung Methodenkontext* wird in der folgenden Abbildung 4.5 dargestellt.

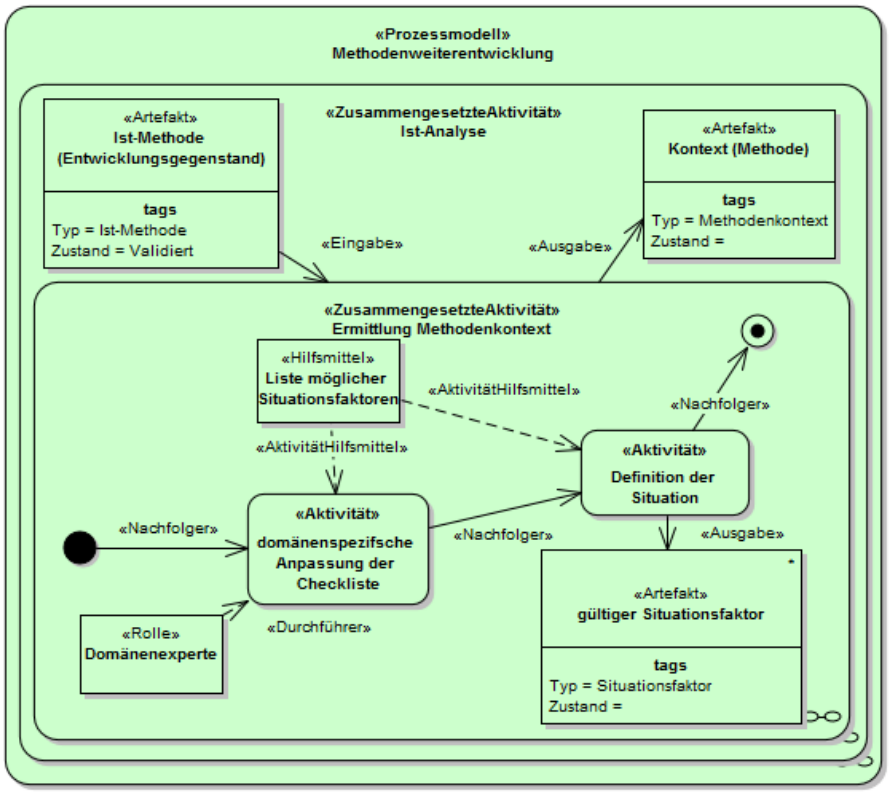


Abbildung 4.5: *Ermittlung Methodenkontext mit MetaMe++ – Prozessmodell*

Methodenkontext

Nachdem der Methodenentwickler die Ist-Methode definiert hat, muss er in einem weiteren Schritt den Methodenkontext definieren. Dazu benutzt er die Checkliste möglicher Situationsfaktoren zur Ermittlung der gültigen Situationsfaktoren, um die Situation zu charakterisieren, in der die zu entwickelnde Methode ausgeführt werden soll. Die Checkliste kennt allgemein mögliche Situationsfaktoren, aber nicht spezielle Domänenspezifika. In dem vorangegangenen Abschnitt 4.3.1 ist festgestellt worden, dass wichtige Arbeitsprodukte der Domäne von einem Domänenexperten benannt werden müssen, da diese Einfluss auf die Gestaltung einer Methode haben. Genau diese Festlegung der Arbeitsprodukte wird in Aktivität *domänenspezifische Anpassung der Checkliste* durchgeführt.

«Aktivität» domänenspezifische Anpassung der Checkliste
Eingabe: Liste möglicher Situationsfaktoren
Erweiterung der Liste durch einen Domänenexperten.
Ausgabe: Aktualisierte Liste möglicher Situationsfaktoren

Der Methodenentwickler erhält eine aktualisierte Liste möglicher Situationsfaktoren. Diese Checkliste wird zu Hilfe genommen, um gültige Situationsfaktoren zu bestimmen. Die Aktivität *Definition der Situation* erstellt demnach eine Menge[1] an gültigen Situationsfaktoren, die bei der Methodenweiterentwicklung als Artefakt *gültiger Situationsfaktor* vom Typ Situationsfaktor definiert sind (siehe Abschnitt 4.2).

«Aktivität» Definition der Situation
Eingabe: Liste möglicher Situationsfaktoren
Für jeden möglichen Situationsfaktor wird definiert, ob er im Kontext der Methodenweiterentwicklung eine Herausforderung darstellt. Neben der Gültigkeit sind entsprechende Notizen, die die Gültigkeit argumentieren, anzugeben.
Ausgabe: gültige Situationsfaktoren

Validierung

Die Menge der gültigen Situationsfaktoren definiert den Methodenkontext, der in der Abbildung 4.5 als *Kontext (Methode)* dargestellt wird und Ausgabe der Aktivität *Ermittlung Methodenkontext* ist. An dieser Stelle ist die Validierung des Methodenkontexts nicht explizit spezifiziert. MetaMe++ beschreibt an dieser Stelle keine Vorgaben. Die Validierung ist durch Durchsicht und Besprechung des Methodenkontexts zu erfolgen.

1 dargestellt als Multiplizität der Aktivität

Produktmodell

Der strukturelle Zusammenhang der Artefakte wird über die Artefakttypen definiert, die wiederum im Produktmodell definiert werden. Der für den Lösungsbaustein relevante Ausschnitt des MetaMe++ Produktmodells wird in Abbildung 4.6 dargestellt.

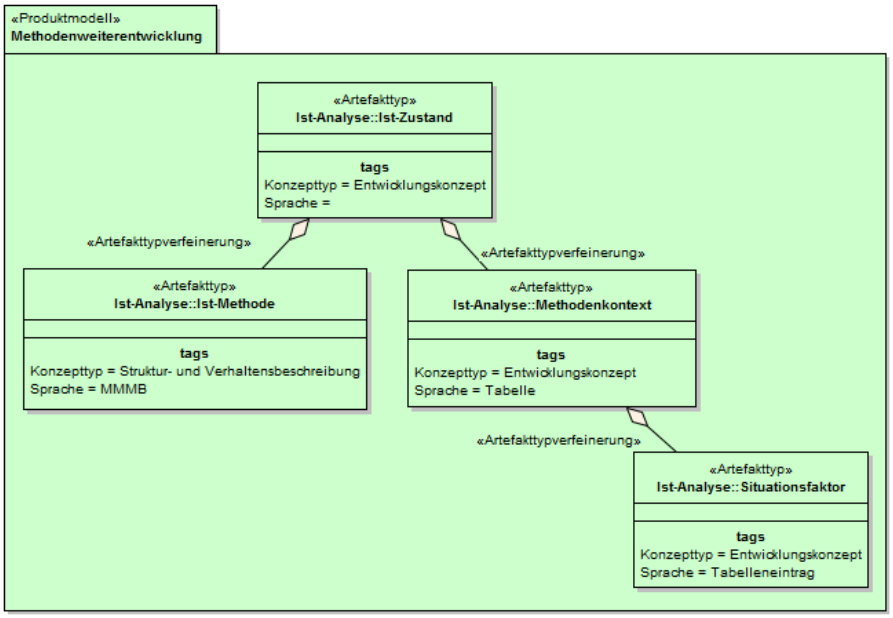


Abbildung 4.6: *Ermittlung Methodenkontext mit MetaMe++ – Produktmodell*

Es ist definiert, dass der Artefakttyp *Situationsfaktor* über den Methodenkontext aggregiert wird. Der Situationsfaktor wird beschrieben als Tabelleneintrag mit dem in Abschnitt 4.2 beschriebenen Schema. Der Methodenkontext enthält dann die vollständige Tabelle der Situationsfaktoren. Der Methodenkontext wird wiederum über den Ist-Zustand aggregiert, der zusätzlich die Ist-Methode enthält. Die Ist-Methode wird mittels der Sprache MMMB beschrieben. Die Darstellung des Ist-Zustands ist offen gelassen. Es empfiehlt sich jedoch, ihn als ein Dokument zu definieren.

4.4.2 Domänenspezifische Erweiterung der Checkliste

Checkliste für FTS

Dieser Abschnitt zeigt exemplarisch, wie die Unterkategorie der Checkliste *Wichtige Arbeitsprodukte der Domäne* für die Anwendungsbeispiele bei FTS aussieht. Die Anwendungsbeispiele beziehen sich auf die Entwicklungstätigkeiten von FTS

bezüglich der Planung von Server-Systemen mit ihren Elementen und der Softwareentwicklung bezüglich der Verwaltungssoftware von Servern und Server-Systemen. Die Hardware-Entwicklung wird nur am Rande berücksichtigt, da in den Forschungs- und Entwicklungsprojekten nur wenig Berührungspunkte vorhanden waren.

Arbeitsprodukte

In dem Abschnitt 3.2 sind die einzelnen Elemente von Server-Systemen mit ihren strukturellen Beziehungen beschrieben worden. Genau aus diesem Domänenwissen müssen die relevanten Arbeitsprodukte für die Planung und Entwicklung von Server-Systemen ermittelt werden. Der Abschnitt 3.4 beschreibt die strukturellen Eigenschaften der Verwaltungssoftware, aus denen die Arbeitsprodukte der Softwareentwicklung abgeleitet werden können.

Begriffsmodelle

Weiterhin sind in der Entwicklung Begriffsmodelle wichtig. Der Abschnitt 3.2 beschreibt die allgemeinen Begriffsmodelle für die verschiedenen Arten der Server-Systeme. Bei der Entwicklung eines spezifischen Produkts werden für das jeweilige Produkt eigene Begriffe gebildet. Die Abbildung 4.7 stellt beispielsweise die allgemeinen Begriffe der Entwicklung eines modularen Server-Systems den Begriffen des konkreten Blade-Server-Systems BX900 nebeneinander. Die Begriffsdefinition, am besten mit Abbildung auf die allgemeinen Begriffe, ist für das gemeinsame Verständnis im Entwicklungsprojekt sinnvoll.

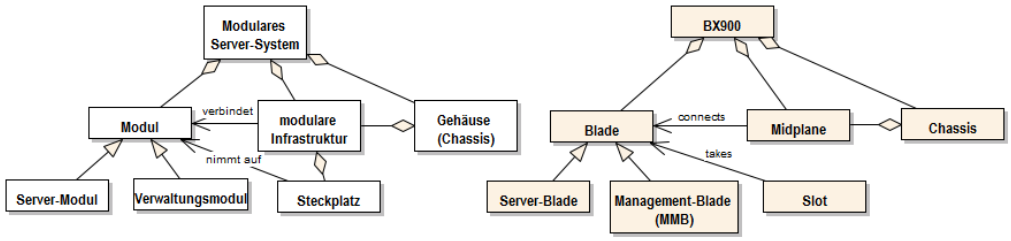


Abbildung 4.7: *Gegenüberstellung allgemeiner und projektspezifischer Begriffe bei der Server-System-Entwicklung*

Die folgende Tabelle stellt die Zusammenfassung der wichtigen Arbeitsprodukte im Sinne der Checkliste möglicher Situationsfaktoren dar.

Tabelle 4.13: *Unterkategorie: Wichtige Arbeitsprodukte der Domäne*

möglicher Situationsfaktor	gültig
Beschreibung Systemarchitektur	☐
Beschreibung der Konfigurationseigenschaften	☐

Beschreibung der Architektur von System-Elementen	□
Beschreibung der System-Infrastruktur	□
Beschreibung der Schnittstellen	□
Beschreibung der Software(-Verwaltungsarchitektur)	□
Beschreibung der Systemkonzepte	□
Beschreibung des Datenmodells der Systeminformation	□
Beschreibung von Protokollen	□
Begriffsmodelle	□

4.4.3 Gültige Situationsfaktoren für FTS

Zur Ermittlung des Methodenkontexts bei FTS ist es notwendig die allgemeinen Herausforderungen zu erkennen. In den folgenden Abschnitten werden die Herausforderungen vorgestellt, um anschließend die daraus resultierende Charakterisierung mit gültigen Situationsfaktoren zu definieren. Die vorgeschriebenen Notizen werden nicht dargestellt, da die notwendige Beschreibung der Herausforderung zu Beginn des jeweiligen Abschnitts dargestellt wird.

Eigenschaften der Domäne

allgemeine Herausforderungen

Der Abschnitt 3.1 nennt bereits Herausforderungen bei der Server-System-Entwicklung, wie *Fehlende Festlegung des Domänenwissens*, *Darstellung der Systemarchitekturen* und *Ermittlung von Systemkonzepten*, *Fehlende Software-Entwicklungskonzepte* und *Fehlende Sprachen*.

technologische Herausforderungen

Server-Systeme sind komplexe eingebettete Systeme. Die Entwicklung eines solchen Systems ist ein interdisziplinäres Vorhaben mit Entwicklungsaspekten aus der Mechanik, Thermodynamik, Elektrotechnik und Softwareentwicklung. Hierbei entstehen Abhängigkeiten zwischen den Disziplinen, die unter anderem zu konträren Anforderungen im Entwicklungsprozess führen.

- Ein Server-System ist ein komplexes mechanisches System. Als Beispiel dient ein beliebiges modulares Server-System. Die einzelnen Server müssen beliebig in ein Chassis gesteckt werden können. Die Entwicklung eines Chassis stellt Anforderungen an die einzelnen Systemmodule, die wieder-

um durch bestimmte Charakteristiken Anforderungen an das Chassis stellen. Insbesondere sind hier mechanische Maße und deren Toleranzen sowie die Anordnung von mechanischen Steckern zu berücksichtigen. Es spielen nicht nur äußere Eigenschaften wie Maße und Positionierungen eine Rolle, sondern interne Eigenschaften wie Luftwiderstände, Leistungsaufnahme und Abfuhr produzierter Wärme müssen bei der mechanischen Planung des Gesamtsystems einbezogen werden. Die Existenz bestimmter mechanischer Elemente mit ihren Maßen und Positionierungen im Gesamtsystem stellt thermodynamische und elektrotechnische Anforderungen an das Gesamtsystem.

- Thermodynamische Eigenschaften betreffen hauptsächlich die Kühlungsstrategie des Systems, der Systemmodule und der Modulkomponenten. Unter einer aktiven Kühlung wird die Kühlung durch ein Kühlmodul oder Kühlkomponente eines Submoduls oder Subkomponente angesehen. Reichen die Kühleigenschaften der Betriebsumgebung zur Abfuhr der produzierten Wärme, handelt es sich um eine passive Kühlung. Hat ein Server eigene Lüfter zur Kühlung, wird der Server aktiv gekühlt. Gibt es Lüfter im Modularen-Server-System, die auch die Kühlung der Server übernehmen, welche dann keine eigenen Lüfter mehr benötigen, wird das Server-System aktiv gekühlt, die einzelnen Server jedoch passiv. Bei der Entwicklung des Kühlungskonzepts eines Server-Systems ist die entstehende abzuführende Wärme des Gesamtsystems und aller Systemkomponenten zu betrachten. Im Vergleich dazu muss das Potenzial der möglichen Wärmeabfuhr betrachtet werden. Einflussgrößen sind Leistungsaufnahme und mechanische Größen wie Maße und Positionierungen. Das Kühlkonzept stellt mechanische und elektrotechnische Anforderungen an das Gesamtsystem.
- Bei den elektrotechnischen Eigenschaften steht die Verbindung aller Systemmodule und Systemkomponenten untereinander sowie die Stromversorgung im Vordergrund. Einfache Signalleitungen bis hin zu spezialisierten High-Speed-Verbindungen sind zu berücksichtigen. Der Sammelbegriff für alle elektrischen Verbindungen innerhalb eines modularen Server-Systems wird in dieser Arbeit als (System-)Infrastruktur bezeichnet. Die Infrastruktur stellt mechanische Stecker und deren Verbindungen untereinander bereit. Jedes Systemmodul wird mit der Infrastruktur verbunden. Bei aktuellen Systemen, beispielsweise eines BX900[2], umfasst die Infrastruktur ca.

2 Bei Blade-Server-Systemen wird die Infrastruktur auch Midplane genannt. Eine Midplane hat jedoch die Eigenschaft als Platine (Leiterplatte) konstruiert zu werden und dadurch wird ein, nicht immer kostengünstiger, minimaler Formfaktor möglich.

10000 Signalverbindungen. Die Entwicklung der Infrastruktur stellt Anforderungen an die Positionierung von Steckerverbindungen. Die Stromversorgung wird durch die bereit zu stellenden Ressourcen (Speicher, Rechenleistung und durchsatzstarke Infrastruktur) sowie den Systemmodulen und -komponenten und deren Leistungsaufnahme definiert. Größer und schneller sind in diesem Zusammenhang Adjektive mit Bezug auf eine erhöhte Leistungsaufnahme. Umso höher die Leistungsaufnahme ist, umso mehr abzuführende Wärme wird produziert. Die Entwicklung der Elektrotechnik stellt thermodynamische Anforderungen an das Gesamtsystem.

Selbst-Adaptivität

Server-Systeme sind selbst-adaptive Systeme. Selbst-adaptive Mechanismen dienen dazu, den manuellen Verwaltungsaufwand zu reduzieren, indem Fehlerzustände im laufenden System erkannt werden und das System versucht selbstständig darauf zu reagieren, um einen sicheren Systemzustand zu erreichen, ohne dass ein Systemadministrator manuell eingreifen muss. Diese selbst-adaptiven Mechanismen sind komplex. Als Beispiel nehmen wird das Kühlkonzept. Eine Menge verschiedener Temperatursensoren sind notwendig, um die Systemtemperatur ganzheitlich überwachen zu können. Weiterhin muss eine Menge an Aktoren, beispielsweise die Drehzahlsteuerung der Systemlüfter, vorhanden sein. Werte der Sensoren werden verarbeitet, um die Aktoren zu steuern. Sensoren und Aktoren sind über die verschiedenen Systemmodule verteilt und demnach ist eine Vielzahl von Schnittstellen zu berücksichtigen. Andere Systemkonzepte können die Übersicht bei der Entwicklung der selbst-adaptiven Mechanismen behindern. Beispielsweise sind Performanz-Einstellungen im Server-System möglich, die die Rahmenbedingung für die Lüftersteuerung beeinflussen. Schutzmechanismen sind nicht nur in der Regelsoftware, sondern auch hardwaremäßig umgesetzt, so dass es dort zu unerwünschten Wechselwirkungen kommen kann.

Unterschiedliche Datenbasis

Im Abschnitt 3.3.3 wird die Verwaltungsarchitektur CIM und im Anhang A.2 SNMP und IPMI vorgestellt. Dort sind ebenfalls ihre unterschiedlichen Architekturen dargestellt. Jeder Standard basiert auf einer unterschiedlichen Datendefinition. FRUs, SDRs und MOF-Dateien definieren System-Daten und -Funktionen für die jeweilige Verwaltungsarchitektur in ihren eigenen Formaten mit eigenen Datentypen. Der Zugriff auf diese Systeminformationen unterliegt Standardspezifischen Zugriffsmethoden. Es ergeben sich teilweise redundante Datendefinitionen, wenn eine bestimmte Systeminformation über verschiedene Verwaltungsschnittstellen, auf Basis unterschiedlicher Verwaltungsarchitekturen, bereitgestellt wird. Aufgrund der unterschiedlichen Datendefinitionen ist es teilweise

nicht möglich eine vollständige Abbildung der Daten zwischen den verschiedenen Standards zu definieren. Als Beispiel dient das Spezifikationsdokument [Int06], welches versucht, IPMI-basierte Systeminformationen auf dem CIM-Standard abzubilden. An vielen Stellen ist eine Abbildung der Systeminformationen nicht möglich. Das liegt grundlegend daran, dass bei der Entwicklung der einen oder anderen Verwaltungsarchitektur unterschiedliche Interpretationen der von Hardware bereitgestellten Bits und Bytes stattfinden. Ein grundlegendes Datenmodell der Systeminformationen, auf dessen Basis die übergeordneten Verwaltungsarchitekturen agieren, ist hilfreich. Es hilft bei der Vermeidung über verschiedene Verwaltungsschnittstellen inkonsistente Systeminformationen bereitzustellen.

Komplexität

Der Abschnitt 3.3.1 beschreibt die vielfältige Verwaltungsfunktionalität, die in ihrer Gesamtheit eine hohe Komplexität mit sich bringt und deshalb unübersichtlich ist. Hier besteht die Aufgabe darin die Verwaltungsfunktionalität überschaubar zu machen. Die Lösung besteht in der richtigen Vereinfachung. Verschiedene Softwareentwicklungskonzepte bieten Lösungen an, deren Anwendung sich in der Praxis bei Fujitsu Technology Solutions GmbHund der Vielzahl an Zulieferern nicht beobachten lässt. Die dieser Arbeit zugrunde liegenden Ideen werden in der folgenden Aufzählung vorgestellt und sind als mögliche Anforderungen an die Server-System-Entwicklung zu verstehen

- Abstraktion: Das Ausblenden technischer Eigenschaften der spezifischen Eigenschaften der Verwaltungsarchitekturen (siehe Abschnitt 3.3.3).
- Komponentenbasierte Softwareentwicklung: Das Konzept der komponentenbasierten Softwareentwicklung definiert Komponenten und deren Schnittstellen, welche durch Kontrakte neben strukturellen auch semantische Eigenschaften der Schnittstellen beschreiben. Der physikalische Aufbau eines Server-Systems beschreibt die Unterteilung in Systemmodule, die wiederum aus einer Menge von Komponenten besteht, die über physikalische Stecker und Verbindungen miteinander integriert werden ähnlich einer komponentenbasierten Softwarearchitektur.
- Separation-of-Concerns: Es fehlt eine Klassifikation der Verwaltungsfunktionalitäten. In der WBEM-Architektur (siehe Abbildung 3.9) erkennen wir die unterschiedlichen Zugriffsebenen, Datenmodellzugriff auf den CIM-Server vom CIM-Client und den Hardwarezugriff vom CIM-Provider auf Managed Objects. Der Abschnitt 3.3.3 beschreibt die technischen Eigenschaften der Verwaltungsarchitekturen mit ihren spezifischen Funktionsaufrufen. Weiterhin bestehen die Server-Systeme aus immer größer werdenden Softwarean-

teilen, die unterschiedliche Aufgaben erfüllen. Hierbei muss die Trennung unterschiedlicher Aspekte, aufgrund möglicher Abhängigkeiten, ersichtlich sein. Die Aufgabe besteht darin die unterschiedlichen, unabhängigen Kategorien der Verwaltungsfunktionalität zu erkennen und eine getrennte Entwicklung dieser zu ermöglichen.

- Modellgetriebene Softwareentwicklung: Sind die genannten Vorteile der Abstraktion, komponentenbasierten Entwicklung und Separation-of-Concerns für die Domäne der Server-System-Entwicklung definiert können entsprechende Modelle zur Abbildung der Informationen entwickelt werden.

Abhängigkeit zur Infrastruktur

Das Schreiben und Lesen von Konfigurationsdaten sowie die Ausführung von Verwaltungsfunktionen über eine Verwaltungsschnittstelle stellt Bedingungen an die Infrastruktur. Soll beispielsweise eine Lüfterdrehzahl eines aktiv gekühlten Servers korrigiert werden, muss ein BMC, über welchen der administrative Zugriff geschieht, eine physikalische Verbindung zum Lüfter des Servers haben.

Konfigurationen

Die Festlegung der möglichen Konfigurationen eines modularen Server-Systems ist eine interdisziplinäre Herausforderung. Die mögliche Konfiguration muss mechanisch ermöglicht werden. Ein Steckplatz für ein Modul muss verschiedene Module aufnehmen können oder die verschiedenen Module müssen in mechanischer Hinsicht (Maße, Stecker) kompatibel sein. Thermodynamische Eigenschaften (aktive, passive Kühlung, Luftdurchsatz) müssen unabhängig von verschiedenen Konfigurationen eingehalten werden. Notwendige elektrische Verbindungen der Infrastruktur müssen vorhanden sein. Unterschiedliche Systemkonfigurationen müssen in der Verwaltungssoftware abgebildet werden. Ein einfaches Konzept wäre die Auslegung der Infrastruktur innerhalb des modularen Server-Systems auf maximale Kompatibilität. Das würde bei einem System mit 18 Server-Steckplätzen bedeuten, dass alle 18 Steckplätze die maximale Konnektivität, abhängig von den angebotenen Schnittstellen des Servers mit der größten Anzahl von Schnittstellen, unterstützen müssen. Die Bereitstellung der maximalen Konnektivität für alle möglichen Steckplätze ist aus Gründen der hohen Kosten selten möglich. Die Aufgaben der unterschiedlichen Systemmodule (Versorgungsmodule, Verwaltungsmodule oder Server) sind so unterschiedlich, dass universale Steckplätze, Stecker, Modulchassis gar nicht umsetzbar wären. Es werden unterschiedliche Steckplätze mit unterschiedlicher Konnektivität bereitgestellt. Geringere Hardwarekosten werden eingehalten, jedoch wird die Komplexität in der Entwicklung gesteigert. Die Konfigurationsmöglichkeiten werden eingeschränkt, indem nicht alle unterstützten Systemmodule in allen mechanischen Slots voll un-

terstützt werden, da elektrische Verbindungen fehlen. Diese strukturellen Informationen müssen wiederum in der Verwaltungssoftware abgebildet werden. Die Entwicklung benötigt für die Definition der Server-System-Architektur, welche die Konfigurationsmöglichkeiten berücksichtigt, ein Konfigurationsmodell.

Die folgende Liste gültiger Situationsfaktoren der Kategorie *Eigenschaften der Domäne* wird erstellt.

Tabelle 4.14: *Kategorie: Eigenschaften der Domäne – gültig für die Anwendungsbeispiele*

Unterkategorie	**möglicher Situationsfaktor**	**gültig**
Systemklasse	Eingebettetes System	☒
	Selbst-Adaptives-System	☒
Qualitätseigenschaften	Formalität	☒
	Beziehungen	☒
	Abhängigkeiten	☒
	Komplexität	☒
Wichtige Arbeitsprodukte der Domäne	Beschreibung Systemarchitektur	☒
	Beschreibung der Konfigurationseigenschaften	☒
	Beschreibung der Architektur von System-Elementen	☒
	Beschreibung der System-Infrastruktur	☒
	Beschreibung der Schnittstellen	☒
	Beschreibung der Software(-Verwaltungsarchitektur)	☒
	Beschreibung der Systemkonzepte	☒
	Beschreibung des Datenmodells der Systeminformation	☒
	Beschreibung von Protokollen	☒
	Begriffsmodelle	☒
Techniken und Paradigmen	modellbasierte Entwicklung	☒
	komponentenbasierte Entwicklung	☒
	Abstraktion	☒
	Separation-of-Concerns	☒
Hardware-Entwicklung	Mechanik	☒
	Elektrotechnik	☒
	Thermodynamik	☒

Eigenschaften des Projekts

verteilte Entwicklung

Die Hardware- und Softwareentwicklung ist aufwendig. Die Entwicklungsaktivitäten werden verteilt und dadurch entstehen die verschiedensten Zuliefererthematiken – von der Hardwarezulieferung über Softwarezulieferung bis hin zur kombinierten Hardware-Softwarezulieferung über mehrere am Entwicklungsprozess beteiligte Entwicklungs- und Produktionsstätten. Am Beispiel der Fujitsu Technology Solutions GmbHmit verschiedenen Standorten innerhalb Deutschlands und weltweit ansässigen Zulieferern erkennen wir die typische geografisch und interkontinental verteilte Entwicklung. Der Wissenstransfer und die Verständigung über die Charakteristiken der Aufgaben im Projekt ist in abgesicherter Form nur über gute Dokumentations- und Spezifikationsdokumente zu gewährleisten. Die Etablierung eines Pflichtenhefts als Vertragsgrundlage dient hier nur als Beispiel. An allen Schnittstellen der interdisziplinären Entwicklung, bei denen Informationen von einzelnen Abteilungen oder Rollen ausgetauscht werden, müssten Dokumentrichtlinien bezüglich Inhalt und Sprache definiert sein. Genau an dieser Stelle sind domänenspezifische, standardisierte, formale Sprachen die Grundlage des eindeutigen und verständlichen Informationsaustauschs. Dieser Sachverhalt trifft für FTS nicht zu (siehe auch Abschnitt 3.5 und 3.1).

Tabelle 4.15: *Kategorie: Eigenschaften des Projekts – gültig für die Anwendungsbeispiele*

Unterkategorie	möglicher Situationsfaktor	gültig
	Anzahl der Stakeholder	☒
	verteilte Entwicklung	☒

Menschliche Eigenschaften

viele Akteure im Entwicklungsprozess

Wir kennen bereits die interdisziplinären Disziplinen der Server-System-Entwicklung. Dadurch kommen unterschiedliche Personen mit den unterschiedlichsten Rollen in jeder Disziplin vor. Neben den Entwicklern sind Produktmanager sowie Qualitätssicherer nur einige Beispiele. An dieser Stelle treffen unterschiedliche Ziele und Erwartungen der verschiedenen Personengruppen aufeinander. Hier besteht die Schwierigkeit darin Begriffswelten unterschiedlicher Disziplinen und Abstraktionsebenen interdisziplinär zu vereinen.

Tabelle 4.16: *Kategorie: Menschliche Eigenschaften – gültig für die Anwendungsbeispiele*

Unterkategorie	möglicher Situationsfaktor	gültig
Eigenschaften der Stakeholder	Klarheit	☒
	Wünsche	☒

Eigenschaften der Methodenelemente

Anforderungserhebung

Nach der Anforderungserhebung ist davon auszugehen, dass die Implementierung eines Server-Systems in Hardware- und Softwareentwicklung aufgeteilt wird. Wie wird jedoch vorher ein Kompromiss bei der Anforderungserhebung getroffen? Dazu ist es notwendig die Abhängigkeiten der verschiedenen Disziplinen bereits bei der Anforderungsdefinition zu kennen und zu berücksichtigen. Hier sind zum einen die Beziehungen zwischen Funktionalitäten der Systemelemente, der Infrastruktur und der Konfigurationsmöglichkeiten abzubilden und zum anderen die eventuell konkurrierenden Anforderungen der Mechanik, Thermodynamik und Elektrotechnik (siehe *Eigenschaften der Domäne*). Bei FTS gibt es keine übergeordnete Anforderungserhebung.

Beispiel CIM-Entwicklung

Beim Anwendungsfall der CIM-Entwicklung ist es der Fall, dass FTS Neuland betritt. Das heißt, es besteht keinerlei Vorstellung über die notwendigen Entwicklungstätigkeiten. Insbesondere soll eine Übersicht über die Anforderungen etabliert werden, der Entwicklungsprozess soll effizient sein, da nur eingeschränkte Ressourcen zur Verfügung stehen, und, da FTS Wert auf Qualität legt, soll ein Testkonzept definiert werden. Aufgrund der wenigen Informationen über die CIM-Entwicklung werden diese schlecht einschätzbaren Herausforderungen auf die Situationsfaktoren der Teilmethoden Analyse, Entwurf, Realisierung und Testen abgebildet. Weiterhin definiert der CIM-Standard eine standardisierte Form der Spezifikationsdokumente. Aus diesem Grund werden die genannten Aspekte als Herausforderung angesehen. Da die Entwicklung einer Spezifikationsmethode für die CIM-Entwicklung als Projektgegenstand des Forschungs- und Entwicklungsprojekts definiert ist, erwartet FTS eine Dokumentation der Spezifikationsmethode.

Meilensteine

Wie in Abschnitt 3.6.2 bereits erwähnt legt FTS besonderen Wert auf definierte Meilensteine ihrer Methodenhandbücher, da diese von der Management-Ebene sichtbar sind und darüber der Projektfortschritt bewertet wird.

Tabelle 4.17: *Kategorie: Eigenschaften der Methodenelemente – gültig für die Anwendungsbeispiele*

Unterkategorie	möglicher Situationsfaktor	gültig
Arbeitsprodukte	Normen und Standards	☒
	Dokumentation	☒
Meilensteine	Anforderungen an Meilensteine	☒
(Teil-)Methoden	Anforderungen am Dokumentationsmanagement	☒
	Anforderungen an die Analyse	☒
	Anforderungen an den Entwurf	☒
	Anforderungen an die Realisierung	☒
	Anforderungen an das Testen	☒

Eigenschaften des Entwicklungsvorhabens

Wiederverwendung von Systemkonzepten

Ein zu erreichendes Ziel bei der Entwicklung der Verwaltungssoftware ist die möglichst hohe Wiederverwendung. Beispielsweise sollen generische Systemkonzepte, wie beispielsweise eine adaptive Lüftersteuerung, auf verschiedenen Systemen mit unterschiedlichen Verwaltungsarchitekturen wiederverwendet werden können. Hierbei muss der Entwickler den Unterschied zwischen generell zu erwartenden Daten und Daten, die spezifisch für einzelne Systeme oder der Verwaltungsarchitektur sind, kennen. IPMI definiert die generell zu erwartenden Daten in SDRs, SNMP in der MIB und CIM im CIM-Schema. Es ist jedoch nicht gesagt, dass ein Standard für ein spezifisches System generell alle diese Daten bereitstellt. Das bedeutet, dass eine IPMI-basierte adaptive Lüftersteuerung nicht zwangsläufig auf verschiedenen Systemen funktionsbereit ist. Zur frühzeitigen Evaluierung sind Spezifikationen zur systemspezifischen Überdeckung der generellen Daten nötig. Des Weiteren ist eine Übertragung einer IPMI-basierten adaptiven Lüftersteuerung, aufgrund der technischen Eigenschaften, nicht ohne erheblichen Aufwand auf andere Verwaltungsarchitekturen zu übertragen. Eine, wie im vorigen Absatz vorgeschlagene, grundlegende Datenmodellierung ermöglicht die Spezifikation von Systemkonzepten auf technologisch unabhängiger Ebene. Eine Anpassung eines Systemkonzepts an eine Technologie ist einfacher als die Überführung eines Systemkonzepts von einer Technologie in eine andere.

Tabelle 4.18: *Kategorie: Eigenschaften des Entwicklungsvorhabens – gültig für FTS*

Unterkategorie	möglicher Situationsfaktor	gültig
Wiederverwendungsstrategie	Code	☒
	Funktionselemente	☒
	Schnittstellen	☒

Markteigenschaften

Vereinfachte Verwaltung

Der Bereich der Server-System-Entwicklung ist stark geprägt von den Anforderungen des Markts. Anforderungen bezüglich der Benutzerfreundlichkeit stehen deshalb stets im Vordergrund. Unter den Schlagwörtern *Single-Point-of-Administration* und *Reduzierung des Verwaltungsaufwands* ist dieses Thema bereits im Abschnitt 3.1 beschrieben worden.

Schnittstellenkompatibilität

Aufgrund fehlender Standards in der Vergangenheit sind in verschiedenen Rechenzentren unterschiedliche proprietäre Infrastrukturen zur Verwaltung der Server-Systeme aufgebaut worden. Sollen nun neue Server-Systeme in einer solchen proprietären Verwaltungsinfrastruktur eingesetzt werden, müssen sie eine Vielzahl, teilweise veralteter, Verwaltungsschnittstellen anbieten, um Integration zu gewährleisten.

technologiegetrieben

Die Definition und Neuentwicklung verbesserter Verwaltungsschnittstellen geht weiter voran. Unter dem Begriff *technologiegetrieben* stehen sämtliche Marktanforderungen an die neuesten Verwaltungsschnittstellen.

steigender Entwicklungsaufwand

Letztendlich sieht sich die Server-System-Entwicklung mit der Tatsache konfrontiert, eine hohe Anzahl verschiedener standardisierter Verwaltungsschnittstellen anzubieten, zum einen die vielen etablierten und zum anderen die immer neueren. Aus diesem Grund steigt der Implementierungsaufwand stetig an.

Tabelle 4.19: *Kategorie: Markteigenschaften – gültig für die Anwendungsbeispiele*

Unterkategorie	möglicher Situationsfaktor	gültig
	Standard Dominanz	☒
	technologiegetrieben	☒
	Schnittstellenkompatibilität	☒

Produkteigenschaften

Verlagerung der Systemintegration

Die Aufgabe der Systemintegration verlagert sich vom Betreiber auf die Entwickler der Server-Systeme. Während ein Betreiber aus den vielen Möglichkeiten des Zusammenschlusses der Server-System-Elemente eine, für ihn brauchbare, auswählt, müssen die Entwickler der heutigen modularen Server-Systeme auf verschiedene Kunden Rücksicht nehmen und somit mehrere Konfigurationsmöglichkeiten unterstützen.

Tabelle 4.20: *Kategorie: Produkteigenschaften – gültig für die Anwendungsbeispiele*

Unterkategorie	möglicher Situationsfaktor	gültig
	konfigurierbar	☒

Weitere Kategorien

Es gibt noch die Kategorien Eigenschaften der Kunden und Auftraggeber- und Auftragnehmer-Beziehungen. Für diese Kategorien konnten im Forschungs- und Entwicklungsprojekt zu wenig verlässliche Informationen gesammelt werden, die auf eine besondere Herausforderung hinweisen. Darum sind an dieser Stelle keine gültigen Situationsfaktoren definierbar. Die Kategorie Methodenentwicklungsprojekt wird erst in der Phase *Verbesserungs-Analyse* relevant und wird im Kapitel 5 thematisiert.

4.5 Zusammenfassung

Dieses Kapitel hat sich mit der Ermittlung des Methodenkontexts befasst. Ziel war es Eigenschaften zu definieren, die beschreiben, in welchem Kontext die Ist-Methode agiert. Der Methodenkontext wird über Situationsfaktoren beschrieben. Eine große Menge heterogener Situationsfaktoren wird in der Literatur über SME genannt (siehe Abschnitt 4.1). Ein Situationsfaktor beschreibt eine Eigenschaft, die Einfluss auf die Entwicklung einer Methode hat. Das heißt, eine Methode muss an einen Situationsfaktor angepasst werden, damit sie passend zu der vom Situationsfaktor beschriebenen Eigenschaft ist. Der Abschnitt 4.2 beschreibt die Schwierigkeit der Bewertung der Situationsfaktoren. Zum einen ist es schwierig,

jedem Situationsfaktor einem Wertebereich zuzuordnen und zum anderen ist es schwierig, einen Wert zu interpretieren und Rückschlüsse auf Maßnahmen bei der Methodenweiterentwicklung zu ziehen. Das funktioniert momentan nur für einige Eigenschaften. Aus diesem Grund macht es keinen Sinn eine vollständige Charakterisierung des Methodenkontexts über mit Werten behaftete Situationsfaktoren im Sinne eines Ist-Zustands zu definieren, denn wir können keine Bewertung gegenüber der Ist-Methode erstellen. Aus diesem Grund ist das Ergebnis in dieser Arbeit, dass Situationsfaktoren genutzt werden, um Herausforderungen im Kontext der Methode zu identifizieren. Es wird das Wissen erstellt, welche Eigenschaften von der Ist-Methode nicht ausreichend berücksichtigt werden und zu Schwierigkeiten bei der Anwendung der Methode führen. Genau diese Eigenschaften müssen bei der Methodenweiterentwicklung berücksichtigt werden. Für eine vollständige Ermittlung der Herausforderungen ist es wichtig eine umfassende Liste möglicher Situationsfaktoren zu haben. Diese wird in Abschnitt 4.3 dargestellt. Der methodische Lösungsbaustein der Ermittlung gültiger Situationsfaktoren auf Basis der Checkliste möglicher Situationsfaktoren wird in Abschnitt 4.4 mittels MMMB als Baustein für MetaMe++ definiert. Er stellt einen Baustein der Metamethode zur Entwicklung der spezifischen Entwicklungsmethode dar. Die Anwendung des Lösungsbausteins auf die Anwendungsbeispiele dieser Arbeit wird in den Abschnitten 4.4.2 und 4.4.3 exemplarisch durchgeführt und führt zur Beschreibung der Herausforderungen bei der Ist-Methode, also der Methodenkontext, der bei der Methodenweiterentwicklung konkret zu beachten ist.

5 Methodenanforderungen

Definition Soll-Methode

Der Methodenentwickler überführt bei der Methodenweiterentwicklung eine Ist-Methode in eine weiterentwickelte Soll-Methode. In den vorangegangenen Kapiteln ist die Ist-Analyse thematisiert worden. Dabei wird ein Ist-Zustand ermittelt, der aus der modellbasierten Definition der Ist-Methode und einem Methodenkontext, der gültige Situationsfaktoren beschreibt, besteht. Die gültigen Situationsfaktoren definieren Rahmenbedingungen, die Herausforderungen bei der Ausführung der Ist-Methode darstellen. Die Bewältigung der Herausforderungen der Ist-Methode stellt das primäre Ziel der Methodenweiterentwicklung dar. Die ermittelten Rahmenbedingungen haben einen Einfluss auf die Definition der Soll-Methode, denn die Soll-Methode soll so definiert werden, dass sie die Rahmenbedingungen optimal berücksichtigt und auf diese Weise die Herausforderungen eliminiert.

Ziele der Methodenweiterentwicklung

Nachdem die Phase *Ist-Analyse* der Methodenweiterentwicklung mit der Definition des Ist-Zustands abgeschlossen ist, sollen in der Phase *Verbesserungs-Analyse* die Optimierungspotenziale der Ist-Methode ermittelt werden. Auf Basis der Optimierungspotenziale soll ein Konzept für eine Soll-Methode erstellt werden, die erstens bestehende Herausforderungen behebt, also an den Methodenkontext angepasst ist, und zweitens weitere Ziele des Auftraggebers an die Methodenweiterentwicklung berücksichtigt. Die Frage nach der Dokumentation und Nachvollziehbarkeit bei der Realisierung dieser Ziele besteht.

Anforderungen

Bei der Methodenentwicklung wird an dieser Stelle auf die Erhebung von Methodenanforderungen verwiesen (siehe Abschnitt 1.4). Die Anforderungen können die gesetzten Ziele verfeinern und dokumentieren. Das Problem ist hier das umfassende Themengebiet der Anforderungsentwicklung (Requirements Engineering (RE)) mit den unterschiedlichen Aspekten Erhebung, Dokumentation und Verwaltung.

fehlende Ansätze

Die Autoren beschreiben in ihren Artikeln [CATP10], [CATP11] und [CATP12] die Evolution eines modellbasierten Frameworks zur der Definition von Methoden

und Generierung von Methodensoftware. Das definierte Vorgehen beschreibt die drei Phasen Methoden-Design, Methoden-Konfiguration und Methoden-Implementierung. Beim Methodendesign werden lediglich Aussagen über vorhandene Methodenanforderungen getätigt, nicht aber auf deren Erhebung eingegangen. Verschiedene allgemeine RE-Ansätze, beispielsweise nach [RR99], [Rup09] oder [Wei08], beziehen sich auf die Software- und System-Entwicklung, aber nicht auf die Methodenweiterentwicklung. Lediglich ein Ansatz von Ralyté (vgl. [Ral02]) befasst sich speziell mit der Erhebung von Anforderungen an Methoden. Dieser Ansatz richtet sich aber speziell an Prozessbestandteile der Methode und berücksichtigt nicht die weitergehenden Methodenbestandteile oder gar die gültigen Situationsfaktoren bei der Methodenweiterentwicklung.

Methoden-anforderungen

Können Methodenanforderungen auf Basis des Methodenkontexts der Ist-Methode und weiterer Ziele definiert werden, stellen sie eine Dokumentation der vorliegenden Ziele der Methodenweiterentwicklung dar. Weiterhin ist es möglich Methodenanforderungen beliebig zu verfeinern. Die Gesamtmenge der möglichen Methodenanforderungen stellt die Anforderungen an eine Ideal-Methode dar. Das Methodenweiterentwicklungsprojekt unterliegt jedoch dem Methodenentwicklungskontext, der die Rahmenbedingungen für die Methodenweiterentwicklung in Form von Kosten, Ressourcen und weiteren einschränkenden Bedingungen vorgibt. Der Methodenentwicklungskontext lässt in der Regel nicht zu, dass alle Methodenanforderungen realisiert werden können.

Iterative Methoden-Entwicklung

Allgemeine Konzepte der Anforderungsermittlung (Traceability, Priorisierung und Anforderungsanalysen) können auf die Methodenanforderungen angewendet werden. Auf diese Weise ist es möglich die Gesamtmenge der Methodenanforderungen in iterative Teilmengen aufzuteilen. Eine Teilmenge der Methodenanforderungen, die im Rahmen des Methodenentwicklungskontexts umgesetzt werden können, definiert dann eine Iteration der Methodenweiterentwicklung. Eine Iteration ist nach initialer Ausführung der *Ist-Analyse* und *Verbesserungs-Analyse* eine Teilmenge der Methodenanforderungen an die Ideal-Methode. Die Teilmenge der Methodenanforderungen beschreibt die im Methodenweiterentwicklungsprojekt zu erreichende Soll-Methode. In einem weiteren Methodenweiterentwicklungsprojekt wird die Soll-Methode dann als Ist-Methode in eine weitere Soll-Methode überführt, indem eine weitere Teilmenge der Methodenanforderungen an die Ideal-Methode realisiert wird. Auf diese Weise wird ein iteratives Vorgehen zur Methodenweiterentwicklung definiert.

Aus der Problemstellung in diesem Kapitel wird die folgende verfeinerte Aufgabe der Dissertation abgeleitet.

Teilaufgabe der Dissertation 7
Anpassung der Anforderungserhebung für die Methodenweiterentwicklung zur Erhebung von Methodenanforderungen auf Basis des Methodenkontexts und der übergeordneten Ziele der Methodenweiterentwicklung.

pragmatisches Vorgehen

Diese Aufgabe wird gelöst durch ein pragmatisches Vorgehen zur Erhebung von Methodenanforderungen [Spi13a]. Die Motivation kam aus dem Forschungs- und Entwicklungsprojekt, in dem ein Vorgehen zur Erhebung der Anforderungen im Bereich der Systementwicklung genutzt worden ist. Das Vorgehen auf Basis der allgemeinen Erkenntnisse der Requirements Engineering (bspw. [Rup09] und [RR99] sowie dem für die Systementwicklung spezifischen SYSMOD Vorgehen aus [Wei08] umfasst die Schritte Definieren von Zielen, Erstellung eines Kontextdiagramms, Definition der Stakeholder, Erstellung eines Glossars, Ermittlung funktionaler Anforderungen und Ermittlung nicht-funktionaler Anforderungen. Das Vorgehen hat sich als praktikabel erwiesen und ist gut angenommen worden. Die positive Erfahrung ist der Auslöser das pragmatische Vorgehen von der Systementwicklung auf die Methodenentwicklung zu übertragen.

Struktur des Kapitels

Dieses Kapitel erklärt die Erhebung von Methodenanforderungen im Rahmen der Methodenweiterentwicklung. Dafür wird in Abschnitt 5.1 das Vorgehen nach Ralyté vorgestellt und dessen Schwachpunkte ermittelt. Auf den Erkenntnissen aufbauend wird das vorhandene pragmatische Vorgehen der Anforderungserhebung bei der Systementwicklung auf die Methodenentwicklung übertragen. Dabei ist es wichtig die Einflüsse der gültigen Situationsfaktoren zu erkennen und daraus entsprechende Methodenanforderungen zu erstellen. Das methodische Vorgehen wird im Lösungsbaustein *Verbesserungs-Analyse* definiert und beispielhaft in den Anwendungsbeispielen dieser Arbeit durchgeführt (siehe Abschnitt 5.2). Das Kapitel schließt mit einer Zusammenfassung (siehe Abschnitt 5.3).

5.1 Erhebung von Methodenanforderungen

Einleitung

Dieser Abschnitt unterteilt sich in die Beschreibung zweier Herangehensweisen. Der Ansatz aus [Ral02] wird im Unterabschnitt 5.1.1 kritisch betrachtet. Es wird

erläutert, warum der Ansatz schwierig umzusetzen ist. Der in der Einleitung dieses Kapitels erwähnte pragmatische Ansatz wird im Unterabschnitt 5.1.2 vorgestellt. Es wird gezeigt, dass es notwendig ist den Einfluss von gültigen Situationsfaktoren auf die Methode zu ermitteln. Das wird in Abschnitt 5.1.3 thematisiert.

5.1.1 Methodenanforderungen nach Ralyté

verschiedene Strategien

In [Ral02] wird ein Vorgehen zur Erhebung von Methodenanforderungen vorgeschlagen. Zwei grundlegende Strategien, die *Process driven strategy* zur Erhebung von Methodenanforderungen bei einer Methodenneuentwicklung und die *Intention driven strategy* zur Erhebung von Methodenanforderungen bei einer Methodenanpassung, werden vorgestellt.

Process driven strategy

Betrachten wir die *Process driven strategy* (siehe Abbildung 5.1).

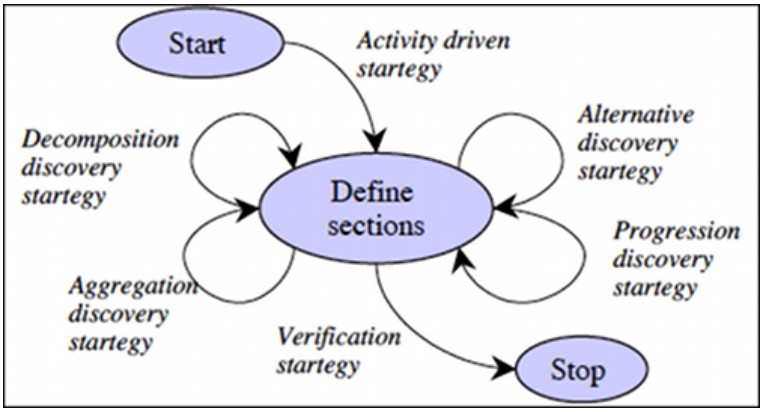


Abbildung 5.1: *Process driven strategy zur Erhebung von Methodenanforderungen ([Ral02])*

Die Abbildung 5.1 zeigt eine Map. Die Map beschreibt ein Prozessmodell als gerichteten, beschrifteten Graphen, bei dem die Knoten Intentionen und die Kanten Strategien beschreiben. Eine Intention ist ein zu erreichendes Ziel und eine Strategie beschreibt eine Guideline zur Erreichung der Intention. Die Abbildung beschreibt auf diese Weise die Guideline *Activity driven strategy* mit der Sections definiert werden können. Eine Section ist in diesem Fall eine Aktivität, die in einer weiteren Map, die Anforderungs-Map, definiert wird.

Beispiel

In einem Beispiel aus [Ral02] wird die Entwicklung einer Methode für ein

Backend-to-Backend-System beschrieben. Es wird also eine Reihe von notwendigen Aktivitäten im Entwicklungsprozess zur Entwicklung des Backend-to-Backend-Systems ermittelt (vergleiche mit Abbildung 5.2).

- Ermittlung von Anforderungen
- Validierung der Anforderungen
- Erstellung einer Sicht auf die Struktur
- Erstellung einer Sicht auf das Objektverhalten
- Erstellung einer Sicht auf das globale Verhalten
- Erstellung einer Sicht auf die Navigationsstruktur

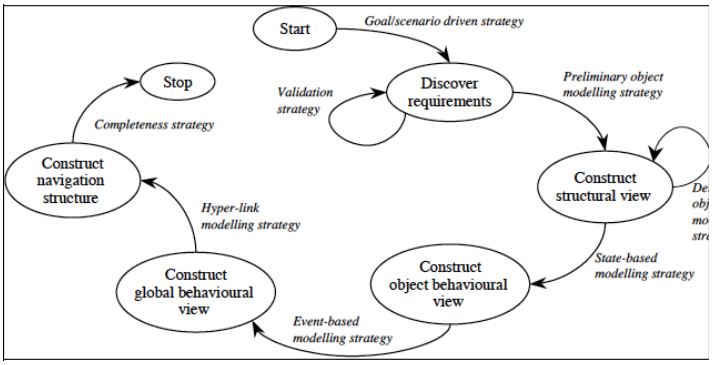


Abbildung 5.2: *Anforderungsmap für das Backend-to-Backend System ([Ral02])*

Finden von Aktivitäten

Die Guideline *Activity driven strategy* ist weitergehend formalisiert und Regeln, wie eine Aktivität beschrieben werden soll, als auch die Konstruktionsbeschreibungen zu Erstellung der Anforderungs-Map sind vorgegeben. Was jedoch fehlt ist eine Anleitung zur methodischen Ermittlung einer Aktivität, wie beispielsweise die *Erstellung einer Sicht auf das Objektverhalten*. Die Identifikation der Aktivitäten wird rein argumentativ durchgeführt und die Aktivität *Erstellung einer Sicht auf das Objektverhalten* wird aus einer impliziten Verhaltensbeschreibung des Backend-to-Backend Systems extrahiert. Die relevante Aussage lautet: *It must specify life cycles of prinzipal objects . . .* (Seite 8 [Ral02]).

formalisiertes Vorgehen

Eine Darstellungsform mittels einer Map wird definiert und formalisiert. Weitere Fragestellungen, wie beispielsweise der Abstraktionsgrad der Aktivitäten angeglichen werden kann, wie Aktivitäten aggregiert und dekomposiert werden können, werden in den Strategien *Progression discovery strategy*, *Alternative discovery strategy*, *Aggregation discovery strategy* und *Decomposition discovery strategy* vorgestellt (siehe Abbildung 5.1). Im Endeffekt entsteht ein Prozess aus der Aneinanderreihung der identifizierten Aktivitäten (siehe Abbildung 5.2).

Kritik

Der Sinn und Zweck dieser Technik wird an dieser Stelle nicht kritisiert. Die Ermittlung der Aktivitäten wirkt intuitiv beim Vorhandensein einer groben Systembeschreibung, wie sie im Beispiel vorhanden ist. Das Erstellen, beziehungsweise Ermitteln einer Systembeschreibung und die Ableitung von notwendigen Aktivitäten für eine Methode, bedarf jedoch Wissen über den Aufbau von Methoden und ist letztendlich nicht intuitiv. Im Gegensatz zu den herkömmlichen RE-Vorgehen, welche in der Standardliteratur über Requirements Engineering (bspw.: [Rup09] und [RR99]) beschrieben werden, wird in der *Activity driven strategy* explizit das Wissen über Aktivitäten, als Element der Methode, berücksichtigt. Der Ansatz zeichnet sich als sehr formal aus und bringt dadurch einen Aufwand bezüglich der Erlernbarkeit der Map-Technik mit sich. Des Weiteren müsste er erweitert werden, sobald Situationsfaktoren oder andere Methodenelemente als Aktivitäten durch Methodenanforderungen abgebildet werden sollen, beispielsweise Artefakte, Techniken, Hilfsmittel oder Rollen der Entwicklungsmethode. Bei der Arbeit im Industrieprojekt bei FTS ist dieser Ansatz schnell als *zu kompliziert* abgestempelt worden. Das liegt hauptsächlich an der vorgeschlagenen Darstellungsform. Weiterhin ist erkannt worden, dass die Definition der Anforderungen selber keinem methodischen Vorgehen unterliegt, sondern wie gehabt intuitiv, nach besten Wissen und Gewissen, durchgeführt wird.

5.1.2 Pragmatisches Vorgehen zur Erhebung von Methodenanforderungen

pragmatische Anforderungserhebung

Bei der Erhebung und Beschreibung von Anforderungen im Bereich der Systemarchitekturentwicklung bei FTS ist ein pragmatisches Vorgehen auf Basis der Standardliteratur genutzt worden. Es hat sich im Umfeld von FTS als praktikabel erwiesen und ist gut angenommen worden. Diese positive Erfahrung ist der Auslöser den Fokus des pragmatischen Vorgehens von der Systementwicklung auf die Methodenentwicklung zu verschieben mit dem Anspruch mehr relevante Aspekte für die Methode zu berücksichtigen, als nur Aktivitäten wie in [Ral02] beschrieben wird. Weiterhin soll das Vorhandensein des erstellten Methodenkontexts berücksichtigt werden.

das Vorgehen

Das pragmatische Vorgehen für die Anforderungserhebung in der Systementwicklung sieht die folgenden allgemeinen Schritte vor:

1 Definieren von Zielen

2 Erstellung eines Kontextdiagramms
3 Definition der Stakeholder
4 Erstellung eines Glossars
5 Ermittlung funktionaler Anforderungen
6 Ermittlung nicht-funktionaler Anforderungen

Angleichung des Vorgehens

Die folgenden Absätze beschreiben für jeden Schritt, wie der Fokus von der Systementwicklung auf die Methodenentwicklung verschoben wird und diskutiert, was das konkret für die einzelnen Schritte bedeutet.

1 Definieren von Zielen Die Ziele einer Methode sind zu beschreiben. Hierbei stehen verschiedene Möglichkeiten im Raum, warum eine Methode weiterentwickelt wird. Im Sinne dieser Dissertation wird das Hauptaugenmerk auf die Qualitätssteigerung der Produkte liegen, die mit der Methode erstellt werden. Die folgende Auflistung beschreibt eine Übersicht über weitere mögliche Ziele, um die unterschiedlichsten Gründe für eine Methodenweiterentwicklung darzustellen:

- Qualitätssteigerung der zu entwickelnden Produkte
- Verbesserung des Projektmanagements
- Einhaltung von Normen und Standards
- Erhöhung von Maturity Level (bspw.: CMMI)
- Erhöhung des Verständnisses über die Entwicklungsaktivitäten
- Einführung effizienterer Arbeitsweisen
- Umsetzung von Anforderungen von Auftraggebern oder Auftragnehmern

Mit dem Wissen über das definierte Ziel werden intuitiv die unterschiedlichsten Dringlichkeiten und Umsetzungsanforderungen klar. Im Gegensatz dazu ist es für die Methodenweiterentwicklung katastrophal wenn Unklarheit darüber besteht, warum die Anstrengung der Methodenweiterentwicklung geleistet werden soll.

2 Erstellung eines Kontextmodells Ursprünglich wird die Tätigkeit der Beschreibung des Kontexts dazu genutzt mögliche Schnittstellen eines Systems zu identifizieren, um diese im weiteren Anforderungserhebungsprozess weiter zu detaillieren. Darüber hinaus sollen Rahmenbedingungen erkannt werden, die ein System im Betrieb einhalten soll. Transferieren wir diese Idee auf die Methodenentwicklung, scheint die Idee der Schnittstellen zunächst abwegig, wird aber die Ent-

wicklung von Teilmethoden oder die Anpassung bestehender Methoden betrachtet, werden auch hier Schnittstellen erkannt. Schnittstellen können aus notwendigen Artefakten zum Durchführen einer Aktivität oder aus Artefakten, welche von einer Aktivität erstellt werden, bestehen. Neben den Schnittstellen der Methode wird die Beschreibung der Rahmenbedingungen einer Methode benötigt. Das ist der bereits festgelegte Methodenkontext aus der Ist-Analyse.

3 Definition der Stakeholder Die Stakeholder der Methode sollen definiert werden. Im Vergleich zu der Frage *Wer hat Interesse an dem System?* aus dem ursprünglichen Vorgehen ist es einfach diese Frage umformulieren zu *Wer hat Interesse an der Methode?*. Eine Auswahl der relevanten Personengruppen scheint abhängig von dem Ziel der Methode zu sein. Ein Beispiel ist die Verbesserung einer Teilmethode, bei der ein Systemarchitekt konkrete Vorschriften zur Erstellung einer Spezifikation erhalten möchte, ein Entwickler konkrete Anforderungen an bestimmte Inhalte der Spezifikation stellt oder der Manager das Ziel der konstruktiven Qualitätssicherung definiert hat und diese in der Entwicklungsmethode integrieren möchte. Ein weiteres Beispiel ist gegeben, mit dem vom Manager definierten Ziel der Verbesserung der Zusammenarbeit mit dem Zulieferer. Stakeholder können in diesen Fällen Systemarchitekt, Entwickler, Manager und Zulieferer sein. Es gibt mannigfaltige Kombinationen von Stakeholdern. Zur professionellen Erhebung möglicher Stakeholder ist eine Stakeholderanalyse, wie in [DH07] beschrieben, hilfreich.

4 Erstellung eines Glossars Die Erstellung eines Glossars ist für das gemeinsame Verständnis wichtig. Bei einer Methode sollten die folgenden Themen detailliert werden:

- Umfang der Methode: Was ist in der Methode zu berücksichtigen?
- Teilmethoden: Wie wird die Methode in verschiedene Teil-Methoden gegliedert und wie heißen die Teil-Methoden?
- Methodenelemente: Wie werden die einzelnen Elemente der Methode genannt?

5 Ermittlung funktionaler Anforderungen = Ermittlung notwendiger Methodenelemente Die Anpassung des Schritts *Ermittlung funktionaler Anforderungen* an

die Methodenentwicklung bedarf der Definition, was eine funktionale Anforderung einer Methode sein kann. Funktionale Anforderungen beschreiben, welche Funktionalitäten ein System umsetzen soll. Was soll denn die Methode umsetzen? Eine Methode soll die Aktivitäten, Artefakte, Prozesse und Hilfsmittel definieren, beschreiben und dokumentieren, die zur Entwicklung des Entwicklungsziels führen. Also können die Methodenelemente als *funktionale* Anforderungen verstanden werden. Die Tätigkeit *Ermittlung von funktionalen Anforderungen* wird durch *Ermittlung notwendiger Methodenelemente* ersetzt. An dieser Stelle kann die *Activity driven strategy* aus [Ral02] aufgegriffen werden zur Identifikation der Aktivitäten der Methode. Es werden weitere Vorgehen zur Ermittlung der weiteren Methodenelemente gebraucht. Auf Basis der Kernelemente einer Methode, die in dem Metamodell (siehe Abbildung 2.1) definiert sind, wird die Aufgabe der Ermittlung von Methodenelementen wie folgt detailliert.

5.1 Ermittlung von notwendigen Artefakttypen
5.2 Ermittlung von notwendigen Aktivitäten (und zusammengesetzten Aktivitäten)
5.3 Ermittlung von notwendigen Hilfsmitteln
5.4 Ermittlung von notwendigen Rollen
5.5 Ermittlung von notwendigen Phasen
5.6 Ermittlung von notwendigen Meilensteinen

Momentan gibt es keine formalisierten, definierten Herangehensweisen, wie die Aktivitäten (5.1 - 5.6) durchgeführt werden können. Es bedarf einer detaillierten Analyse der Entwicklungsdomäne, welche eine Übersicht der aktuellen Forschungsergebnisse, Begutachtung der vorherrschenden bekannten Spezifikationen und dessen Schwierigkeiten oder die Betrachtung bereits vorhandener Entwicklungsziele umfasst. Durch diese Abhängigkeit zur Entwicklungsdomäne ist es meist unerlässlich auf einen Domänenexperten zurück zu greifen. Die Begutachtung der Domäne kann im größten Detail durchgeführt werden, die Definition der Anforderungen liegt dem gegenüber aber auf einem abstrakteren Level und so ist es ausreichend die wichtigsten und offensichtlichsten Methodenelemente in den Anforderungen zu erheben. In der Praxis eignen sich dafür Interviews und Workshops mit Domänenexperten.

5 Ermittlung nicht-funktionaler Anforderungen = Ermittlung von Methodenanforderungen auf Basis gültiger Situationsfaktoren Nach [RR99] beschreiben nicht-

funktionale Anforderungen Attribute, die ein Produkt oder System haben soll. Etwas detaillierter werden die nicht-funktionalen Anforderungen in [CPL09] als Eigenschaften und Rahmenbedingungen des Produkts angesehen. Eine sehr abstrakte nicht-funktionale Anforderung an die zu erstellenden Methoden wird in dieser Arbeit hervorgehoben, die Situationsgerechtigkeit. In den vorangegangen Abschnitten ist die Situationgerechtigkeit durch die Berücksichtigung von Situationsfaktoren definiert worden. Als hilfreich bei der Erstellung von nicht-funktionalen Anforderungen ist das Vorhandensein bestehender Klassifikationen oder Checklisten. Beispielsweise kann zur Definition von nicht-funktionalen Anforderungen bei der Systementwicklung die IVENA-Gliederung [SOP09] zu Hilfe genommen werden. Im Vergleich dazu wird die Ermittlung der nicht-funktionalen Anforderungen nun mit Hilfe der Checkliste möglicher Situationsfaktoren aus Abschnitt 4.3 durchgeführt. Die Checkliste ist bereits in der Ist-Analyse zur Ermittlung der gültigen Situationsfaktoren angewendet worden. Also müssen Methodenanforderungen auf Basis der gültigen Situationsfaktoren erstellt werden. Der Schritt *Ermittlung nicht-funtionaler Anforderungen* wird durch die Aktivität *Ermittlung von Methodenanforderungen auf Basis gültiger Situationsfaktoren* ersetzt.

5.1.3 Einflüsse von Situationsfaktoren

Einfluss auf die Methode

Für die Ermittlung der Methodenanforderungen auf Basis gültiger Situationsfaktoren ist es wichtig den konkreten Einfluss des Situationsfaktors auf die Methode zu kennen, damit eine entsprechende Methodenanforderung erstellt werden kann. Die Ermittlung des konkreten Einflusses ist allerdings schwierig, denn es gibt nur wenig anerkannte Aussagen darüber.

bekannte Einflüsse

Lediglich in [Coc00] werden einige konkrete Einflüsse der Situationsfaktoren auf eine Methode angegeben:

- Eine größere Gruppe von Entwicklern braucht eine größere Methode.
- Umso kritischer der Einsatz des zu entwickelden Systems ist, umso höher muss die konstruktive Qualitätssicherung sein.
- Eine kleine Vergrößerung der Methode kann eine große Erhöhung Projektkosten mit sich ziehen.
- Eine interaktive und persönliche Kommunikation ist am effizientesten.

Erstellung der richtigen MA

Die Ermittlung solcher Einflüsse ist eigentlich für jeden Situationsfaktor notwendig, damit im Falle der Gültigkeit die richtige Methodenanforderung definiert werden kann. Allerdings bleibt aus Mangel an Erkenntnissen jedoch nur die Möglichkeit eine nachvollziehbare Schlussfolgerung für eine passende Methodenanforderung zu treffen. Das ist vergleichbar mit der Anforderungserhebung in Softwareentwicklungsprojekten. Nehmen wir beispielsweise die nicht-funktionale Anforderung *Usability* am Beispiel von grafischen Benutzungsystemen. Sie umfasst die unterschiedlichsten Auffassungen zur Realisierung der Anforderung. Beispielsweise gibt es verschiedene ISO Standards (9241, 13407, 14915 und weitere), die Eigenschaften von benutzerfreundlichen Oberflächen definieren (Verfügbarkeit, Übersichtlichkeit, Selbstbeschreibungsfähigkeit, Erlernbarkeit und weitere). Für diese Eigenschaften müssten wieder einzuhaltende Regeln oder Guidelines zur Verfügung stehen, um passende verfeinerte Anforderungen definieren zu können. Letztendlich müssen Metriken oder Expertenbefragung belegen, dass die Anforderungen entsprechend umgesetzt worden sind. Für die Situationsfaktoren gibt es keine Regeln oder Guidelines. Das heißt, in einem ersten Schritt müssen Methodenanforderungen auf Basis gültiger Situationsfaktoren intuitiv erstellt werden. In einem weiteren Schritt muss überprüft werden, ob eine erstellte Anforderung zur Lösung einer Herausforderung, die über einen gültigen Situationsfaktor beschrieben worden ist, beiträgt.

5.2 Lösungsbaustein: Verbesserungs-Analyse

In den vorangegangenen Abschnitten dieses Kapitels ist beschrieben worden, dass der Methodenentwickler im Rahmen der *Verbesserungs-Analyse* Methodenanforderungen erheben muss. Die nun anstehende Aufgabe besteht darin die Erhebung der Methodenanforderungen formal zu definieren. Das wird im Rahmen der formalen Definition der Phase *Verbesserungs-Analyse* durchgeführt (siehe Abbildung 5.3).

Die formale Definition der Aktivität mit Hilfe der Sprache MMMB wird in Abschnitt 5.2.1 vorgestellt. Die Aktivität „Verbesserungs-Analyse“ wird anschließend zum einen auf die System-Architektur-Spezifikation (siehe Abschnitt 5.2.2) und zum anderen auf die CIM-Entwicklung (siehe Abschnitt 5.2.3) angewendet. In Abschnitt 5.2.4 wird dargestellt, welche Vorteile die Methodenanforderungen bei der iterativen Methodenweiterentwicklung mit sich bringen.

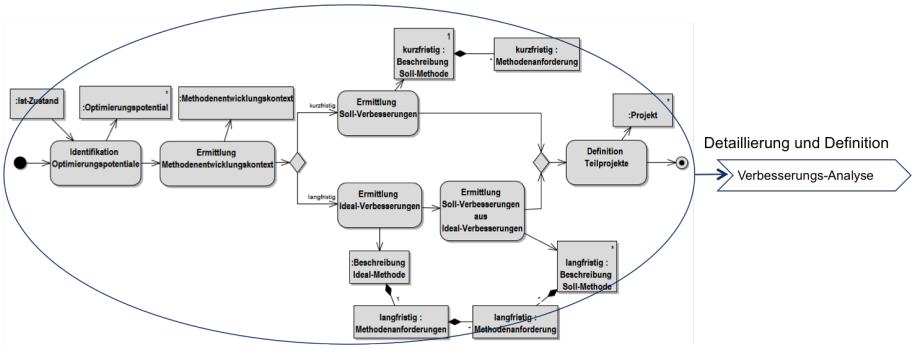


Abbildung 5.3: *Detaillierung: Verbesserungs-Analyse*

5.2.1 Detaillierung der Aktivität *Verbesserungs-Analyse*

Definition Soll-Methode

Die zu erledigende Aufgabe ist es die Methodenanforderungen an die Soll-Methode zu definieren. Auf Basis des vorliegenden Ist-Zustands muss zuerst das Optimierungspotenzial ermittelt werden. Der im Ist-Zustand definierte Methodenkontext beschreibt zunächst die Herausforderungen der bestehenden Methode im Rahmen des Einsatzgebietes der Methode. Darüber hinaus zeigt die Ist-Methode selber auch direkte Schwächen auf. Beide Aspekte werden als Optimierungspotenzial betrachtet. Danach wird der Methodenentwicklungskontext definiert, denn er gibt die Rahmenbedingungen für die Methodenweiterentwicklung vor. Sind Optimierungspotenzial und Methodenentwicklungskontext bekannt, werden die Methodenanforderungen auf Basis des pragmatischen Vorgehens erhoben (vgl. Abschnitt 5.1.2). Es gibt dabei zwei Wege für das weitere Vorgehen im Methodenweiterentwicklungsprojekt. Die kurzfristige Lösung, bei der nur die notwendigsten Anforderungen erhoben werden, um eine schnelle Lösung zu erhalten, wird in dieser Arbeit aufgrund ihrer Schwächen nicht weiter thematisiert (vgl. 2.3.2). Die langfristige Lösung mit Definition aller Methodenanforderungen an eine Ideal-Methode bedingt eine Priorisierung der Methodenanforderungen und Bildung von Teilprojekten für die Weiterentwicklung der Ist-Methode zu einer Soll-Methode im Rahmen des Methodenentwicklungskontexts.

Formalisierung mit MetaMe++

Die Überführung der Schritte in MetaMe++ wird als nächstes betrachtet. Das Prozessmodell der Methodenweiterentwicklung der zusammengesetzten Aktivität *Verbesserungs-Analyse* wird im Folgenden beschrieben (siehe Abbildung 5.4).

Es gibt zwei parallel voneinander durchführbare Aktivitäten, die *Ermittlung Methodenentwicklungskontext* und die *Ermittlung Methodenanforderungen*.

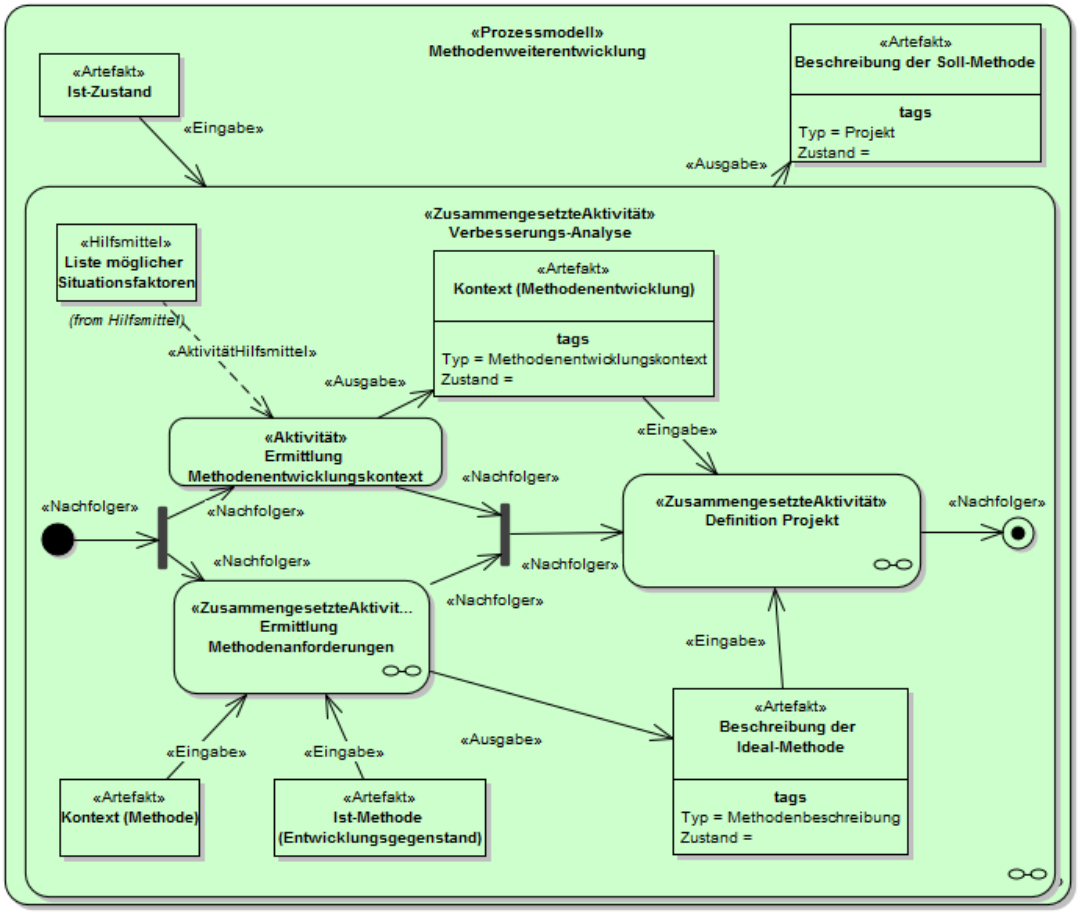


Abbildung 5.4: *Verbesserungs-Analyse mit MetaMe++ – Prozessmodell*

Ermittlung Methodenentwicklungskontext

Die Aktivität *Ermittlung Methodenentwicklungskontext* nutzt die Liste möglicher Situationsfaktoren als Hilfsmittel. Die Aktivität definiert mit Hilfe der Checkliste die gültigen Situationsfaktoren für den Methodenentwicklungskontext festzulegen (siehe Abschnitt 4.3.11). Das Artefakt *Kontext (Methodenentwicklung)* beschreibt letztendlich den ermittelten Methodenentwicklungskontext.

«Aktivität» Ermittlung Methodenentwicklungskontext

Eingabe: Kategorie Methodenentwicklungskontext der „Liste möglicher Situationsfaktoren“
Beschreibung: Für jeden referenzierten Situationsfaktor wird überprüft, ob der Situationsfaktor eine Herausforderung für die Weiterentwicklung der Methode darstellt. Der Kontext ist das Methodenentwicklungsprojekt.
Ausgabe: Kontext (Methodenentwicklung)

Ermittlung Methodenanforderungen

Die zusammengesetzte Aktivität *Ermittlung Methodenanforderungen* definiert

die Ermittlung der Methodenanforderungen, wie sie im pragmatischen Vorgehen (siehe Abschnitt 5.1.2) vorgestellt worden ist. Nach Durchführung der Aktivität erhalten wir eine Beschreibung einer Ideal-Methode.

Projektdefinition

Auf Basis der Beschreibung der Ideal-Methode, die sämtliche Methodenanforderungen enthält, wird ein konkretes Projekt für die Methodenweiterentwicklung definiert (Aktivität *Definition Projekt*). Dazu wird als Eingabe zusätzlich der Methodenentwicklungskontext betrachtet.

Ideal-Methode

Zunächst betrachten wir detaillierter die zusammengesetzte Aktivität *Ermittlung Methodenanforderungen*, um besser zu verstehen, welche Informationen in der Beschreibung der Ideal-Methode enthalten sind (siehe Abbildung 5.5).

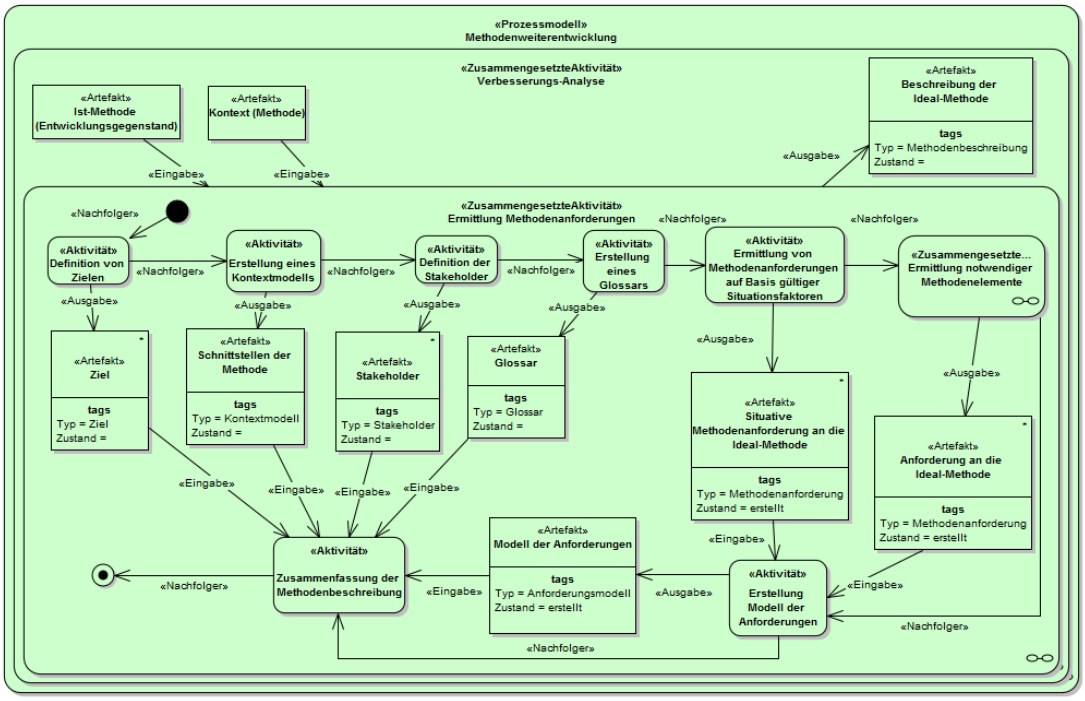


Abbildung 5.5: *Ermittlung von Methodenanforderungen mit MetaMe++ – Prozessmodell*

Wie bereits erwähnt, stammen die Aktivitäten *Definition von Zielen*, *Erstellung eines Kontextmodells*, *Definition der Stakeholder*, *Erstellung eines Glossars*, *Ermittlung von Methodenanforderungen auf Basis gültiger Situationsfaktoren* und *Ermittlung notwendiger Methodenelemente* aus dem pragmatischen Vorgehen aus Abschnitt 5.1.2. Es wird nochmal darauf hingewiesen, dass die Ist-Methode

und der Methodenkontext für die Durchführung der Aktivität zur Verfügung stehen.

«Aktivität» Definition von Zielen
Beschreibung: Die Ziele werden vom Auftraggeber der Methodenweiterentwicklung vorgegeben oder auf Basis der Herausforderungen, die bei der Ist-Analyse erkannt worden sind, ermittelt. Sie sind also aus den gültigen Situationsfaktoren abzuleiten.
Ausgabe: Ziele

«Aktivität» Erstellung eines Kontextmodells
Beschreibung: Das Kontextmodell wird aus der Ist-Methode abgeleitet. Entweder wird die Ist-Methode als Ganzes betrachtet oder es können Phasen oder Aktivitäten als Teilmethode definiert werden. Die Ist-Methode wird auf Ein- und Ausgabeartefakte sowie einen Platzhalter der Methode oder Teilmethoden reduziert. Auf diese Weise erhält man ein Kontextmodell.
Ausgabe: Schnittstellen der Methode

«Aktivität» Definition der Stakeholder
Beschreibung: Die Stakeholder können aus der Ist-Methode abgeleitet werden. Die in der Ist-Methode definierten Rollen stellen die Stakeholder der Methode dar. Weitere Stakeholder können durch die Auftraggeber der Methodenweiterentwicklung vorgegeben werden. Der Methodenentwickler ist implizit ein Stakeholder der weiter zu entwickelnden Methode.
Ausgabe: Stakeholder

«Aktivität» Erstellung eines Glossars
Beschreibung: Das Glossar soll alle Begriffe enthalten, die im Rahmen der Ist- und Verbesserungs-Analyse von Bedeutung sind. Das sind die Begriffe der Methodenelemente der Ist-Methode, insbesondere Artefakte, Artefakttypen, Rollen, Hilfsmittel und Meilensteine. Aktivitäten beschreiben in der Regel keine Begriffe. Es kann jedoch sein, dass eine Zusammengesetzte Aktivität durch einen prägnanten Begriff, wie beispielsweise *Anforderungsanalyse*, beschrieben wird. Dieser muss dann auch Bestandteil des Glossars sein.
Ausgabe: Glossar

«Aktivität» Ermittlung von Methodenanforderungen auf Basis gültiger Situationsfaktoren
Eingabe: Methodenkontext, Ziele
Beschreibung: Der Methodenkontext enthält eine Liste gültiger Situationsfaktoren. Weiterhin sind im Rahmen der Verbesserungs-Analyse bereits Ziele definiert worden. Die Informationen aus beiden Quellen müssen in Anforderungen überführt werden. Es müssen alle gültigen Situationsfaktoren sowie alle Ziele betrachtet werden, damit eine gewisse Vollständigkeit erreicht wird.
Ausgabe: Situative Methodenanforderungen an die Ideal-Methode

Die Aktivitäten zur Ermittlung der Informationen *Ziele, Kontextmodell, Stakeholder* und *Glossar* erstellen entsprechende Artefakte, wie in Abbildung 5.5 zu erkennen ist. Die Aktivität *Ermittlung notwendiger Methodenelemente* ist zusammengesetzt. Im Vergleich zum Ansatz nach Ralyte (Abschnitt 5.1.1) sollen nicht nur Anforderungen an Aktivitäten definiert werden, sondern an alle notwendigen Methodenelemente. Entsprechende Aktivitäten werden in Abbildung 5.6 dargestellt.

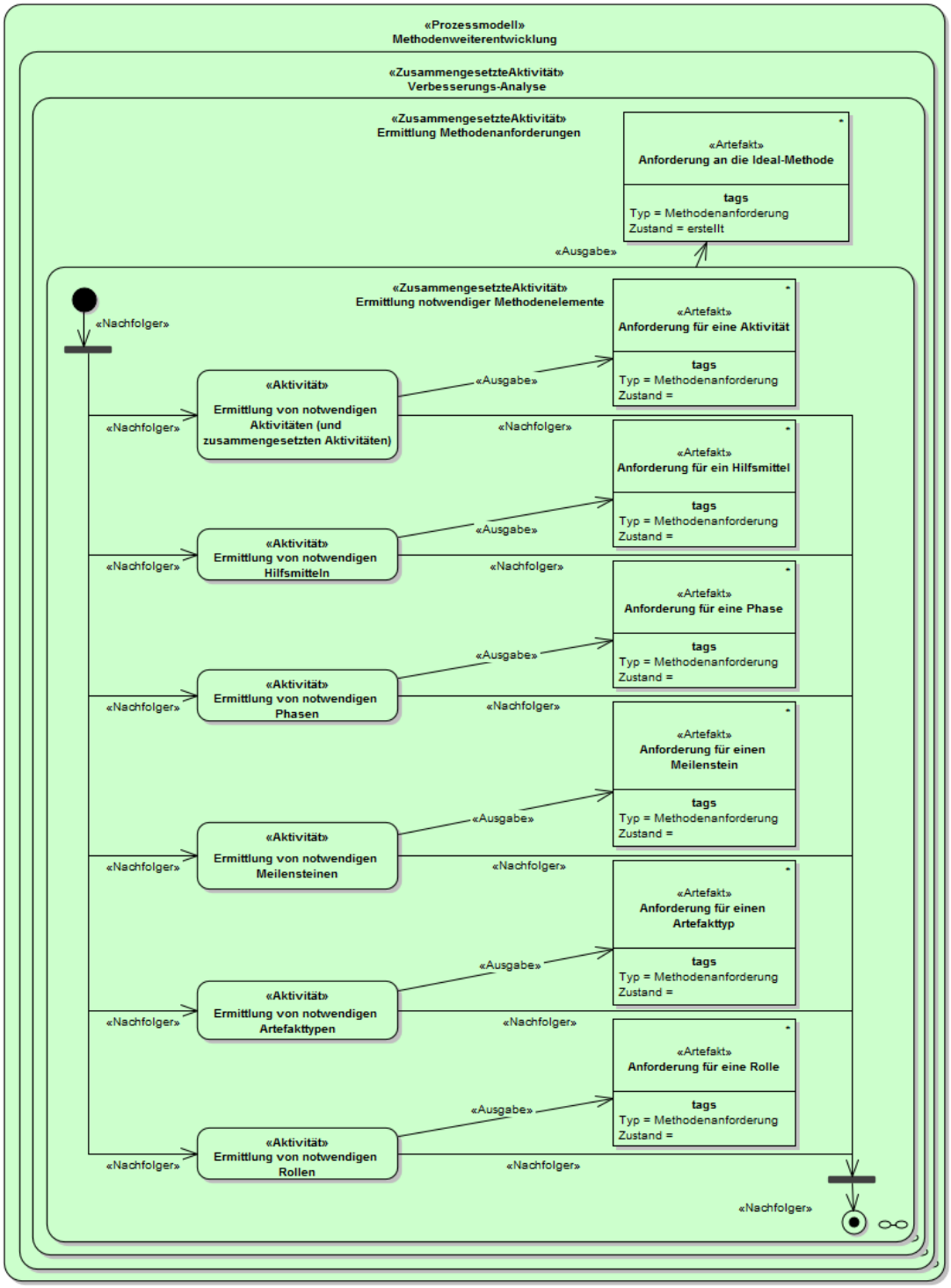


Abbildung 5.6: *Ermittlung notw. Methodenelemente mit MetaMe++ – Prozessmodell*

«Aktivität» Ermittlung von notwendigen Aktivitäten und zusammengesetzten Aktivitäten

Beschreibung: Für alle gewünschten oder erforderlichen Aktivitäten soll eine Anforderung erstellt werden.
Mögliche Vorlage: *Die Methode soll die Aktivität <Name der Aktivität> enthalten.*

«Aktivität» Ermittlung von notwendigen Hilfsmitteln

Beschreibung: Wünschen sich die Stakeholder der Methode bestimmte Hilfsmittel, wie Dokumentvorlagen, muss eine entsprechende Anforderung erstellt werden. Mögliche Vorlage: *Die Methode soll <Name des Hilfsmittels> als Hilfsmittel für die Aktivität <Name der Aktivität> definieren.*

«Aktivität» Ermittlung von notwendigen Meilensteinen

Beschreibung: Werden Meilensteine vorgegeben oder sollen etabliert werden, muss eine entsprechende Anforderung erstellt werden.
Mögliche Vorlage: *Die Methode soll den Meilenstein <Name des Meilensteins> definieren. Der Meilenstein soll über die folgenden Artefakte definiert werden: <Namen des Artefakts > im Zustand <Zustand>*

«Aktivität» Ermittlung von notwendigen Phasen

Beschreibung: Soll der Entwicklungsprozess über Phasen strukturiert werden, sind entsprechende Anforderungen anzugeben.
Mögliche Vorlage: *Die Methode soll die Phase <Name der Phase> definieren. Die Phase soll die folgenden Aktivitäten enthalten: <Name der Aktivität>*

«Aktivität» Ermittlung von notwendigen Rollen

Beschreibung: Sollen Rollen als Teilnehmer oder Durchführer für einzelne Aktivitäten definiert werden, müssen entsprechende Anforderungen definiert werden.
Mögliche Vorlage: *Die Methode soll die Rolle <Name der Rolle> als (Durchführer|Teilnehmer) für die Aktivität <Name der Aktivität> definieren.*

«Aktivität» Ermittlung von notwendigen Artefakttypen

Beschreibung: Sind Artefakte im Entwicklungsprozess notwendig als Ein- oder Ausgabeartefakte bestimmter Aktivitäten, sind die Artefakttypen der Artefakte als Anforderungen aufzunehmen. Eventuell kann ein mögliches Lebenszyklusmodell eines Artefakttyps ebenfalls schon in den Anforderungen aufgenommen werden.
Mögliche Vorlage: *Die Methode soll den Artefakttyp <Name des Artefakttyps> definieren. Im Lebenszyklusmodell sind gegebenenfalls die folgenden Zustände zu definieren: <Name des Zustands>.*

Ablage der MA

Zusätzlich zum pragmatischen Vorgehen definiert MetaMe++ die Ablage der Methodenanforderungen in einem Modell, das Anforderungsmodell. Dieses ist notwendig, damit die Anforderungen verwaltet und in einem weiteren Schritt Abhängigkeiten zwischen den Methodenanforderungen definiert werden können.

«Aktivität» Erstellung Modell der Anforderungen
Eingabe: Methodenanforderungen
Beschreibung: Die aus dem vorangegangenen Schritten ermittelten Methodenanforderungen sollen modellbasiert verwaltet werden. Ein Werkzeug für die Anforderungsverwaltung muss die Möglichkeiten besitzen Anforderungen zu priorisieren und Abhängigkeiten zwischen den Anforderungen zu definieren. Jede ermittelte Anforderung soll in das Anforderungsmodell integriert werden.
Ausgabe: Anforderungsmodell

Dokumentation

Abschließend definiert die Aktivität *Zusammenfassung der Methodenbeschreibung* die Sammlung und Dokumentation aller ermittelten Informationen in dem Artefakt *Beschreibung der Ideal-Situation*.

«Aktivität» Zusammenfassung der Methodenbeschreibung
Eingabe: Ziele, Schnittstellen der Methode, Stakeholder, Glossar, Anforderungsmodell
Beschreibung: Die Informationen sollen konsolidiert werden. Es ist möglich die Informationen in einem Dokument oder in einem Modell zusammenzufassen.
Ausgabe: Methodenbeschreibung

Projekte

Nun wissen wir, welche Informationen in der Beschreibung der Ideal-Situation enthalten sind und kommen zu der Verfeinerung der zusammengesetzten Aktivität *Definition Projekt* (siehe Abbildung 5.7).

Priorisierung der MA

Eine Menge an Methodenanforderungen ist erhoben worden. Es wird jedoch der Fall sein, dass die im Methodenentwicklungskontext erhobenen Situationsfaktoren und die Informationen über die zur Verfügung stehenden Ressourcen es nicht zulassen alle Methodenanforderungen im Methodenweiterentwicklungsprojekt zu bearbeiten. Deshalb müssen Teilmengen der Methodenanforderungen erstellt werden. Eine Teilmenge definiert ein Methodenweiterentwicklungsprojekt. Zur Erstellung der Teilmengen von Methodenanforderungen ist es zunächst notwendig die Anforderungen zu priorisieren.

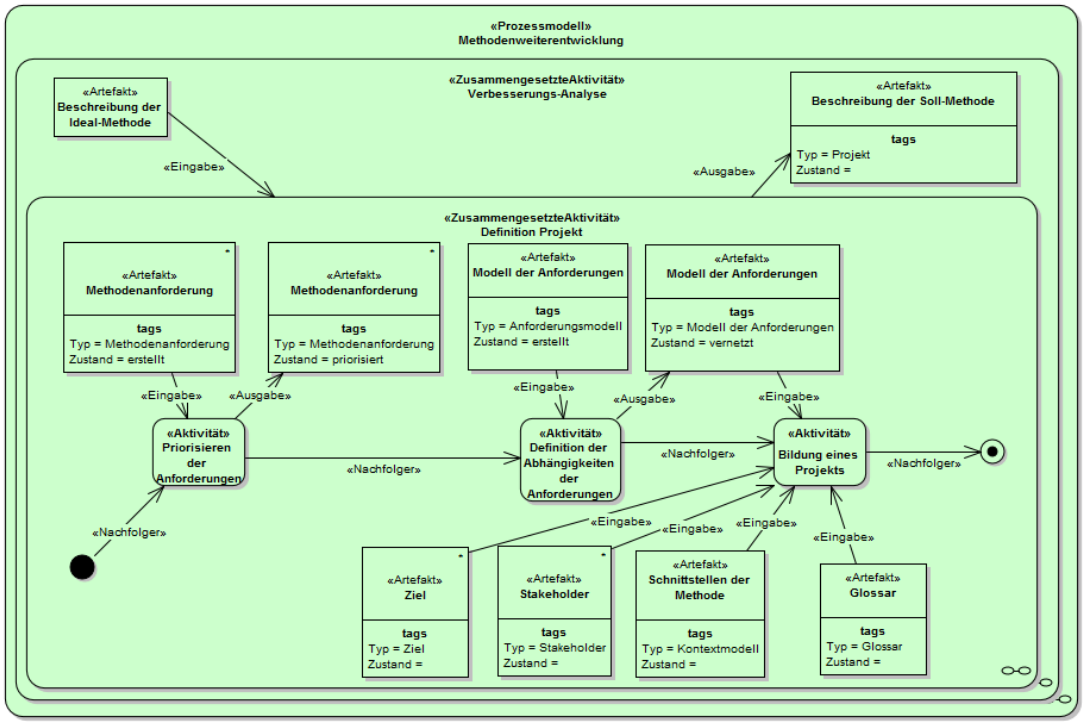


Abbildung 5.7: *Verbesserungs-Analyse mit MetaMe++ – Produktmodell*

«Aktivität» Priorisieren der Anforderungen

Eingabe: Methodenanforderungen aus der Methodenbeschreibung
Beschreibung: Jede Anforderung im Anforderungsmodell soll priorisiert werden.
Ausgabe: Priorisierte Methodenanforderung

Abhängigkeiten

Danach müssen die Abhängigkeiten der Anforderungen untereinander im Anforderungsmodell definiert werden.

«Aktivität» Definition der Abhängigkeiten der Anforderungen

Eingabe: Anforderungsmodell aus der Methodenbeschreibung
Beschreibung: Für alle Methodenanforderungen sind deren Abhängigkeiten zu definieren. Ist eine Anforderung 1 abhängig von einer Anforderung 2, soll die Anforderung 2 als abhängig von Anforderung 1 markiert werden.
Ausgabe: Anforderungsmodell mit Abhängigkeiten der Methodenanforderungen

Bildung von Teilmengen der MA

Mit Hilfe der Abhängigkeiten können Teilmengen von Anforderungen gebildet werden, so dass enthaltene Anforderungen nicht abhängig von Anforderungen ei-

ner anderen Teilmenge sind. Die Priorisierung hilft bei der Auswahl der Anforderungen für eine Teilmenge, so dass die wichtigen Anforderungen ausgewählt werden können. Auf diese Weise gebildete Teilmengen von Anforderungen beziehen sich jedoch nur auf eine Teilmenge der ermittelten Ziele der Methodenweiterentwicklung, Begriffe aus dem Glossar, Stakeholder und Schnittstellen der Methoden im Kontextmodell. Darum muss abschließend eine Methodenbeschreibung erstellt werden, die nur die entsprechenden Teilinformationen enthält. Das geschieht in der Aktivität *Bildung eines Projekts*.

«Aktivität» Bildung eines Projekts

Eingabe: Methodenbeschreibung
Beschreibung: Zuerst muss die eine passende Teilmenge der Methodenanforderungen gefunden werden. Die Teilmenge soll die Methodenanforderungen mit den höchsten Prioritäten enthalten. Des Weiteren darf die Teilmenge der Methodenanforderungen keine Abhängigkeiten zu Methodenanforderungen enthalten, die nicht in der Teilmenge enthalten sind. Es müssen die relevanten Ziele, Stakeholder, Schnittstellen der Methode und Glossar-Einträge ermittelt werden, die zu der Teilmenge der Methodenanforderungen gehören. Aus diesen Informationen wird eine Methodenbeschreibung der Soll-Methode abgeleitet. Die Soll-Methode definiert im Sinne einer Iteration einen Schritt in Richtung Ideal-Methode. Sie wird einem Projekt hinzugefügt, welches den organisatorischen Rahmen der Iteration der Methodenweiterentwicklung darstellt.
Ausgabe: Projekt

Der strukturelle Zusammenhang aller Artefakte ist im Produktmodell mit den grundlegenden Artefakttypen festgelegt (siehe Abbildung 5.8).

Soll-Methoden

Der Methodenentwicklungskontext ist als Artefakttyp definiert. Des Weiteren ist ein Projekt erstellt worden, welches strukturell aus einer Methodenbeschreibung besteht. Das ist der Fall für das erstellte Projekt mit der Beschreibung einer Soll-Methode. Eine Methodenbeschreibung kann jedoch auch autonom und nicht nur als Teil eines Projektes bestehen, wie es in dem Fall der Beschreibung der Ideal-Methode ist. Eine Methodenbeschreibung besteht aus Zielen, einem Glossar, einem Kontextmodell der Methode sowie der Beschreibung von Stakeholdern. Das Anforderungsmodell ist ebenfalls ein Teil der Methodenbeschreibung. Das Modell hat die Zustände *erstellt*, wenn initial alle Methodenanforderungen hinzugefügt werden und *vernetzt*, nachdem die Abhängigkeiten der Methodenanforderungen hinzugefügt worden sind. Eine einzelne Methodenanforderung ist demnach ein Element des Anforderungsmodells und hat die Zustände *erstellt* nach initialer Erhebung und *priorisiert*, nachdem die Priorisierung zugewiesen worden ist. Die Definition der Sprachen ist für diese Artefakttypen offen gelassen. Es obliegt dem Methodenentwickler, welche Darstellung für ihn passend ist. Das wird durch die

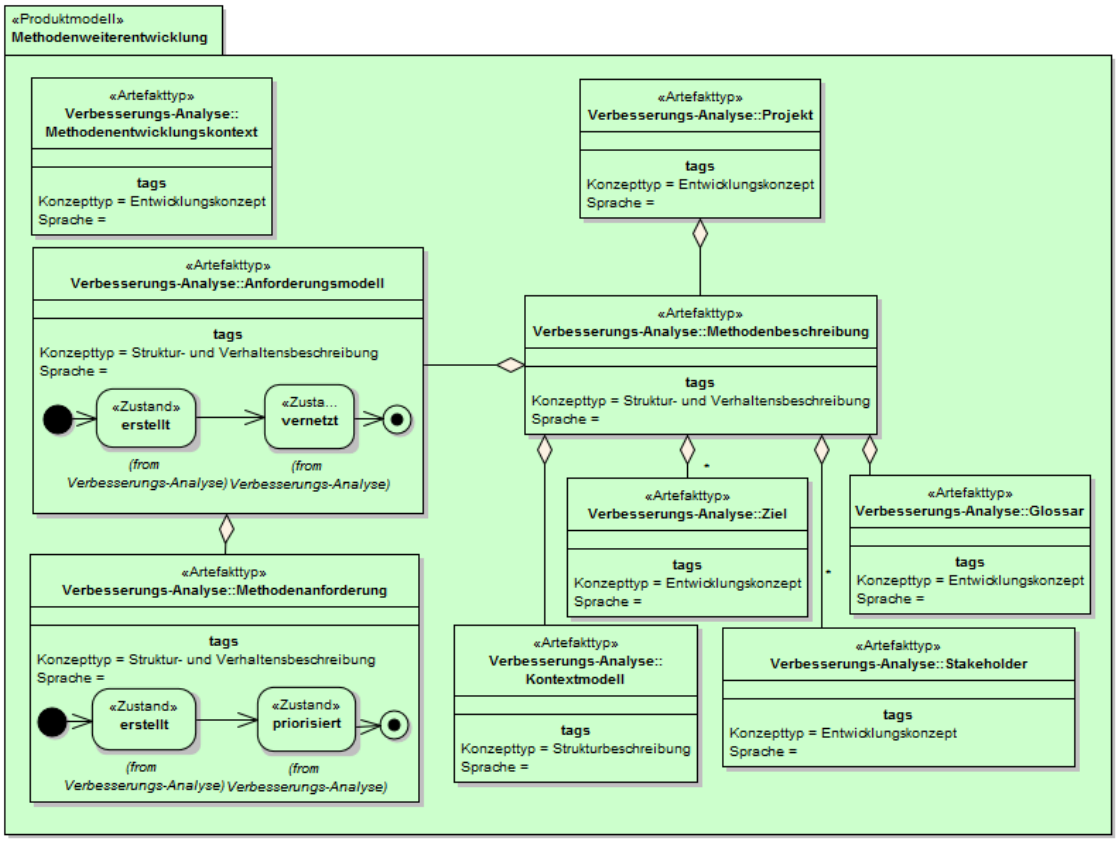


Abbildung 5.8: *Verbesserungs-Analyse mit MetaMe++ – Produktmodell*

genutzten Werkzeuge und der gewählten Sprache zur Darstellung der Anforderungen festgelegt.

Verbesserungen

In der Einleitung dieses Abschnitts ist der Begriff Optimierungspotenzial verwendet worden, es taucht aber kein entsprechender Artefakttyp im Produktmodell auf. Das liegt daran, dass die Optimierungspotenziale implizit vorhanden sind. Zum einen in den gültigen Situationsfaktoren, da diese eine Herausforderung darstellen (siehe Abschnitt 4.2). Zum anderen verstecken sich die direkten Optimierungspotenziale in der Beschreibung der Ziele, denn ein Ziel beschreibt die Lösung eines Optimierungspotenzials.

5.2.2 Verbesserungs-Analyse für das Anwendungsbeispiel der System-Architektur-Spezifikation

Dieser Abschnitt beschreibt die Ergebnisse der Verbesserungs-Analyse für das Anwendungsbeispiel der System-Architektur-Spezifikation mit Hilfe des für Meta-Me++ definierten Vorgehens (siehe Abschnitt 5.2.1).

Ergebnis System-Architektur-Spezifikation

Als Grundlage dienen die Ergebnisse der Ist-Analyse (Abschnitt 3.6.2) und der Situationsbeschreibung (Abschnitt 4.4.3). Aufbauend auf den Informationen wird das Ergebnis *Methodenentwicklungskontext* exemplarisch vorgestellt. Danach wird die Beschreibung der Ideal-Methode für die System-Architektur-Spezifikation vorgestellt. Es umfasst die Ziele, das Kontextmodell der Ideal-Methode, die Stakeholder, das Glossar sowie das Anforderungsmodell. Dieses Anwendungsbeispiel beschreibt die Ausprägung der gesamtheitlichen Planung der Methodenweiterentwicklung. Das bedeutet, die Ziele und die Anforderungen an eine Ideal-Methode sind über mehrere Iterationen der Methodenweiterentwicklung robust und ändern sich nicht.

Methodenentwicklungskontext

Für dieses Methodenweiterentwicklungsprojekt beschreiben die folgenden Situationsfaktoren Herausforderungen.

- Projektgröße: Die Projektgröße für die Methodenweiterentwicklung ist nicht definiert. Zu Beginn wird lediglich eine Vorabstudie zur Ermittlung von Schwächen und Herausforderungen in Auftrag gegeben. In dem Fall, dass Lösungen erkannt werden, die Probleme in der aktuellen Entwicklungsmethodik lösen können, werden weitere Aufträge zur Verbesserung bestehender Probleme beauftragt.
- Projektkritikalität: Die Projektkritikalität ist ausreichend hoch, so dass eine externe Partei, das s-lab – Software Quality Lab, unterstützende Hilfe anbieten soll.
- Projektpriorität: Aufgrund der Tatsache, dass der momentane Entwicklungsprozess einen großen Overhead an Fehlerbehebungsmaßnahmen hervorbringt, ist die Priorität hoch. Es soll der laufende Entwicklungsprozess angepasst werden.

- Ressourcenverfügbarkeit: Das s-lab – Software Quality Lab bietet eine Vollzeitkraft für die Methodenweiterentwicklung. FTS hat keine Ressourcen zur Verbesserung der Entwicklungsmethode. Es ist lediglich möglich Hilfestellung bei der Einarbeitung des s-lab –Software Quality Lab-Mitarbeiters zu leisten.
- Fertigkeiten: Bei FTS gibt es keine Kenntnisse zur Nutzung formalisierter Sprachen zur Verbesserung von Spezifikationsdokumenten.
- Auftraggeber- und Auftragnehmer-Beziehung: Das s-lab – Software Quality Lab tritt als Auftragnehmer im Methodenweiterentwicklungsprojekt auf. Es ist das erste Projekt dieser Art zwischen den beiden Parteien. Es gibt keine Hinweise auf die Art der Beziehungen der Parteien untereinander.
- Einführungsstrategie::Ersetzung: Verbesserungsmaßnahmen sollen sofort in das aktuelle Entwicklungsprojekt transferiert werden.
- Entwicklungsstrategie::Iterativ: Je nach Feststellung von Schwachstellen wird priorisiert, welche Schritte der Verbesserung angegangen werden sollen.
- Änderungswünsche: Auf Basis der Ergebnisse der schnellen Integration der Verbesserungen müssen jederzeit Änderungswünsche seitens FTS umgesetzt werden.
- Klarheit der Spezifikationsdokumente: Der zu verbessernde Entwicklungsprozess ist wenig bis gar nicht dokumentiert. Er liegt implizit vor.

Betrachten wir die definierten Situationsfaktoren für das Methodenweiterentwicklungsprojekt können wir erkennen, dass eine schlechte Planbarkeit für langfristige Lösungen besteht. Der Projektrahmen ist ungenau definiert. Das führt zu der Tatsache, dass kurze, vorteilsbringende Iterationen der Methodenweiterentwicklung definiert werden müssen. Weiterhin muss für jede Iteration die erarbeitete Verbesserung belegt werden.

Ziele

Die Ziele der Methodenweiterentwicklung werden aus den bestehenden Herausforderungen abgeleitet. Während die Herausforderungen, die in dem Abschnitt 3.1 vorgestellt worden sind, tatsächlich die Hauptprobleme für FTS darstellen, wird die Lösung der Herausforderungen durch grundlegende Schwachstellen erschwert (siehe auch Abschnitt 3.6.2). Die nächsten Absätze beschreiben zuerst

die vorliegenden Schwierigkeiten und danach exemplarisch auftretende Fehlersituationen.

Schwächen in den Spezifikationsinhalten

Die erste Aufgabe im Beratungsprojekt war die Untersuchung der Spezifikationsinhalte. Die Analyse der Spezifikationsdokumente lässt sich zusammenfassend mit der folgenden Tabelle darstellen:

Schwächen	**Risiken**
• Hauptsächlich informelle Spezifikationsinhalte • unvollständige Spezifikationsinhalte • technische Grundlagen werden vorausgesetzt • Inhalte sind teilweise sehr technikorientiert und lösungsbasiert beschrieben	• inkonsistente Inhalte • Missverständnisse • technische Zusammenhänge sind verborgen • schlechte Wiederverwendbarkeit

Dokumenten-Management

Es gibt keine einheitliche und robuste Dokumentenverwaltung. Viele Dokumente werden neu erstellt und auf viele bestehende Dokumente wird referenziert. Die Entwicklungsdokumente werden ständig erweitert und verbessert. Daraus resultieren viele unterschiedliche Dokumentversionen, bei denen Informationen hinzugefügt, verändert und verworfen werden. Die Dokumente werden beliebig verwaltet und an unterschiedlichen Stellen abgelegt. Ein Beispiel für einen Speicherort ist ein von FTS verwaltetes Netzlaufwerk, welches Terabytes an Daten und unzählbar viele Dokumente enthält. Es ist keine einheitliche Struktur erkennbar. Auch nicht innerhalb eines Projektordners des Netzlaufwerks. Die Ordnerstrukturen werden von jedem Benutzer beliebig, nach seinen Vorstellungen, erstellt. Es gibt keine definierte Arbeitsweise mit dem Netzlaufwerk. Es ist wichtig, Informationen schnell zu finden. Das ist schwierig, wenn Dokumentenbenutzer nicht wissen, wo welche Information zu finden ist. Alle Entwickler müssen auf denselben Informationen arbeiten. Es ist möglich, dass ein Entwickler mit einer bestimmten Dokumentversion arbeitet, die einem anderen Entwickler noch nicht bekannt ist. Die Informationen sollten konsistent und nicht redundant oder widersprüchlich sein. Bei einem fehlenden Überblick über alle Dokumente kann jedoch keine Überprüfung stattfinden. Meistens dienen vorhandene Dokumente als Design-

und Inhaltsvorlage. Oft sind die Anforderungen an die Dokumente nicht klar festgelegt. Auf diese Weise ist die Vollständigkeit der Dokumente nicht überprüfbar.

High-Level-Entwicklungsprozess

Ein definierter Entwicklungsprozess muss die Tätigkeiten zur Erstellung von Artefakten, die für die System-Entwicklung notwendig sind, definieren und somit konstruktiv bei der Entwicklung unterstützen. Der Entwicklungsprozess bei FTS weist große Freiheiten in der konkreten Ausführung desselben auf. Auf der Management-Ebene werden grobe Meilensteine definiert. Die Server-System-Entwicklung bei FTS ist in die Entwicklungsphasen *Analyse/Planung*, *Entwurf/Design* und *Realisierung* unterteilt. Auf der Entwickler-Ebene ist das Erreichen der Meilensteine durch ein selbstorganisierendes und Ad-hoc-Vorgehen geprägt. Auf diese Weise werden viele Erkenntnisse, die ein erfolgreiches Durchführen eines Entwicklungsprojekts ermöglichen, wenig bis gar nicht dokumentiert. Die Wiederholbarkeit von Entwicklungsaktivitäten ist nicht gewährleistet. Hauptsächlich wird auf das Wissen und die Erfahrung der Entwickler gesetzt. In der Softwareentwicklung typischerweise auftretende Artefakte wie Anforderungen, Anwendungsfälle, Entwurfs- und Analyse-Modelle sind nicht festgelegt.

unvollständige Anforderungen

Sämtliche Entwicklungstätigkeiten sind allgegenwärtig mit fehlenden Anforderungen konfrontiert. Das liegt nur zum geringen Teil an den sich während der Entwicklungszeit ändernden und kurzzeitig neu auftretenden Anforderungen sondern hauptsächlich an der vernachlässigten vollständigen Erhebung und Analyse der Anforderungen.

gehemmte Innovationskompetenz

Die Entwicklerkompetenzen sind stark durch die jahrelang festgelegten Arbeitsweisen und genutzten Technologien geprägt. Die Notwendigkeit bessere Dokumentationen und Spezifikationen zu erstellen, um den wachsenden Aufwand und die steigende Komplexität in den Griff zu bekommen, ist erst seit kurzer Zeit verstanden. Wissen über die Entwicklung domänenspezifischer Sprachen und formalisierter Inhalten sowie modellgetriebener Entwicklung ist wenig bis gar nicht verbreitet. In den letzten Jahren war die Entwicklung von Verwaltungsfunktionalität klar begrenzt auf die Programmierung eingebetteter Systeme. Erst durch den Einzug leistungsstarker Systeme in den Bereich der Systemverwaltung sind die Möglichkeiten zur Nutzung höherer Programmiersprachen möglich, deren Konzepte und Paradigmen, wie beispielsweise die Objektorientierung, noch nicht flächendeckend bekannt sind. Die wachsende Komplexität der Software macht es notwendig Softwarearchitekturen zu definieren. Vorhandene Software lässt historisch gewachsene Strukturen erkennen. Alles in allem fehlt verbreitetes Wis-

sen typischer Konzepte aus der Informatik, welches damit zu begründen ist, dass der größte Anteil der Entwickler eine Ausbildung im Bereich der Elektrotechnik vorweist. Die Schwierigkeit ist die Umsetzung einer umfassenden Weiterbildung, ohne Festlegung von Konzepten, Sprachen, Prozessen und darauf aufbauenden Werkzeugen.

Die Priorität beim Methodenweiterentwicklungsprojekt liegt in der Behebung grundlegender Schwächen, mit Blick auf die Hauptprobleme. Aus den Herausforderungen sind die folgenden Ziele abgeleitet worden. Die Ziele können in kurzfristige und langfristige Ziele gruppiert werden.

Kurzfristige Ziele:

- Verbesserung der Spezifikationsinhalte: Dieses Ziel soll erreicht werden, indem die Dokumentenstruktur durch Dokumentvorlagen vorgegeben wird. Der Verantwortliche eines Dokuments muss alle definierten Felder und Absätze des Dokuments berücksichtigen. Er ist gezwungen sich Gedanken über den geforderten Inhalt zu machen. Dadurch kann, bis zu einem gewissen Grad, die Vollständigkeit der Spezifikation konform zur Vorlage sichergestellt werden. Dabei sollen Dokumenten-Standards berücksichtigt werden.
- Dokumentenmanagement: Bei Schwächen im Bereich Dokumentenmanagement ist es sinnvoll ein Dokumenten-Management-System (DMS) einzuführen sowie der Aufbau der Dokumente und die Arbeit an den Dokumenten zu definieren. Ein DMS bietet die Möglichkeit der definierten Ablage von Dokumenten, meist in visualisierten Strukturen. Es bietet zusätzlich Werkzeugunterstützung zum Suchen und Finden von Informationen über Metadaten an und beschleunigt so diese Aufgabe. Ein Versionsmanagement ist in der Regel Bestandteil von gängigen DMS. Dadurch wird das Risiko des Datenverlusts durch Überschreiben oder versehentliches Löschen von Daten sehr gering. Professionelle DMS bieten weiterhin die Möglichkeiten der automatisierten Archivierung von Dokumenten, der Vorlagenverwaltung, der Automatisierung von Geschäftsprozessen, der Nutzung von Benutzerverwaltungs- und Rechtekonzepten sowie des verteilten Arbeitens an gemeinsamen Dokumenten.
- Definition der Abstraktionsebenen: Betrachten wir das Entwicklungsvorgehen V-Modell, wie es beispielsweise in [Boe79] vorgestellt wird, sehen wir unterschiedliche Abstraktions-Ebenen in der Entwicklung. Typischerweise beginnt man in diesem Vorgehen sehr abstrakt, beispielsweise mit der Anfor-

derungsdefinition an das Gesamtsystem. Schrittweise wird detailliert, möglicherweise über einen Systementwurf, Komponentenentwurf bis hin zu der technischen Lösung. Diese Ebenen sollen für FTS definiert werden.
- Detaillierung des methodischen Vorgehens: Hierbei steht die Definition von Entwicklungsschritten innerhalb der Phase *Analyse/Planung*, *Entwurf/Design* im Vordergrund.

Langfristige Ziele:

- Formelle Darstellungen: Die Probleme durch die informelle Darstellung sollen durch die Einführung geeigneter formeller Beschreibungssprachen ausgeräumt werden. Die Vorteile der Eindeutigkeit, Standardisierung, Modellunterstützung, Nutzung High-Level-Konzepten und leichte Erlernbarkeit sollen genutzt werden.
- Abstraktere Spezifikation der System-Architektur: Für die System-Architektur-Spezifikation soll das gesamte System mit seinen einzelnen Komponenten beschrieben werden. Hierbei ist die Beschreibung auf der konkreten technischen Ebene zu komplex und umfangreich. Eine modellbasierte Herangehensweise, passend zur komponentenbasierten Entwicklung, soll Systemkomponenten definieren. Aufbauend darauf sollen Schnittstellen definiert und Abhängigkeiten zwischen den Systemkomponenten ermittelt werden können. Der interne Aufbau des Systems soll dargestellt werden. Dies ist unabhängig von der Trennung mechanischer, elektrischer und Software-Aspekten. Hier ist es nicht notwendig genaue Schnittstellen zu definieren. Es sollen lediglich die expliziten Abhängigkeiten der Komponenten untereinander dargestellt werden. So wird bei dem weiteren Entwurf und der Verfeinerung direkt darauf Bezug genommen, ohne dass die Abhängigkeiten erst bei der Implementierung erkannt werden und Probleme mit sich bringen.
- Anforderungserhebung mit Anwendungsfällen: Die Aufgaben der Anforderungs-Ermittlung, -Formulierung, -Validierung und -Verwaltung sollen für FTS definiert werden.

Kontextmodell

Das Kontextmodell im Rahmen der Verbesserungs-Analyse ist schwierig zu definieren, da keine eindeutige Methode vorliegt. Betrachten wir die Ist-Methode aus dem Abschnitt 3.6.2. Wir reduzieren die Ist-Methode auf die relevanten Artefakte,

denn die Ausgabe-Artefakte einer Phase definieren die Schnittstellen der Phasen (siehe Abbildung: 5.9).

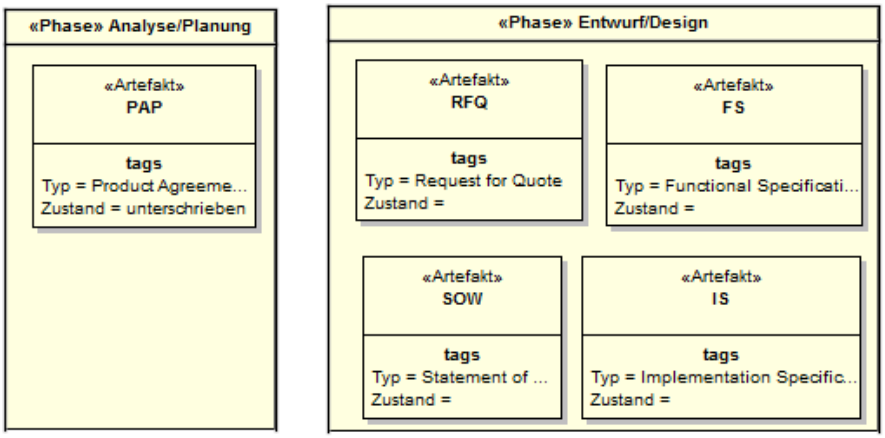


Abbildung 5.9: *Kontextmodell für das Anwendungsbeispiel der System-Architektur-Spezifikation*

In diesem Fall definieren wir die Phasen als Teilmethoden. Bei der Phase *Analyse/Planung* definiert die PAP (Product Agreement Paper) die Schnittstelle zu der Phase *Entwurf/Design*. Bei der Phase *Entwurf/Design* gibt es keine eindeutige Schnittstelle für die weiteren Phasen, da nicht klar ist, unter welchen Umständen welche Dokumente *Request-For-Quote*, *Statement-of-Work*, *Interface Specification* oder *Implementation Specification* erstellt werden (vergleiche mit Abschnitt 3.6.2). Wir können also keine festgelegten Schnittstellen der Phasen ermitteln, was eine weitere Schwachstelle der Ist-Methode darstellt.

Stakeholder

Die Stakeholder können zum Teil aus der Ist-Methode abgeleitet werden. Wir nehmen die definierten Rollen aus der Ist-Methode:

- Core Team
- Entwickler
- Produktmanager
- Produktverantwortlicher

Glossar

Aus Platzgründen wird das Glossar nicht dargestellt.

Anforderungsmodell

Wir haben die Grundlagen für die Anforderungserhebung in Form des Methodenentwicklungskontexts, der Ziele, des Kontextmodells und der Stakeholder in den vorangegangenen Absätzen definiert. Auf Basis dieser Informationen können wir die Anforderungen aus gültigen Situationsfaktoren und an notwendigen Methodenelementen ermitteln. Dabei werden die bisherigen Informationen zu Hilfe genommen und die ermittelten Stakeholder bei der Ermittlung und Bewertung der Anforderungen mit einbezogen. MetaMe++ schlägt die modellbasierte Ablage der Methodenanforderungen vor. Ein Beispiel für ein solches Anforderungsmodell wird in der Abbildung 5.10 für dieses Anwendungsbeispiel vorgestellt.

Aus Gründen der Übersicht ist in der Abbildung 5.10 auf Identifikatoren der einzelnen Anforderungen und der Darstellung der Prioritäten verzichtet worden. Lediglich die Kurzbeschreibung sowie die relevanten Abhängigkeiten der Anforderungen untereinander werden dargestellt. Die farbliche Unterscheidung der Abhängigkeiten definiert keine Semantik, sondern ist nur aus Gründen der Übersichtlichkeit verwendet worden. Wir erkennen allgemeine abstrakte Methodenanforderungen hauptsächlich auf der linken Seite der Abbildung. Umso weiter wir zur rechten Seite schauen, umso konkreter werden die Anforderungen. Die Kurzbeschreibung der Anforderung beschreibt, ob es sich um geforderte Hilfsmittel, Artefakttypen, Phasen oder Aktivitäten handelt.

Anforderungen

Inhaltlich sehen wir die Anforderung an eine System-Architektur-Spezifikation. Sie soll ein Konfigurationsmodell, ein Begriffsmodell, ein Datenmodell, Protokoll-Spezifikationen, Schnittstellen-Spezifikationen, System-Infrastrukturen, die Verwaltungsarchitektur, Beschreibungen der Systemkonzepte und Beschreibungen des selbst-adaptiven Verhaltens enthalten. Aus diesem Grund ist die Anforderung an den Artefakttyp *System-Architektur-Spezifikation* von weiteren Anforderungen an die anderen Artefakttypen abhängig. Es gibt eine Anforderung Abstraktionsebenen zu definieren. Daraus resultiert die Anforderung die Phasen *Analyse/Planung* und *Entwurf/Design* zu definieren. Wobei *Analyse/Planung* die

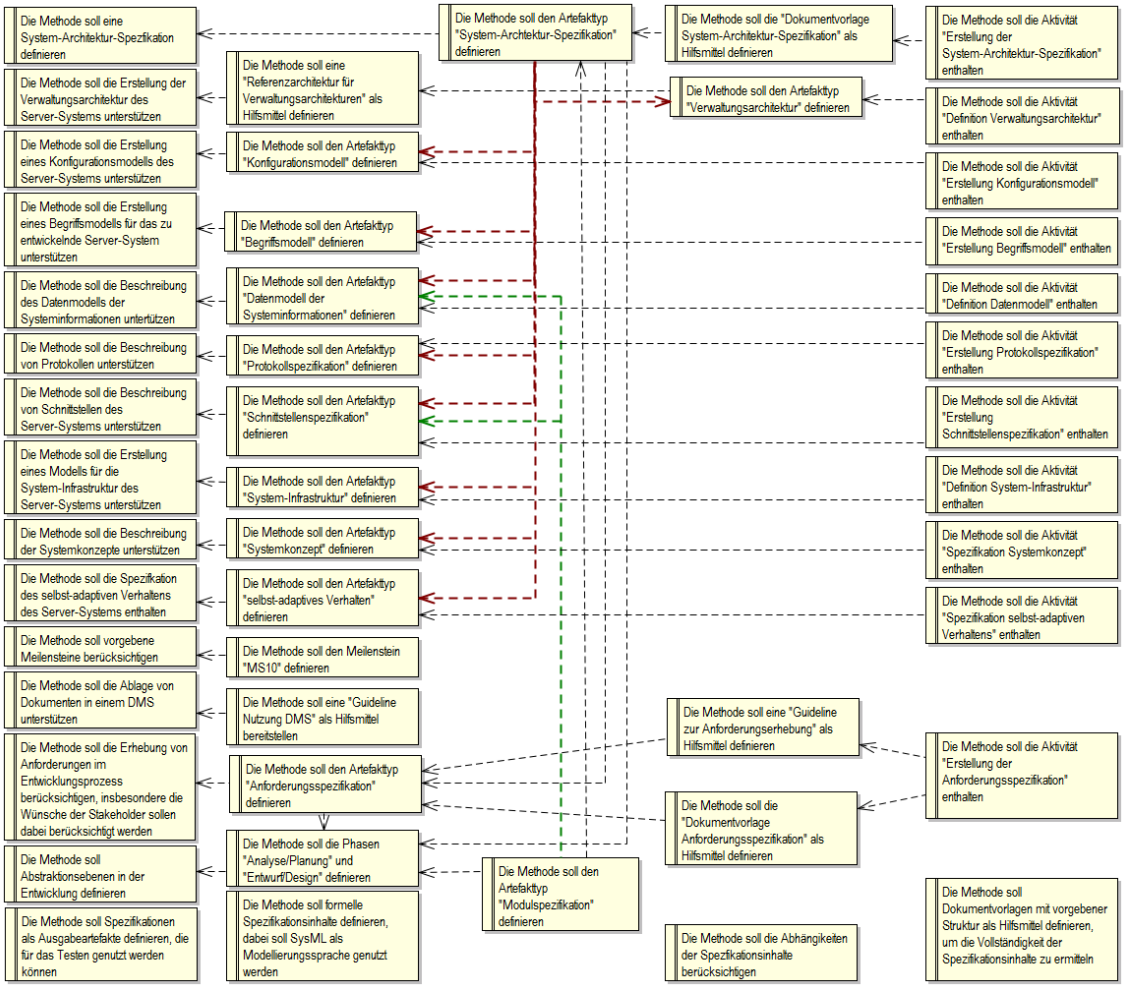


Abbildung 5.10: *Anforderungsmodell für das Anwendungsbeispiel der System-Architektur-Spezifikation*

abstrakteren Spezifikationsinhalte enthalten soll und *Entwurf/Design* die konkreteren. Die Anforderungsspezifikation und Systemarchitektur sollen die grundlegenden Spezifikationen für ein Gesamtsystem darstellen. Sie werden der Phase *Analyse/Planung* zugeordnet. In einem weiteren Konkretisierungsschritt sollen für definierte Elemente des Systems Modulspezifikationen erstellt werden, die weiter ins Detail gehen. Diese werden der Phase *Entwurf/Design* zugeordnet. Die Modulspezifikation soll eine Weiterentwicklung der System-Architektur-Spezifikation darstellen und dadurch ist die Anforderung an die Modulspezifikation abhängig von der Anforderung an die System-Architektur-Spezifikation, welche wiederum abhängig von der Anforderung an die Anforderungsspezifikation ist.

Abhängigkeiten der MA

Genereller Natur sind die Abhängigkeiten von Anforderungen an Aktivitäten und Anforderungen an Artefakttypen. Wenn eine Aktivität ein Artefakttyp instanziieren soll, bedeutet dies automatisch, dass die Anforderung daran, abhängig von der Anforderung an den entsprechenden Artefakttyp ist. Bei Anforderungen an Hilfsmittel verhält sich das ähnlich. Ist das Hilfsmittel zur Unterstützung einer Aktivität gedacht, so ist die Anforderung an das Hilfsmittel abhängig von der Anforderung an die Aktivität. Eine Besonderheit gibt es im Beispiel. Die Anforderung an eine Referenzarchitektur als Hilfsmittel muss betrachtet werden bevor die Anforderung an den Artefakttyp *Verwaltungsarchitektur* umgesetzt wird, da die Verwaltungsarchitektur auf der Referenzarchitektur basieren soll.

Eine Reihe weiterer Abhängigkeiten sind aus Gründen der Übersichtlichkeit nicht dargestellt. Beispielsweise sind alle Anforderungen an Artefakttypen abhängig von der Anforderung formeller Darstellung mit SysML, da fast jeder Artefakttyp Inhalte enthält, die mit SysML-Sprachelementen dargestellt werden können.

Projekte

Im Anforderungsmodell des vorangegangen Abschnitts erkennen wir eine Menge abhängiger Anforderungen an eine Ideal-Methode. Wie bereits in der Motivation vorgestellt, können nicht alle Anforderungen bearbeitet werden, da der Methodenentwicklungskontext dafür nicht die notwendigen Voraussetzungen an Ressourcen bereitstellen muss. Auf Basis des Anforderungsmodells werden Teilmengen der Anforderungen gebildet. In dem iterativen Forschungs- und Entwicklungsprojekt aus dem Anwendungsbeispiel sind verschiedene, aufeinander aufbauende Methodenweiterentwicklungsprojekte definiert worden.

Die Abbildung 5.11 gibt einen Überblick über die zu betrachtenden Anforderungen in dem ersten Projekt der Methodenweiterentwicklung. In erster Linie soll eine System-Architektur-Spezifikation erstellt werden. In der ersten Iteration soll die System-Architektur-Spezifikation als Dokument etabliert werden. Das Dokument soll während der Phase *Analyse/Planung* erstellt werden. Das fertige Dokument soll den Meilenstein *MS10* definieren. Die Inhalte sollen sich in erster Linie auf die System-Architektur, in Form der Definition der Systemmodule mit deren Schnittstellen untereinander und den externen System- und Benutzungsschnittstellen, fokussieren. Die geforderten Systemkonzepte sollen dargestellt werden. Die Dokumenteninhalte sollen soweit wie möglich durch eine for-

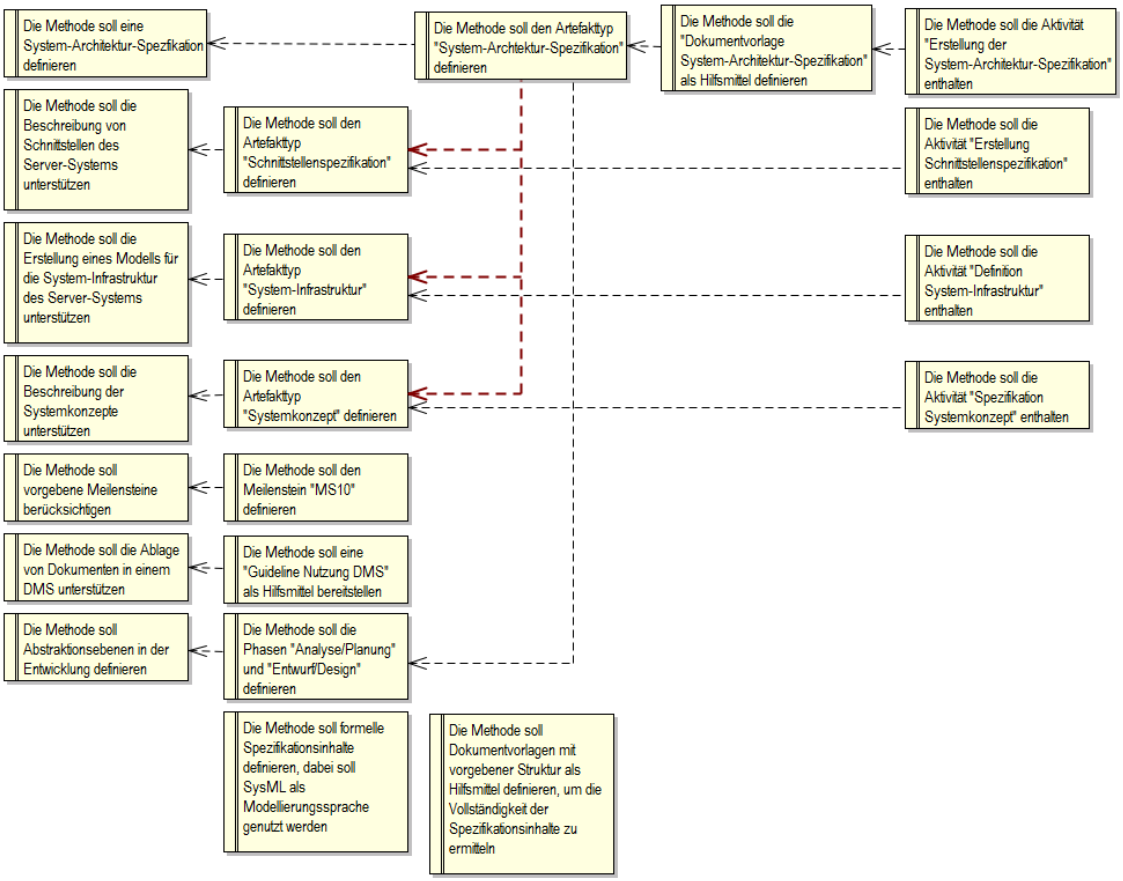


Abbildung 5.11: *Projekt 1 für das Anwendungsbeispiel der System-Architektur-Spezifikation*

melle Darstellung mit SysML-Diagrammen angereichert werden. Darüber hinaus soll eine Dokumentvorlage erstellt werden. In zweiter Linie soll ein Dokumenten-Management-System eingeführt und etabliert werden. Dafür soll die technische Infrastruktur und ein Nutzungskonzept erstellt werden.

Teilmengen

Wir erkennen, dass das erste Projekt eine Teilmenge der gesamten Anforderungen an die Ideal-Methode umfasst. Die Auswahl der Anforderungen für das erste Projekt ist in Zusammenarbeit mit dem Projektpartner und dessen Prioritäten definiert worden. Im Vergleich zum Anforderungsmodell an die Ideal-Methode erkennen wir, dass Anforderungen, die eigentlich von nicht berücksichtigten Anforderungen abhängig sind, im ersten Projekt definiert worden sind. Beispielsweise soll der Artefakttyp *System-Architektur-Spezifikation* ein Begriffsmodell, ein Datenmodell der Systeminformation, Protokollspezifikationen und die Beschreibung des selbst-adaptiven Verhaltens enthalten. Theoretisch müssen die Grundla-

gen geschaffen werden, damit eine ordentliche System-Architektur-Spezifikation möglich ist. Der Projektpartner hat jedoch dazu gedrängt, erst ein initiales Dokument zu erstellen, welches in weiteren Methodenweiterentwicklungsprojekten erweitert wird. Auf diese Weise ist es möglich das neue Dokument direkt in den aktuellen Entwicklungsprozess einzugliedern.

Projekte

Zur Übersicht wird in der folgenden Aufzählung vorgestellt, welche Themen/Anforderungen in den weiteren Methodenentwicklungsprojekten bearbeitet worden sind.

Projekt2 Das zentrale Thema in dem zweiten Projekt waren die Anforderungen an ein neues System. Damit einhergehend muss eine Anforderungsspezifikation mit Dokumentvorlagen sowie eine Vorgehensbeschreibung zur Erhebung der Anforderungen erstellt werden. Die Tätigkeiten und Dokumente sollen zu den Ergebnissen des ersten Projekts passen und ebenfalls der Phase *Analyse/Planung* zugeordnet werden. Ein weiterer Punkt war die Definition und Beschreibung der Verwaltungsarchitektur. Die System-Architektur-Spezifikation aus dem ersten Methodenweiterentwicklungsprojekt muss um die Beschreibung der Verwaltungsarchitektur erweitert werden. Letztendlich müssen auf Basis der System-Architektur-Spezifikation weitergehende Spezifikationsdokumente erstellt werden. Diese Dokumente werden durch den Artefakttyp *Modulspezifikation* beschrieben. Dessen Eingliederung in die Entwicklungsmethodik des ersten Projekts schließt das zweite Projekt ab.

Projekt3 Nachdem im zweiten Methodenweiterentwicklungsprojekt eine Verwaltungsarchitektur definiert worden ist, sollen diese Erkenntnisse in Form einer Referenzarchitektur festgehalten werden. Die Erkenntnisse über eine Referenzarchitektur sollten als Hilfsmittel in der entwickelten Entwicklungsmethode aus den ersten beiden Projekten berücksichtigt werden.

Iterationen

Aus der Definition der drei Projekte ist zu erkennen, dass es auf Basis des Anforderungsmodells möglich ist iterative Methodenweiterentwicklungsprojekte zu definieren, die einem Gesamtziel, den Anforderungen an die Ideal-Methode, folgen.

Einschränkungen

Die Definition der Methodenweiterentwicklungsprojekte ist stets streng an aktuelle Entwicklungsaufgaben bei FTS gekoppelt worden. Daher gibt es auch ein paar Anforderungen, deren Umsetzung nicht vollständig durchgeführt werden

konnte. Die Themen Protokollspezifikation, Begriffsmodelle der Domäne, Datenmodell der Systeminformationen und die Beschreibung selbst-Adaptives Verhalten der Server-Systeme sind nicht im Zusammenhang mit der System-Architektur-Spezifikation als Methodenweiterentwicklungsprojekt bearbeitet worden.

Themenverschiebung

Während der Durchführung der drei Methodenweiterentwicklungsprojekte hat eine Verschiebung vom Thema der vollständigen System-Architektur-Spezifikation in eine integrative Entwicklungsmethodik auf die Entwicklung der Verwaltungsarchitektur mit dem Common Information Model stattgefunden. An dieser Stelle wird auf das zweite Anwendungsbeispiel der CIM-Entwicklung verwiesen, das die Themen Datenmodell der Systeminformationen erneut aufgreift.

Für eine Übersicht über das Thema Spezifikation selbst-adaptiven Verhaltens kann auf die Dissertation [Luc13] verwiesen werden.

5.2.3 Verbesserungs-Analyse für das Anwendungsbeispiel der CIM-Entwicklung

Dieser Abschnitt beschreibt die Ergebnisse der Verbesserungs-Analyse für das Anwendungsbeispiel der CIM-Entwicklung mit Hilfe des für MetaMe++ definierten Vorgehens (siehe Abschnitt 5.2.1).

Als Grundlage dienen die Ergebnisse der Ist-Analyse (Abschnitt 3.6.2) und der Situationsbeschreibung (Abschnitt 4.4.3). Weiterhin sind die Grundlagen des CIMs aus dem Abschnitt 3.3.3 zu berücksichtigen.

Der Methodenentwicklungskontext ist ähnlich zum vorangegangenen Anwendungsfall der System-Architektur-Spezifikation (siehe Abschnitt 5.2.2) und wird deshalb nicht erneut erläutert.

Die Beschreibung der Ideal-Methode mit den Zielen, Kontextmodell, Stakeholder, Glossar und Anforderungsmodell wird vorgestellt.

In diesem Anwendungsbeispiel wird beschrieben, wie bei der iterativen Methodenweiterentwicklung nach einer Iteration neue Methodenanforderungen für die folgende Iteration integriert werden können.

Ziele

Die folgenden Absätze fassen die vorliegenden Herausforderungen aus der Ist-Analyse der CIM-Entwicklung (siehe Abschnitt 3.6.3) zusammen und stellen die daraus resultierenden Ziele vor.

Erstellung CIM-Provider

Die identifizierte Ist-Methode beschreibt hauptsächlich die Aufgabe CIM-Provider zu erstellen. Dafür muss das CIM-Schema erweitert werden, indem das OEM-spezifische Erweiterungs-Schema (FTS-Schema) definiert wird. Das FTS-Schema definiert die strukturellen Eigenschaften des CIMs. Diese werden in MOF-Dateien hinterlegt, die in einer standardisierten Grammatik definiert werden. Ein CIM-Provider stellt die Systeminformationen unter Vorgabe der strukturellen Informationen bereit. Dafür muss das Verhalten der CIM-Provider definiert werden. Das geschieht mittels der OEM-spezifischen CIM-Profildokumente (FTS-Profildokumente).

Herausforderungen

Eine Zusammenfassung der vorliegenden Herausforderungen bei der CIM-Entwicklung werden in der folgenden Aufzählung vorgestellt.

Fehlendes Know-how: Die Anforderung CIM zu unterstützen stellt FTS vor neue Herausforderungen. Das heißt, es gibt kein Vorwissen bezüglich Spezifikation und Umsetzung des CIM-Konzepts. Des Weiteren ist ein großer Einarbeitungsaufwand notwendig, der nicht nur auf der Einarbeitung in das CIM-Konzept basiert, sondern es werden auch Vorkenntnisse im Bereich Objekt-Orientierung benötigt. Das Konzept der Objekt-Orientierung ist für die Entwickler teilweise neu. Das bedeutet, es sind Kenntnisse der UML Voraussetzung, da beispielsweise sämtliche Profile mit Unterstützung von UML-Klassendiagrammen beschrieben werden.

Entwicklungsaufwand: Initial sollen ca. 20 CIM-Profile unterstützt werden. Darunter fallen eine Vielzahl von CIM-Klassen. Für jede CIM-Klasse wird ein CIM-Provider erstellt. Das heißt, eine große Anzahl an CIM-Providern muss spezifiziert, implementiert und getested werden.

Abhängigkeiten des CIMs: Die Entwicklung eines großen Datenmodells mit impliziten und expliziten Abhängigkeiten der einzelnen Modellelemente muss durchgeführt werden. Es gibt Abhängigkeiten der CIM-Klassen untereinander. Die Strukturierung des CIM-Schemas in CIM-Profile bedeutet nicht, dass jedes CIM-Profil entkoppelt von weiteren Profile ist, sondern es bestehen Abhängigkeiten zwischen CIM-Klassen, die unterschiedlichen CIM-

Profilen zugeordnet werden. Bei der Bereitstellung eines CIM-Profils für ein Server-System muss immer eine gesamtheitliche Analyse über das gesamte CIM-Schema und aller abhängigen CIM-Profile durchgeführt werden.

manueller Aufwand: Die Erstellung der MOF-Dateien und der FTS-Profildokumente muss aufgrund fehlender Werkzeugunterstützung manuell durchgeführt werden. Gepaart mit dem Umfang dieser Aufgabe wird eine hohes Fehlerrisiko bei den manuellen Tätigkeiten erwartet.

konsistente Beschreibung der strukturellen Informationen des CIM-Schemas: Ein FTS-Profildokument beschreibt neben den funktionalen Eigenschaften auch strukturelle Eigenschaften der für das Profil relevanten CIM-Klassen. Genau diese Strukturinformationen werden auch in den MOF-Dateien für eine einzelne CIM-Klasse definiert wird (vgl. 3.22). Das bedeutet, es entstehen zwei Ressourcen mit den gleichen Informationen in einer unterschiedlichen Darstellung. Die Aufgabe besteht darin beide Ressourcen konsistent zu halten, was durch den Umfang und die fehlende Werkzeugunterstützung eine weitere Schwierigkeit darstellt.

fehlende Werkzeugunterstützung: Aufgrund der wenigen Kenntnisse über die CIM-Entwicklung gibt es kein Wissen über die Verfügbarkeit von Entwicklungswerkzeugen und Testwerkzeugen.

fehlendes Testkonzept: Bei der Realisierung des CIMs für Server-Systeme besteht letztendlich die Frage, welche Funktionalitäten getestet werden müssen. In erster Linie werden CIM-Provider implementiert. Sie repräsentieren die funktionalen Elemente des CIMs. Das heißt, die vom CIM-Provider implementierten Funktionen müssen getestet werden. Darüber hinaus interagieren die CIM-Provider mit den hardwarenahen Systemschnittstellen und mit dem CIMOM. Anfragen an den CIMOM werden über die CIM-basierten Schnittstellen mit verschiedenen Protokollen gestellt. Es ist nicht klar, welche Elemente der Architektur getestet werden müssen.

Artefakttypen

Wir kennen die erwarteten Artefakttypen aus der Ist-Methode und müssen nach Techniken und Sprachen suchen, die die Erstellung der Artefakttypen unterstützen. Darauf aufbauend können Werkzeuge gesucht oder entwickelt werden, die bei der Anwendung der Techniken und Sprachen behilflich sind.

modellbasierte Entwicklung

Die angedachte Lösung für die genannten Herausforderungen liegt in der modellbasierten Entwicklung des CIMs.

kurzfristig

Das kurzfristige Ziel ist die modellbasierte Spezifikation der strukturellen Infor-

mationen. Das heißt, die OEM-spezifischen Erweiterungen des CIM-Schemas, das FTS-Schema, sollen modellbasiert spezifiziert werden. Ein Modell der Strukturinformationen soll entwickelt werden.

Modell

Das Modell des FTS-Schemas stellt über Integritätsregeln die Wohlgeformtheit der Spezifikationsinhalte sicher. Dadurch wird die Herausforderung der Abhängigkeiten gemeistert. Mit Generierungsmechanismen können weitere Artefakte, wie die FTS-Profildokumente und die MOF-Dateien, erstellt werden und das Problem der Inkonsistenzen wird vermieden. Weiterhin wird durch die Generierung der MOF-Dateien und der strukturellen Informationen des FTS-Schemas der manuelle Aufwand und das Fehlerrisiko minimiert.

Verfeinerung

Die Lösung wird konkretisiert durch die Tatsache, dass das CIM-Schema über das CIM-Metaschema (CIM-Metamodell) definiert wird. Das CIM-Metaschema weist Ähnlichkeiten zu dem UML-Metamodell auf. In der Spezifikation des Standards von der DMTF werden bereits UML-Klassendiagramme für die Darstellung des CIM-Schemas genutzt. Es ist möglich die Elemente des CIM-Schemas auf UML-Elemente abzubilden. Aufgrund der für die UML entwickelten Erweiterungsmöglichkeit des UML-Profilmechanismus, ist es möglich ein UML-Profil für CIM zu definieren. Ein UML-Profil stellt eine Erweiterung der UML dar. Das ist notwendig, denn es gibt CIM-Spezifika, die standardmäßig nicht durch die UML dargestellt werden, aber notwendig für die CIM-Schema-Spezifikation sind.

Werkzeuge

Weiterhin ist es möglich UML-Modellierungswerkzeuge für die Spezifikation des FTS-Schemas zu benutzen. UML-Modellierungswerkzeuge unterstützen neben der Erstellung des Modells auch die Erstellung von Integritätsregeln (bspw. über OCL), den Austausch von Modellinformationen (bspw. XMI) und verschiedene Möglichkeiten zum Zugriff und für die Bearbeitung der Modellinhalte (bspw. Generierungsmechanismen). Auf diese Weise wird es realisierbar ein definiertes Modell der strukturellen Informationen, das über Integritätsregeln die Wohlgeformtheit der Spezifikationsinhalte sicherstellt und mit Generierungsmechanismen die automatisierte Überführung der Modellinhalte in verschieden Darstellungsformen ermöglicht, Werkzeug-unterstützt zu erstellen.

langfristig

Das langfristige Ziel ist der Ausbau der modellbasierten Spezifikation zur vollständigen CIM-Spezifikation aller relevanten Informationen und nicht nur der strukturellen Inhalte. Der Vorteil liegt in der zentralen Spezifikation in dem Modell und weiterer Werkzeug-gestützter Möglichkeiten der Automatisierung. Eine

vollständige Generierung aller Entwicklungsartefakte und der Testfälle, konform zu einem Testkonzept, ist die Vision.

Die resultierenden Ziele an die Entwicklungsmethode werden im Folgenden definiert. Ziele

- Vollständige Abbildung des gesamten Entwicklungsprozesses konform zu den von FTS angedachten Entwicklungsphasen *Analyse/Planung*, *Entwurf/Design*, *Realisierung* und *Qualitätsicherung*
- Modellbasierte Spezifikation des CIMs, um die Beherrschbarkeit der Abhängigkeiten sicherstellen zu können
- Reduzierung des Entwicklungsaufwands durch Automatisierung sich wiederholender Spezifikationsaufgaben
- Nutzung von Generierungsmechanismen für die Artefakte der CIM-Entwicklung zur Reduzierung des manuellen Aufwands
- Definition eines Testkonzepts als Hilfsmittel bei der Qualitätssicherung

Kontextmodell

Wie auch bei der Erstellung des Kontextmodells im Anwendungsbeispiel der System-Architektur-Spezifikation betrachten wir die Ist-Methode und ermitteln die Artefakte, die als Ein- und Ausgabe-Artefakte agieren als die Schnittstellen der Methode. Die Ist-Methode enthält noch keine Definition der Entwicklungsphasen und Zuweisung der Aktivitäten zu den Entwicklungsphasen. Darum wird das Kontextmodell ebenfalls um die Entwicklungsphasen erweitert. Das Resultat wird in Abbildung 5.12 vorgestellt.

Das Artefakt DSP-Dokument wird als Ausgabe der Phase *Analyse/Planung* definiert. Es stellt ebenfalls das Eingabe-Element für die Phase *Entwurf/Design* dar. Auf Basis der Informationen des DSP-Dokuments wird zum einem das SVS-Schema und das SVS-Dokument erstellt. Beide Artefakte sind die Voraussetzung für die Programmierung der zum Profil angehörigen Provider in der Phase *Realisierung*. Die Phase *Qualitätssicherung* wird aufgrund der Ziele ebenfalls hinzugefügt. Die Ist-Methode beschreibt allerdings noch keine entsprechenden Artefakte. Das Artefakt *Testfall* wird jedoch als Platzhalter im Kontextmodell definiert. Artefakte

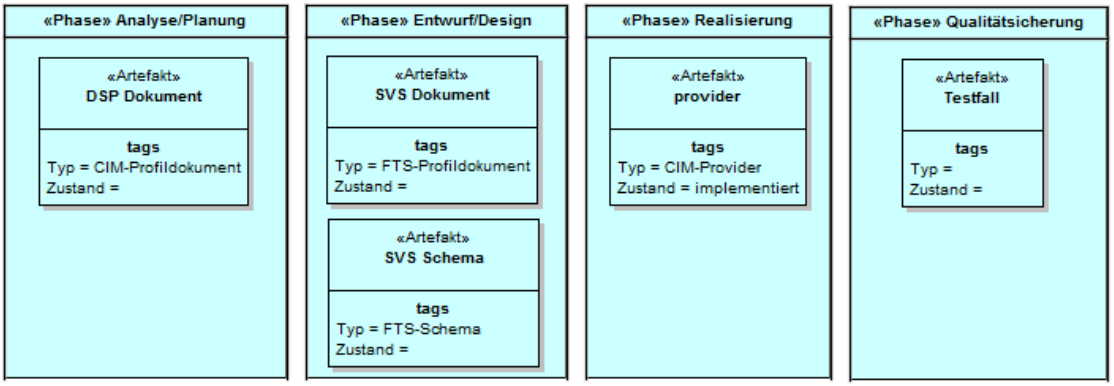


Abbildung 5.12: *Kontextmodell für das Anwendungsbeispiel CIM-Entwicklung*

Stakeholder

Die Ist-Methode definiert noch keine Rollen, so dass wir die Rollen nicht als Stakeholder für diese Verbesserungs-Analyse ableiten können. Da es sich bei der Methodenweiterentwicklung in diesem Anwendungsbeispiel hauptsächlich um die Entwicklungsaktivitäten handelt, werden hauptsächlich die Entwickler als Stakeholder definiert. Die Produktverantwortlichen spielen allerdings auch eine Rolle. Sie verwalten die Ressourcen für die Projekte der Realisierung der Server-System-Verwaltung mit CIM. Das bedeutet, sie sind daran interessiert die Ressourcen Zeit und Kosten abschätzen zu können sowie die Qualität hoch zu halten. Die Ziele der Entwicklungsmethode enthalten neben der Spezifikation der Methode ebenfalls Ressourcen-Aufwände, da Werkzeuge und Techniken entwickelt werden müssen. Diese Aufwände müssen dem Methodenweiterentwicklungsprojekt zugewiesen werden, werden aber dem Projektverantwortlichen in Rechnung gestellt, da er die Vorteile für die zukünftigen Entwicklungsaufgaben nutzen kann. Das bedeutet, der Realisierungsaufwand der Methodenweiterentwicklung wird mit dem sonstigen Entwicklungsaufwand gegen gerechnet. Letztendlich spielen bei der Qualitätssicherung die Tester auch eine Rolle. Sie müssen die Testfälle erstellen und durchführen. Deren Möglichkeiten liegen im Rahmen der vorhandenen Spezifikationsinhalte, die sie für die Erstellung der Testfälle nutzen müssen. Demnach können auch die Tester Anforderungen an die Entwicklungsmethode stellen.

Die Stakeholder der Methodenweiterentwicklung sind:

- Entwickler

- Produktverantwortliche
- Tester

Glossar

Aus Platzgründen wird das Glossar nicht dargestellt.

Anforderungsmodell

Wir haben die Grundlagen für die Anforderungserhebung in Form des Methodenentwicklungskontexts, der Ziele, des Kontextmodells und der Stakeholder in den vorangegangenen Absätzen definiert. Auf Basis dieser Information können wir die Anforderungen aus gültigen Situationsfaktoren und an notwendigen Methodenelementen ermitteln. Dabei werden die bisherigen Informationen zu Hilfe genommen und die ermittelten Stakeholder bei der Ermittlung und Bewertung der Anforderungen mit einbezogen. Das resultierende Anforderungsmodell für dieses Anwendungsbeispiel wird in Abbildung 5.13 vorgestellt.

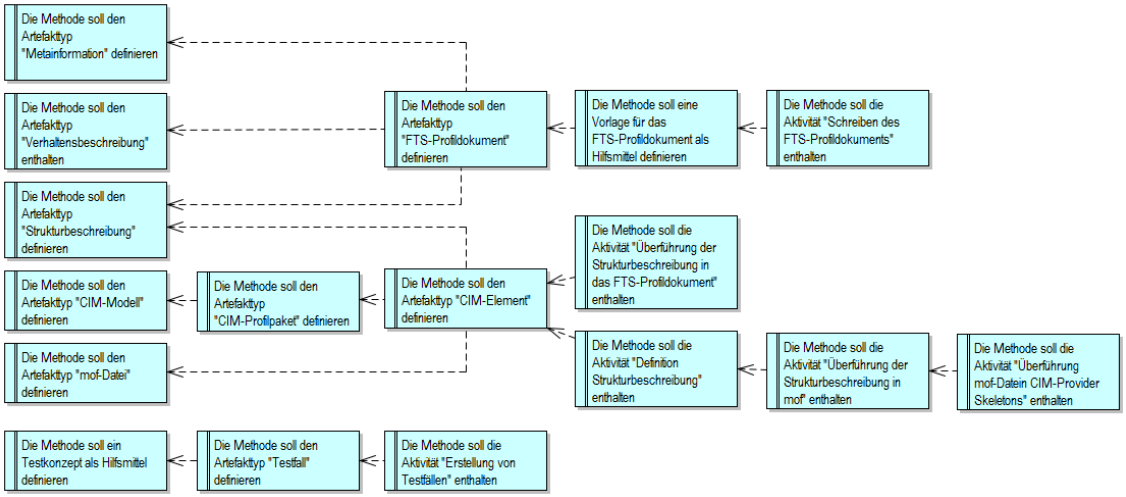


Abbildung 5.13: *Anforderungsmodell für das Anwendungsbeispiel der CIM-Entwicklung*

Themen

In einer ersten Analyse, wie ein FTS-Profildokument auf Basis der Vorgaben der DMTF aussehen kann, konnten drei verschiedene inhaltliche Themen identifiziert werden. Erstens sind Metainformationen enthalten. Das sind CIM-spezifische In-

formationen, die die Inhalte eines CIM-Profils, deren Verwaltungsinformationen und Abhängigkeiten zu weiteren CIM-Profilen definieren. Beispielsweise sind das deskriptive Beschreibungen des Profils, der Dokumentreferenzen, der Glossareinträge, der Versionierung des Dokuments und weitere Informationen. Zweitens definiert die Verhaltensbeschreibung die Art und Weise, wie die CIM-Provider ihre Systeminformationen ermitteln und darstellen und das auf Basis der vorgegebenen Struktur des CIM-Schemas, welches den dritten inhaltlichen Aspekt darstellt. In ihrer Gesamtheit stellen diese drei geforderten Artefakttypen den Artefakttyp *FTS-Profildokument* dar. Sie sind jedoch aufgrund ihrer unterschiedlichen Aufgaben durch separate Anforderungen beschrieben. Beispielsweise muss der Artefakttyp *Strukturbeschreibung* im FTS-Profildokument, aber auch im geforderten CIM-Modell berücksichtigt werden. Für die zu erstellenden FTS-Profildokumente sollen Vorlagen und Handlungsanweisungen zur Erstellung der Dokumente erstellt werden.

CIM-Modell

Das CIM-Modell wird ebenfalls als Artefakttyp gefordert. Aufgrund der logischen Struktur der CIM-Profile soll es strukturell in CIM-Profilpakete unterteilt werden, die wiederum eine Menge an Artefakten vom Typ *CIM-Element* enthalten. Ein CIM-Element repräsentiert CIM-Assoziation oder CIM-Klassen im CIM-Modell. Das CIM-Element soll die strukturellen Informationen für einen Provider enthalten.

Projekte

Die Anforderungen im Modell sind in zwei Teilmengen unterteilt worden, auf deren Basis zwei iterative Methodenweiterentwicklungsprojekte definiert werden konnten (siehe Abbildung 5.14).

5.2.4 Iterative Verbesserungs-Analyse für das Anwendungsbeispiel der CIM-Entwicklung

In dem Beispiel des vorangegangenen Abschnitts ist die Verbesserungs-Analyse der CIM-Entwicklung mit dem Ergebnis zweier definierter Projekte vorgestellt worden. In diesem Abschnitt soll die iterative Methodenweiterentwicklung verdeutlicht werden.

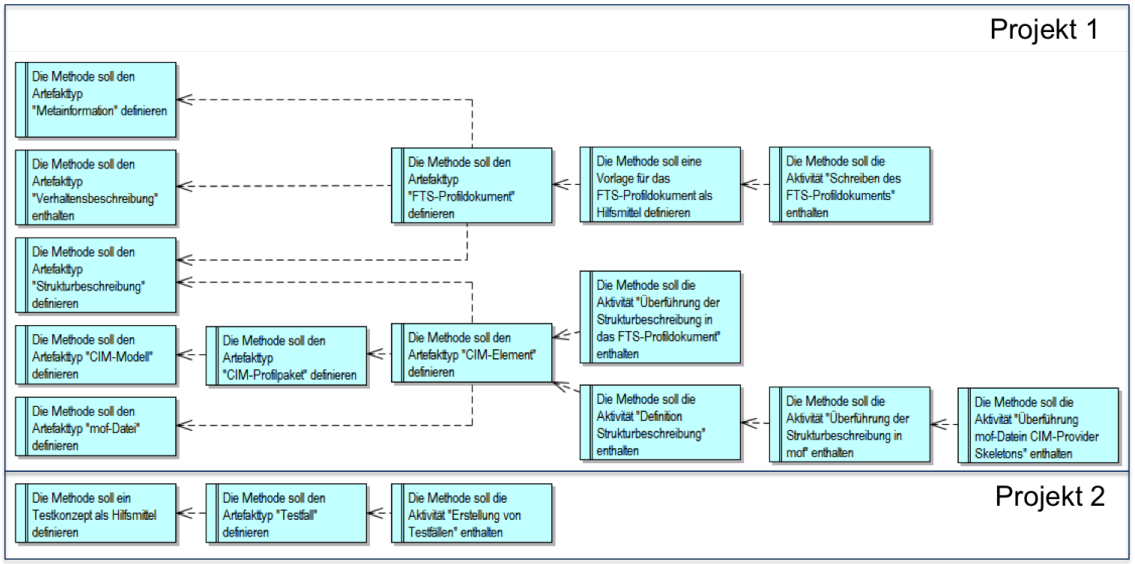


Abbildung 5.14: *Initiale Definition zweier Methodenweiterentwicklungsprojekte*

Ausgangslage

Wir nehmen an, dass das erste Projekt erfolgreich durchgeführt worden ist. Das heißt, die MetaMe++-Phasen *Konzeption* und *Realisierung* sind für das Projekt 1 durchgeführt worden. Wir haben eine Soll-Methode im Methodenweiterentwicklungsprojekt *Projekt 1* erstellt, die in der nächsten Iteration der Methodenweiterentwicklung als Ist-Methode angenommen wird. Die Schritte, die für die Methodenweiterentwicklung in der Ist-Analyse und Verbesserungs-Analyse definiert sind, sind nur zu wiederholen, wenn sich der Methodenkontext oder die Ziele der Methodenweiterentwicklung ändern. Ansonsten wird das Projekt 2 mit seinen Methodenanforderungen, wie im Anforderungsmodell definiert (siehe Abbildung 5.14), auf der neuen Ist-Methode durchgeführt.

Änderung des Kontexts

In dem Anwendungsbeispiel der CIM-Entwicklung bei FTS hat sich jedoch nach der ersten Iteration der Methodenkontext verändert. Aufgrund der Herausforderung durch den großen Entwicklungsumfang hat FTS Unterstützung bei der CIM-Entwicklung bei einer Zuliefererfirma eingekauft. Deren Aufgabe ist es die grundlegenden erforderten CIM-Profile zu spezifizieren und entsprechende CIM-Provider bereitzustellen. Aufbauend auf den bereitgestellten CIM-Providern sollen dann nur OEM-spezifische Anpassungen an den CIM-Providern vorgenommen werden. Dafür wird von der Zuliefererfirma ein Entwicklungsframework mit entsprechenden Erweiterungs- und Anpassungsschnittstellen für die CIM-Provider bereitgestellt. Das bedeutet, bei der Erstellung der FTS-Profildokumente müssen zusätzlich die von der Zuliefererfirma definierten Festlegungen berücksich-

tigt werden. Die Aufgabe des Schreibens der FTS-Profildokumente muss an die neue Situation angepasst werden. Zusätzlich muss für die Entwickler eine Guideline für die Benutzung des Entwicklungsframeworks erstellt werden. Diese Änderungen im Entwicklungsprozess sollen in der Entwicklungsmethode berücksichtigt werden.

Darüber hinaus sollen die Anforderungen an die CIM-Modellierung, also die Anforderungen an die CIM-Provider, besser dokumentiert und verwaltet werden. Es ist festgestellt worden, dass die Anforderung ein bestimmtes CIM-Profil zu realisieren nicht ausreicht. Es muss detaillierter dokumentiert werden können, welche Eigenschaften der CIM-Provider zu unterstützen sind und welche Eigenschaften der bereitgestellten CIM-Provider OEM-spezifisch angepasst und erweitert werden müssen. Die Anforderungsdefinition und Verwaltung soll ebenfalls in der CIM-Entwicklungsmethode berücksichtigt werden.

Schließlich sind die Vorteile der modellbasierten Spezifikation des CIMs erkannt worden und sollen weiter ausgebaut werden. Nicht nur die strukturellen Eigenschaften sollen in dem CIM-Modell spezifiziert werden, sondern auch die Metainformationen und die Verhaltensbeschreibung. Das soll langfristig die Möglichkeit bereitstellen die FTS-Profildokumente sowie mögliche Testfälle vollständig zu generieren. Dadurch wird der manuelle Spezifikationsaufwand reduziert und die Qualität verbessert, indem die Qualität der Spezifikation auf Modell-Ebene gesichert wird.

Iteration 1

Aus Gründen der Übersicht wird lediglich die Soll-Methode des Projekts 1, die gleichzeitig die Ist-Methode für die folgenden Projekte darstellt, skizziert. Die einzelnen Schritte der Erstellung der Ist-Methode sind bereits in dem Abschnitt 3.6 definiert und anhand von Anwendungsbeispielen beschrieben worden. Die Schritte sind analog durchgeführt worden und das Resultat ist das Produkt- und Prozessmodell der neuen Ist-Methode.

Iteration 2

Das Produktmodell der zweiten Iteration wird in Abbildung 5.15 dargestellt. Es enthält die neuen Artefakttypen CIM-Modell, CIM-Profilpaket und CIM-Element. Das CIM-Modell stellt ein Modell des UML-Profils für CIM dar. Es beschreibt das CIM-Schema (FTS-Schema) und deshalb stellt es eine fachliche Verfeinerung dar. Das CIM-Modell enthält die CIM-Profilpakete. Ein CIM-Profilpaket strukturiert das CIM-Modell bezüglich der logischen Strukturen der CIM-Profile. Es wird definiert über das Sprachelement *ExtensionSchema*, welches das UML-Profilelement

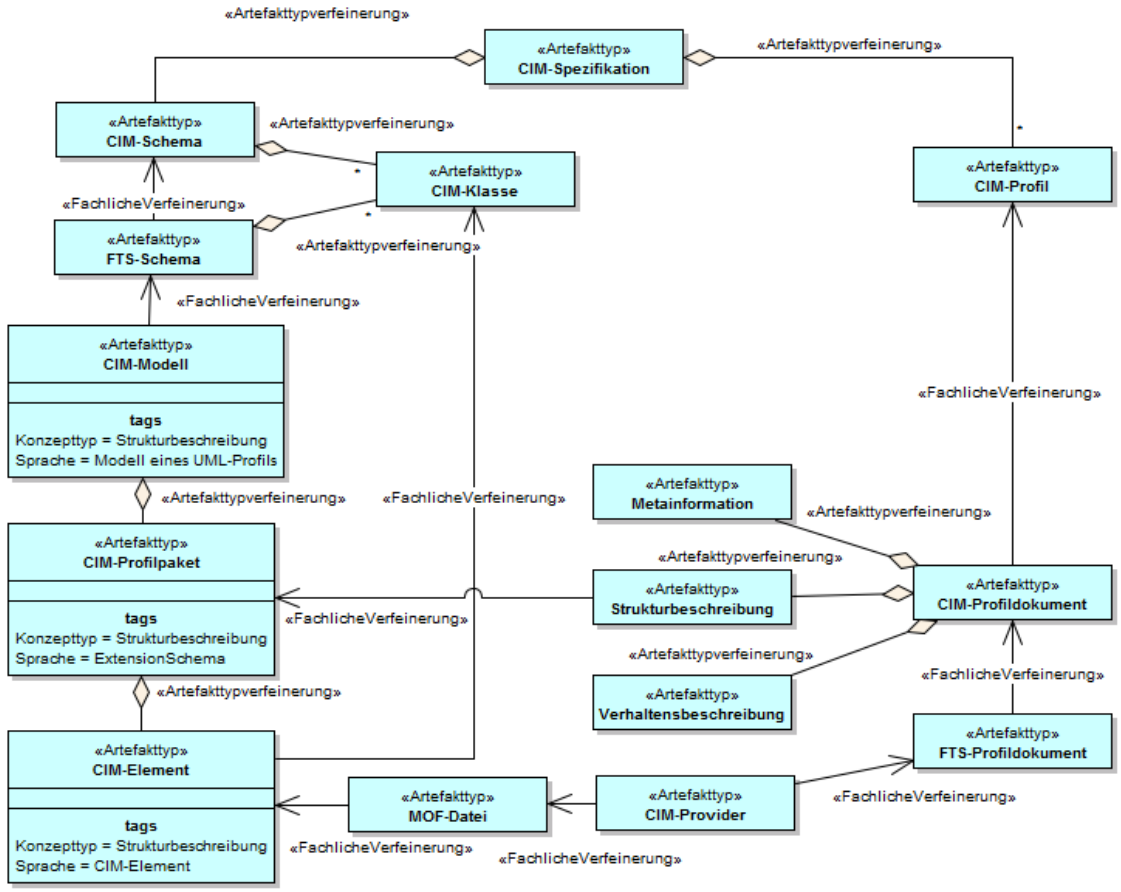


Abbildung 5.15: *Produktmodell der Ist-Methode der CIM-Entwicklung – Iteration 2*

für ein logisches CIM-Profil darstellt. Es aggregiert die CIM-Elemente und repräsentiert die strukturellen Information des CIMs. Diese Strukturbeschreibung wird in das FTS-Profildokument überführt und darum wird die Strukturbeschreibung des FTS-Profildokuments als fachliche Verfeinerung des CIM-Profilpakets angesehen. Das CIM-Element ist ein Element des CIM-Modells und wird einem CIM-Profilpaket zugeordnet. Es wird beschrieben über das UML-Profilelement *CIM-Element*. Es definiert alle für die MOF-Dateien relevanten Informationen und wird in einem Generierungsschritt in die MOF-Datei-Repräsentation überführt. Deshalb stellt die MOF-Datei eine fachliche Verfeinerung für das CIM-Element dar. Aus dem MOF-Dateien können Provider-Skeletons generiert werden, die einen CIM-Provider, konform zu der MOF-Datei, darstellen. Ein CIM-Provider stellt deswegen eine fachliche Verfeinerung der MOF-Datei dar. Ein CIM-Provider muss um die Verhaltensaspekte erweitert werden, die im FTS-Profildokument de-

finiert werden. Dieser Aspekt wird durch die fachliche Verfeinerung des FTS-Profildokuments dargestellt.

Die definierten Aktivitäten, im Sinne des Entwicklungsprozesses, werden in der Abbildung 5.16 veranschaulicht.

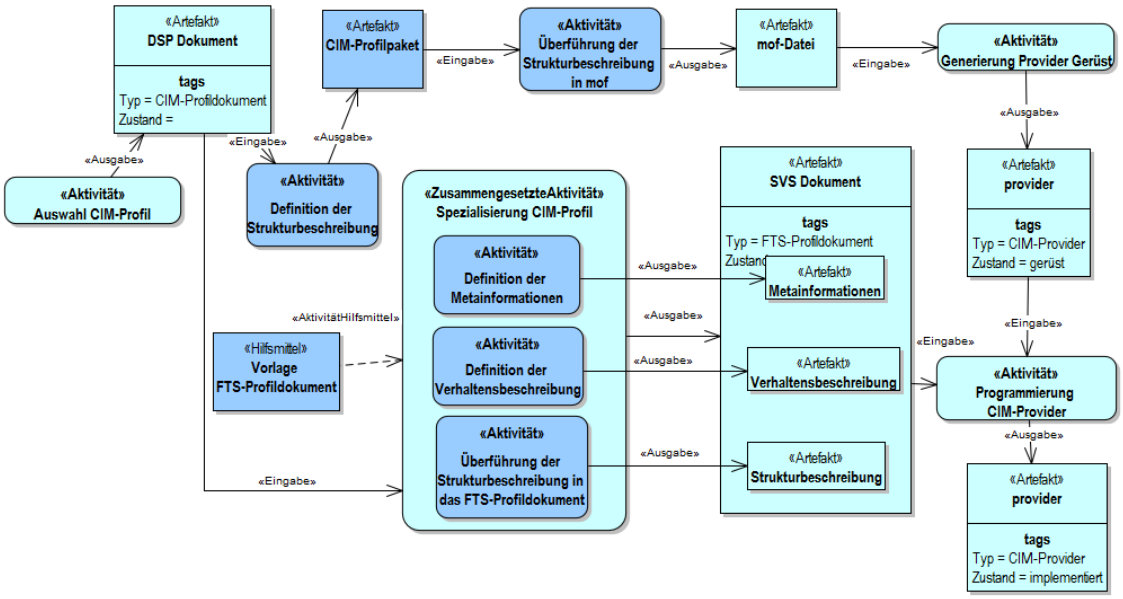


Abbildung 5.16: *Prozessmodell der Ist-Methode der CIM-Entwicklung – Iteration 2*

weitere Aktivitäten

Eine weitere Aktivität ist die Definition der Strukturbeschreibung, bei der das CIM-Profil mit einem UML-Modellierungswerkzeug erstellt wird. Dabei wird das CIM-Profilpaket erstellt. Die Aktivität *Überführung der Strukturbeschreibung in MOF* beschreibt den Generierungsschritt aus den Modellinformationen einzelner CIM-Elemente MOF-Dateien zu erstellen.

weitere Herausforderungen

Auf Basis der beschriebenen neuen Ist-Methode und den neuen Herausforderungen, die in den ersten Absätzen dieses Abschnitts beschrieben worden sind, entstehen neue Methodenanforderungen. Es wird lediglich das Endergebnis dargestellt. Die einzelnen Schritte der Definition der gültigen Situationsfaktoren und die einzelnen Schritte der Verbesserungs-Analyse sind bereits in den vorangegangen Beispielen (siehe Abschnitt 4.4.3 und 5.2.3) nachvollziehbar dargestellt worden. Das Ergebnis der Verbesserungs-Analyse der zweiten Iteration der Methodenweiterentwicklung im Anwendungsbeispiel *CIM-Entwicklung* ist das in Abbildung 5.17 dargestellte Anforderungsmodell. In der Abbildung ist bereits die Grup-

pierung der Anforderungen im Sinne der definierten Projekte skizziert. Die Projekte sind auf Basis der Prioritäten von FTS ermittelt worden. Auf diese Weise können die Ergebnisse der weiterentwickelten Methode direkt in der Produktentwicklung angewendet werden. Das bedeutet, die Bearbeitung der Projekte wird mit der Produktentwicklung synchronisiert.

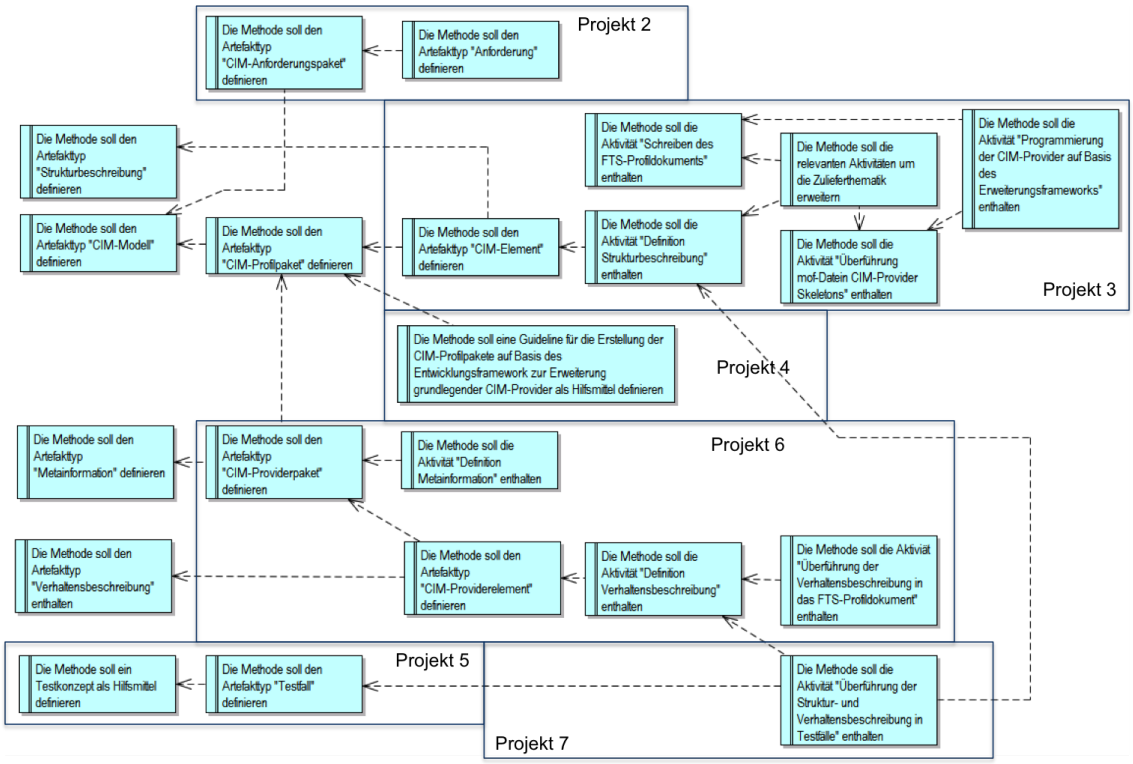


Abbildung 5.17: *Anforderungsmodell für das Anwendungsbeispiel der CIM-Entwicklung – Iteration 2*

Die folgende Beschreibung der Projekte übernimmt ebenfalls die Erläuterung der einzelnen Anforderungen. Projekte

Projekt 2: Die Anforderungen an CIM sollen modellbasiert erhoben werden. Dafür soll das CIM-Modell um ein CIM-Anforderungspaket erweitert werden, welches die einzelnen Anforderungen enthält. Auf diese Weise ist es langfristig möglich die Anforderungen direkt mit Elementen des CIM-Modells in Bezug zu setzen. Dadurch wird eine Nachverfolgbarkeit der Anforderungen im Entwicklungsprozess ermöglicht. Die Definition der Sicht auf die Anforderungen soll aus Dokumentationszwecken Dokument-basiert stattfinden. Allerdings sind keine detaillierten Anforderungen an die Überführung der Anforderun-

gen im CIM-Modell in ein Dokument definiert worden. Diese Möglichkeit ist aber in weiteren Iterationen der Methodenweiterentwicklung realisierbar.

Projekt 3: Dieses Projekt umfasst die Anpassung der vorliegenden Ist-Methode um den Zuliefererraspekt. Hauptsächlich muss die Aktivität zur Erstellung der Strukturbeschreibung angepasst werden, da weitere Rahmenbedingungen und Vorgaben, aufgrund der Festlegungen des Zulieferers für die grundlegenden CIM-Provider, berücksichtigt werden müssen. Danach wird die Spezifikation der FTS-spezifischen CIM-Provider, in Form der FTS-Profildokumente und der MOF-Dateien, definiert.

Projekt 4: Im Anschluss an Projekt 3 können die CIM-Provider nicht mehr direkt, sondern müssen mit Hilfe eines Entwicklungsframeworks des Zulieferers programmiert werden. Dafür soll eine Guideline erstellt werden, die die einzelnen Schritte dokumentiert.

Projekt 5: Nach der Durchführung des Projekts 4 werden die im Entwicklungsprojekt bei FTS spezifizierten CIM-Provider implementiert. In dieser Zeit soll das Testkonzept zum Testen der CIM-Provider erstellt werden. Es werden Testwerkzeuge vom Zulieferer gestellt. Darum müssen die bisherigen Methodenanforderungen an das Testkonzept mit seinen Testfällen angepasst werden.

Projekt 6: Aus Gründen des hohen Aufwands der Spezifikation des CIMs und der manuellen Erstellung und Anpassung der FTS-Profildokumente, sind anschließend an Projekt 5 der Fokus auf die komplette Spezifikation des CIMs mittels des CIM-Modells verschoben worden. Das ermöglicht die Generierung der FTS-Profildokumente und reduziert den Erstellungsaufwand enorm, da mehr Ressourcen für die inhaltliche Spezifikation im CIM-Modell genutzt werden konnten, anstatt diese für das Schreiben der Dokumente mit den Formatierungs-, Layout- und Review-Aufgaben zu belegen.

Projekt 7: Der konsequente Schritt im Projekt 7 befasst sich mit der Anpassung der Methode, so dass die im Testkonzept festgelegten Testfälle ebenfalls aus den Informationen des CIM-Modells abgeleitet und letztendlich automatisiert generiert werden können.

Iterative Methodenentwicklung

In diesem Beispiel sehen wir den Vorteil der Erhebung der Methodenanforderungen bei sich ändernden Anforderungen. Die grundlegende Menge der Methodenanforderungen mit langfristigen Zielen kann in einer weiteren Iteration wieder zur Hand genommen werden, um wie hier dargestellt, das ursprüngliche Anforderungsmodell anzupassen und zu erweitern. Auf diese Weise werden die ursprünglichen Ziele, in den bereits vorhandenen Methodenanforderungen, berück-

sichtigt. In diesem Beispiel ist in der ersten Iteration als ein zweites Projekt definiert worden ein Testkonzept zu entwickeln. In der zweiten Iteration haben sich die Grundlagen verändert, die Methodenweiterentwicklung definiert neue Iterationen, aber die Anforderung an das Testkonzept bleibt bestehen. Letztendlich wird von der iterativ weiterentwickelten Methode profitiert, indem im Projekt 7 sogar die Methodenanforderung betrachtet wird die Testfallgenerierung auf Basis des Testkonzepts umzusetzen.

5.3 Zusammenfassung

Bei der Verbesserungs-Analyse der Methodenweiterentwicklung hat der Methodenentwickler bei der Aufgabe der Definition der Methodenanforderungen die Schwierigkeit gehabt keine passende Anleitung zur Erhebung und Definition der Methodenanforderungen zu Verfügung zu haben. Die Definition von Methodenanforderungen wird in verschiedenen Methodenentwicklungsansätzen vorausgesetzt. Lediglich ein Ansatz zur Ermittlung der notwendigen Anforderungen an Entwicklungsaktivitäten war vorhanden (siehe Abschnitt 5.1.1).

Erhebung MA

Dieses Kapitel stellt einen pragmatischen Ansatz zur Erhebung von Methodenanforderung vor. Der Ansatz bildet genutzte Konzepte der gemeinen Anforderungserhebung in der Software- und Systementwicklung auf die Methodenentwicklung ab und bezieht insbesondere die Kenntnisse über Methodenelemente und definierte Herausforderungen im Sinne gültiger Situationsfaktoren in die Betrachtung mit ein.

MetaMe++

Der pragmatische Ansatz der Erhebung von Methodenanforderungen wird in das Gesamtkonzept MetaMe++ integriert und unterfüttert die MetaMe++-Verbesserungs-Analyse (siehe Abschnitt 5.2.1).

Anwendbarkeit

Es ist die Anwendbarkeit der Verbesserungs-Analyse, deren zentrale Elemente die resultierenden Methodenanforderungen darstellen, in Praxis-relevanten Anwendungsbeispielen aufgezeigt worden (siehe Abschnitt 5.2.2 und 5.2.3). Weiterhin ist eine iterative Projektplanung im Rahmen der Methodenweiterentwicklung durchgeführt worden (siehe 5.2.4).

Grundlage für die Weiterentwicklung

Auch wenn der hier gezeigte Ansatz lediglich grundlegende Erkenntnisse als

Baustein für MetaMe++ definiert und bei weitem nicht so detailliert formalisiert ist wie Techniken zur Anforderungserhebung und Anforderungsverwaltung, die in der Softwareentwicklung genutzt werden, ist das Endergebnis ein Anforderungsmodell. Durch die modellbasierte Definition der Anforderungen ist die Grundlage geschaffen weiterführende Konzepte, wie beispielsweise die Nachverfolgung der Anforderungen im Methodenweiterentwicklungsprojekt, zu realisieren.

Expertenwissen

Allerdings ist festgestellt worden, dass die Erhebung von Methodenanforderung nicht alleinig vom Methodenentwickler durchgeführt werden kann. Er braucht zum einen die Informationen der gültigen Situationsfaktoren und zum anderen Kenntnisse über die notwendigen Methodenelemente. In beiden Fällen muss der Methodenentwickler auf Expertenwissen der Entwickler des Produkts, für die die Entwicklungsmethode erstellt wird, und weiterer Stakeholder der Methode, zurückgreifen können.

situative MA

Auch wenn keine definierten Guidelines für die Berücksichtigung der gültigen Situationsfaktoren in der Methodenweiterentwicklung definiert werden können, bilden insbesondere die situativen Methodenanforderungen eine Möglichkeit auf die durch die Situationsfaktoren definierten Gegebenheiten, nach besten Wissen und Gewissen, reagieren zu können.

Weiterführung

Eine weiterführende Betrachtung im Rahmen von Methodenanforderungen ist die Definition von Überdeckungskriterien. Es ist vorstellbar dadurch Aussagen über den Reifegrad einer Methode zu treffen.

6 MetaMe++

Kontext Methodenweiterentwicklung

Die Phasen *Ist-Analyse*, *Verbesserungs-Analyse*, *Konzeption* und *Realisierung* beschreiben das Vorgehen der Methodenweiterentwicklung

Ausgangslage

Während der Ist-Analyse wird der Ist-Zustand der weiter zu entwickelnden Methode über die Ist-Methode und den Methodenkontext beschrieben. Für die Darstellung der Ist-Methode wird in Abschnitt 2.5 die Sprache MMMB vorgestellt, mit der Methoden, konform zur grundlegenden Metamethode MetaMe, beschrieben werden können. In Abschnitt 3.6 wird ein Vorgehen zur Ermittlung und Beschreibung der Ist-Methode definiert und auf die Anwendungsbeispiele dieser Arbeit angewendet. Das methodische Vorgehen zur Ermittlung des Methodenkontexts (=Situationsbeschreibung) wird im Abschnitt 4.4 vorgestellt. Die beiden resultierenden Lösungsbausteine *Ermittlung Ist-Methode* und *Ermittlung Methodenkontext* definieren die Phase *Ist-Analyse*. Im Rahmen der Phase *Verbesserungs-Analyse* werden mit Hilfe der Erkenntnisse aus der Ist-Analyse Optimierungspotenziale identifiziert und in Methodenanforderungen überführt. Dafür ist im Kapitel 5 das Thema der Methodenanforderung diskutiert und im Abschnitt 5.2 ein Vorgehen zur Ermittlung der Methodenanforderungen im Rahmen der Methodenweiterentwicklung vorgestellt worden. Darüber hinaus werden die Methodenanforderungen analysiert und mittels iterativen Projekten der Methodenweiterentwicklung aufgeteilt. Für ein Projekt der Methodenweiterentwicklung muss nun die Konzeption der Soll-Methode durchgeführt werden, indem die Methodenanforderungen in das Entwurfsmodell der Soll-Methode überführt werden.

Konzeption

Der Lösungsansatz der Dissertation sieht in der Phase *Konzeption* die Nutzung von MetaMe (vgl. Abschnitt 2.4) vor. MetaMe definiert bereits die Tätigkeiten zur Konzeption einer Soll-Methode. Ebenfalls sind entsprechende methodische Vorgaben im Vorgehen der Methodenweiterentwicklung 2.3.3 beschrieben. Diese grundlegenden Definitionen müssen in eine formale Darstellung mit der Sprache MMMB überführt werden, damit sie im Anwendungskontext MetaMe++ integriert werden können. Damit wird der Lösungsbaustein *Konzeption* definiert und

besteht aus Prozess- und Produktmodell konform zum MetaMe-Vorgehen. Die Ergebnisse werden in Abschnitt 6.1 dargestellt und exemplarisch auf die Anwendungsbeispiele dieser Arbeit angewendet.

Integration der Lösungsbausteine

Nachdem die Phase *Konzeption* als weiterer Lösungsbaustein definiert ist, müssen alle Lösungsbausteine in das Gesamtvorgehen integriert werden. Das bedeutet, neben der Integration der einzelnen Prozessmodelle auch die Integration der einzelnen Produktmodelle. Das Resultat ist MetaMe++ mit Prozess- und Produktmodell und wird im Abschnitt 6.2 vorgestellt.

Sprachdefinition

Für das MMMB-Metamodell ist bereits eine konkrete Syntax und Semantik definiert worden (MMMB-UML-Profil). In den einzelnen Lösungsbausteinen werden weitere Produktmodelle mit neuen Artefakttypen definiert, die bei der Methodenweiterentwicklung genutzt werden. Für diese zusätzlichen Artefakttypen ist bisher noch keine konkrete Syntax und Semantik festgelegt worden. In den Anwendungsbeispielen sind jedoch bereits exemplarische Vorgaben für die Darstellung der Artefakte, die über die Artefakttypen getypt werden, vorgestellt worden. In dem Abschnitt 6.3 werden Vorschläge der Sprachfestlegung für alle Artefakttypen des MetaMe++-Produktmodells vorgestellt.

Das Kapitel schließt mit einer Zusammenfassung in Abschnitt 6.4.

6.1 Lösungsbaustein: Konzeption

Konzeption

Dieser Abschnitt beschreibt die Ergebnisse der formalen Definition der Aktivität *Konzeption* (siehe Abbildung 6.1). Zuerst wird die Aktivität mit Hilfe der Sprache MMMB in Abschnitt 6.1.1 dargestellt und anschließend zum einen auf die System-Architektur-Spezifikation (siehe Abschnitt 6.1.2) und zum anderen auf die CIM-Entwicklung (siehe Abschnitt 6.1.3) angewendet.

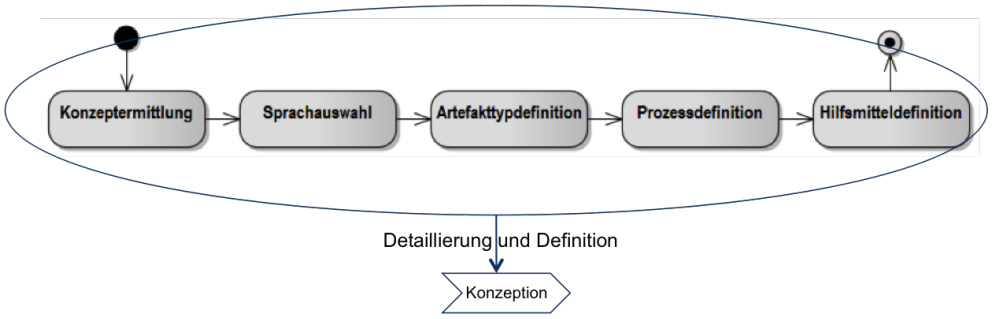


Abbildung 6.1: *Detaillierung: Konzeption*

6.1.1 Detaillierung der Aktivität *Konzeption*

Entwurfsmodell

Die zu erledigende Aufgabe des Methodenentwicklers in der Phase *Konzeption* ist die Überführung der Methodenanforderungen, die für eine Iteration der Methodenweiterentwicklung im Rahmen eines Projekts definiert worden ist, in ein konzeptionelles Entwurfsmodell der Soll-Methode. Dafür definiert MetaMe die einzelnen Schritte *Konzeptermittlung*, *Sprachauswahl*, *Artefakttypdefinition*, *Prozessdefinition* und *Hilfsmitteldefinition*.

Disziplinen

Ursprünglich definiert MetaMe zusätzlich die Tätigkeit *Definiere Domäne und Disziplinen* (siehe Abbildung 2.11). Diese Aufgabe wird bei MetaMe++ nicht mehr benötigt. Die Definition der (Entwicklungs-)Domäne erfolgt bereits durch die Ermittlung der gültigen Situationsfaktoren in der Ist-Analyse (siehe Abschnitt 4.4). Das Konstrukt der Disziplinen bietet die weitergehende Strukturierung eines Prozesses an. Die Ermittlung der Disziplinen erfolgt bei MetaMe++ bereits bei der Ermittlung der Methodenanforderungen. Falls Anforderungen an die weiteren Disziplinen als Strukturierungselemente gestellt werden, müssen daraus Methodenanforderungen bezüglich Entwicklungsphasen, die von der Methode definiert werden sollen, hervorgegangen sein.

Domänenmodell

In dem Schritt der Konzeptermittlung soll nach der Definition von MetaMe ein Domänenmodell der Konzepte erstellt werden. Die Konzepte sollen durch eine Analyse der Entwicklungsdomäne erarbeitet werden. Die Aufgabe der Analyse wird in der Phase *Verbesserungs-Analyse* ähnlich durchgeführt. Die Definition des Domänenmodells der Konzepte definiert in einem ersten Schritt die Entwicklung des Produktmodells, welcher in einem zweiten Schritt um die Sprachdefinition erweitert wird. Ein im Domänenmodell enthaltenes Konzept, gepaart mit einer Sprache, bildet dann ein Artefakttyp. Die Gesamtheit der Artefakttypen definiert dann

das Produktmodell. In dem Fall, dass die Artefakttypen, ohne Berücksichtigung der Sprache im Produktmodell, betrachtet werden, übernimmt das Produktmodell die Rolle des Domänenmodells.

Produktmodell

Es stellt sich also die Frage, warum zwei Modelle, das Domänenmodell und das Produktmodell, erstellt werden müssen. Dieser explizite Schritt resultiert aus der Arbeit von [ESS08]. Hier wird lediglich die Sprachauswahl und Zuweisung der Sprachen zu den Konzepten detailliert. In der Dissertation [Sau11b] wird ausschließlich argumentiert, dass durch die Zuweisung der Sprachen, den Konzepten eine konkrete Syntax und eventuell semantische Eigenschaften hinzugefügt werden. Ausdrücklich wird auf die Kompatibilität der Sprache für die Konzepte, aber auch die Kompatibilität der definierten Abhängigkeiten zwischen den Konzepten und den Abhängigkeiten zwischen den Artefakten hingewiesen. Das bedeutet, die Modelle Domänenmodell und Produktmodell sind bezüglich der Abhängigkeiten der Elemente untereinander kompatibel und lediglich um ihre Spracheigenschaften erweitert. Anders ausgedrückt, das Produktmodell, ohne Spracheigenschaften, ist kompatibel zum Domänenmodell.

Der Analyse- und Entwicklungsschritt der Definition des Domänenmodells aus MetaMe wird bei MetaMe++ bereits implizit durch die Verbesserungs-Analyse mit der Erhebung der Methodenanforderungen ersetzt. Aus den Methodenanforderungen können ebenfalls Hinweise für notwendige Artefakttypen abgeleitet werden, wie aus den Konzepten des Domänenmodells.

Darüber hinaus hat in der Praxis bei der Überführung des Domänenmodells in das Produktmodell in keinem Fall eine Detaillierung der Abhängigkeiten stattgefunden. Es sind lediglich verschiedene Anpassungen und Verfeinerungen an Inkrementen des Produktmodells selber vorgenommen worden.

Domänenmodell vs. initiales Produktmodell

Aus den genannten Gründen wird bei MetaMe++ auf die Erstellung eines expliziten Domänenmodells verzichtet. Es wird direkt ein initiales Produktmodell erstellt, bei dem die ermittelten Artefakttypen noch keine konkrete Syntax und Semantik besitzen müssen.

Betrachten wir nun die Überführung der konzeptionellen Schritte in MetaMe++. Das Prozessmodell der Methodenweiterentwicklung der zusammengesetzten Aktivität *Konzeption* wird im Folgenden beschrieben (siehe Abbildung 6.2).

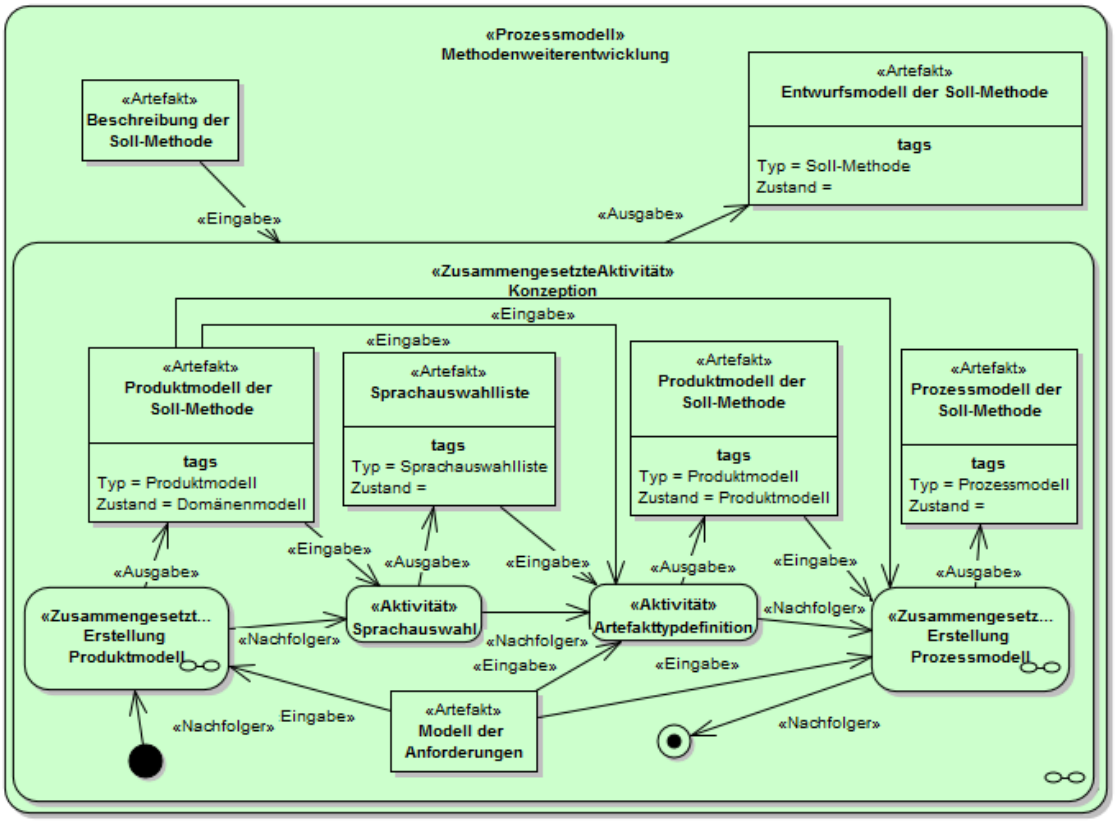


Abbildung 6.2: *MetaMe++-Prozessmodell der Konzeption*

Die zusammengesetzte Aktivität *Konzeption* benötigt als Eingabe-Artefakt das zu bearbeitende Projekt. Dann werden die Aktivitäten *Erstellung Produktmodell* mit Ergebnis des initialen Produktmodells, *Sprachauswahl* mit Ergebnis der Sprachauswahlliste, *Artefakttypdefinition* mit detailliertem Produktmodell und *Erstellung Prozessmodell* mit Ergebnis des Prozessmodells ausgeführt. Im Abschluss liefert die Konzeption das Entwurfsmodell der Soll-Methode, bestehend aus Produkt- und Prozessmodell.

Die *Erstellung Produktmodell* ist eine zusammengesetzte Aktivität (siehe Abbildung 6.3).

Im ersten Schritt werden die initialen Artefakttypen auf Basis der Methodenanforderungen erhoben. Dafür muss ein Produktmodell erstellt werden, in welchem die Artefakttypen definiert werden können.

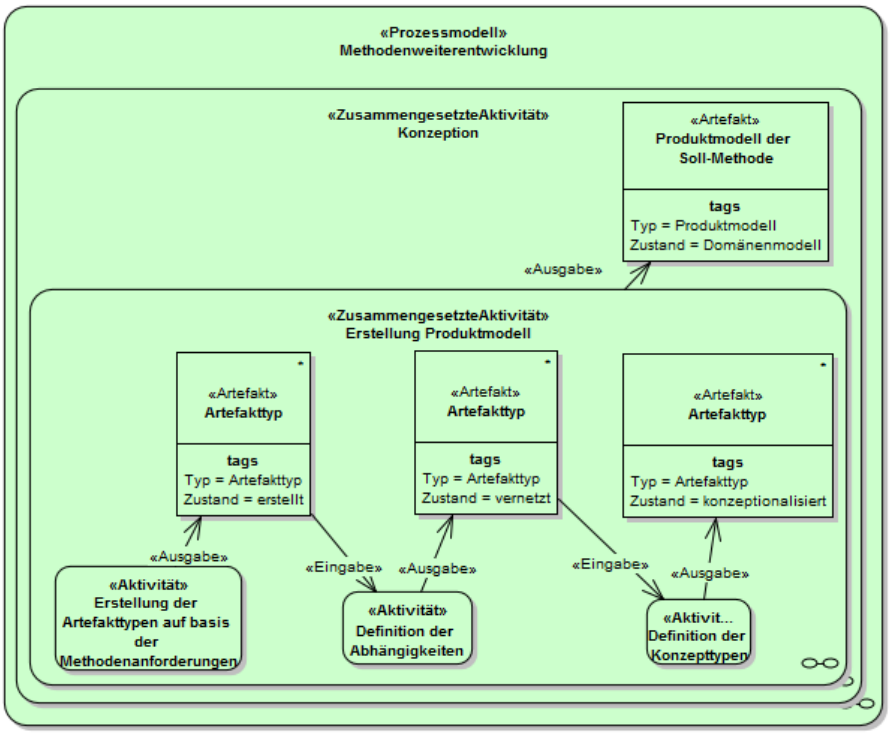


Abbildung 6.3: *MetaMe++ - Prozessmodell der Konzeption::Erstellung Produktmodell*

«Aktivität» Erstellung der Artefakttypen auf Basis der Methodenanforderungen

Beschreibung: Für jede Methodenanforderung, die einen Artefakttyp fordert, wird ein Element vom Typ «Artefakttyp» im Produktmodell erstellt. Der Zustand der Artefakttypen im Rahmen der zu entwickelnden Methoden ist *erstellt*. Jeder Artefakttyp darf wiederum Zustände im Lebenszyklusmodell definieren, die von instanziierten Artefakten der zu entwickelnden Methode eingenommen werden können.
Ergebnis: Produktmodell mit Artefakttypen und deren Lebenszyklusmodell.

Im zweiten Schritt werden die Artefakttypen miteinander in Beziehung gesetzt.

«Aktivität» Definition der Abhängigkeiten

Eingabe: Produktmodell mit Artefakttypen
Beschreibung: Die Artefakttypen des Produktmodells müssen miteinander in Beziehung gesetzt werden. Dafür bietet MMMB die Beziehungen *Abhängigkeit, Fachliche Verfeinerung* und *Artefakttyp Verfeinerung* an.
Ausgabe: Produktmodell mit vernetzten Artefakttypen

Im letzten Schritt der zusammengesetzten Aktivität *Erstellung Produktmodell* werden die Artefakttypen um die Definition ihres Konzepttyps erweitert.

«Aktivität» Definition der Konzepttypen

Eingabe: Produktmodell mit vernetzten Artefakttypen
Beschreibung: Jedem Artefakttyp soll ein Konzepttyp zugewiesen werden. Der Konzepttyp kann die folgenden Werte enthalten: *Produktkonzept* oder *Entwicklungskonzept*. Das Produktkonzept kann weiter unterteilt werden in *Strukturbeschreibung, Verhaltensbeschreibung* oder *Struktur- und Verhaltensbeschreibung*. Ausgabe: Initiales Produktmodell mit konzeptionalisierten Artefakttypen

Der nächste Schritt bei der Konzeption ist die Sprachdefinition, bei der die Artefakttypen um eine Sprache erweitert werden. Dafür wird zuerst eine Übersicht über mögliche Sprachen erstellt. Das heißt, es ist zu ermitteln, mit welcher Sprache ein Artefakt vom Typ des Artefakttyps beschrieben werden kann. Anders ausgedrückt, wie sieht die konkrete Syntax einer Instanz des Artefakttyps aus. Abhängig von den unterschiedlichen Abstraktionsgraden und Informationsinhalten, die für einen Artefakttyp gelten, kann das Ergebnis vielfältig sein. Darüber hinaus können Methodenanforderungen definiert sein, die an dieser Stelle konkrete Vorgaben machen, die zu berücksichtigen sind. Mögliche Sprachelemente sind:

- die natürliche Sprache in Form von Freitext, festgelegt auf eine Landessprache
- strukturierte natürliche Sprachelemente definiert über Vorlagen, Tabellen, Listen, etc.
- visuelle Sprachelemente, Grafiken, Diagramme, Bilder, etc.
- formalisierte standardisierte Sprachelemente, UML, SysML, BPMN, etc.
- formalisierte domänenspezifische Sprachelemente, definiert über abstrakte Syntax, konkrete Syntax und Semantik
- formale Sprachen, definiert über formale Grammatik und formale Semantik
- Mischformen der genannten Ausprägungen

«Aktivität» Sprachauswahl

Beschreibung: Für jeden Artefakttyp muss eine Liste von möglichen Sprachen erstellt werden, mit der die konkrete Syntax einer Instanz des Artefakttyps dargestellt wird. Die Sprachauswahlliste soll konform zu definierten Methodenanforderungen erstellt werden.
Ergebnis: Sprachauswahlliste für jeden Artefakttyp

In dem weiteren Schritt der Artefakttypdefinition wird einem Artefakttyp eine oder mehrere Sprachen aus der Sprachauswahlliste fest zugeordnet.

«Aktivität» Artefakttypdefinition

Beschreibung: Für jeden Artefakttyp des Produktmodells wird eine oder mehrere Sprachen aus der Sprachauswahlliste zugeordnet.
Ergebnis: Produktmodell

Das Resultat ist das vollständig definierte Produktmodell der zu entwickelnden Methode.

Mit der Kenntnis über das Produktmodell wird das Prozessmodell der zu entwickelnden Methode erstellt (siehe Abbildung 6.4).

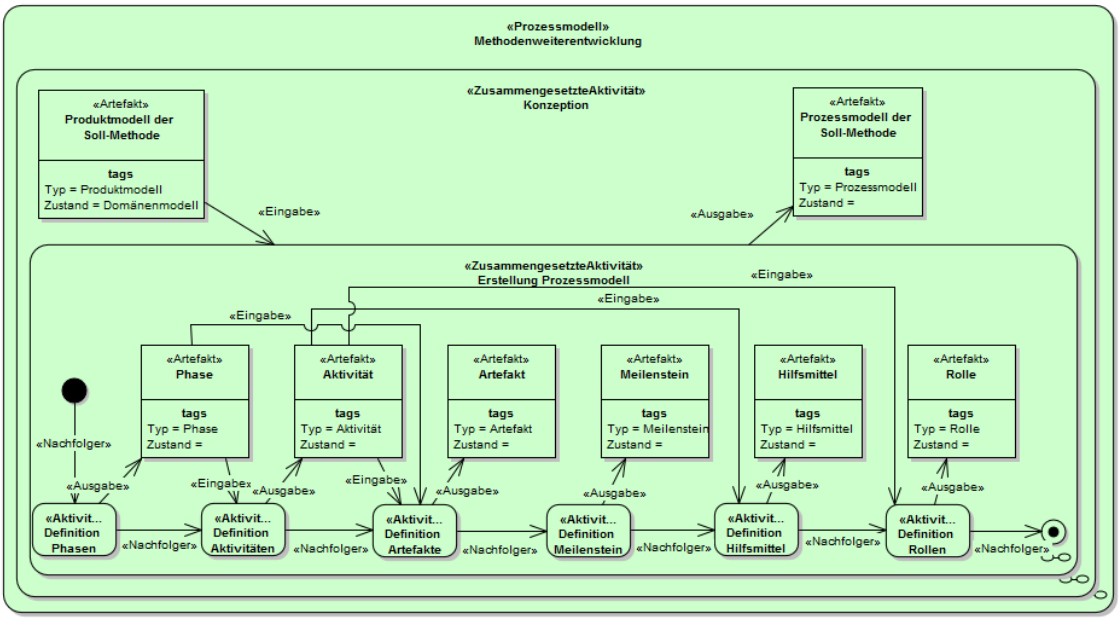


Abbildung 6.4: *MetaMe++-Prozessmodell der Konzeption::Erstellung Prozessmodell*

«Aktivität» Definition Phasen

Beschreibung: Für jede Methodenanforderung, die eine Phase fordert, wird ein Element vom Typ «Phase» im Prozessmodell erstellt.
Ergebnis: Prozessmodell mit Phasen

«Aktivität» Definition Aktivitäten

Beschreibung: Für jede Methodenanforderung, die eine Aktivität fordert, wird ein Element vom Typ «Aktivität» im Prozessmodell erstellt. Die Aktivität kann einer Phase zugeordnet werden.
Ergebnis: Prozessmodell mit Phasen und Aktivitäten

«Aktivität» Definition Artefakte
Beschreibung: Für jede Methodenanforderung, die ein Artefakt fordert, wird ein Element vom Typ «Artefakt» im Prozessmodell erstellt. Dem Artefakt wird ein Typ zugewiesen, der im Produktmodell definiert ist. Fehlt der entsprechende Artefakttyp, so muss das Produktmodell um den entsprechenden Typen erweitert werden. Definiert das Artefakttypmodell Zustände für den Artefakttypen, so muss dem typisierten Artefakt ein gültiger Zustand zugeordnet werden. Das Artefakt kann einer Phase zugeordnet werden.
Ergebnis: Prozessmodell mit Phasen, Aktivitäten und Artefakten

«Aktivität» Definition Meilensteine
Beschreibung: Für jede Methodenanforderung, die einen Meilenstein fordert, wird ein Element vom Typ «Meilenstein» im Prozessmodell erstellt. Der Meilenstein wird über ein oder mehrere Artefakte mit jeweils einem Zustand definiert. Der Meilenstein muss die Artefakte mit der Assoziation Ergebnis referenzieren.
Ergebnis: Prozessmodell mit Phasen, Aktivitäten, Artefakten und Meilensteinen

«Aktivität» Definition Hilfsmittel
Beschreibung: Für jede Methodenanforderung, die ein Hilfsmittel fordert, wird ein Element vom Typ «Hilfsmittel» im Prozessmodell erstellt. Das Hilfsmittel soll einer Aktivität über die Assoziation AktivitätHilfsmittel zugeordnet werden. Fehlt eine entsprechende Aktivität im Prozessmodell, muss das Prozessmodell um die entsprechende Aktivität erweitert werden.
Ergebnis: Prozessmodell mit Phasen, Aktivitäten, Artefakten, Meilensteinen und Hilfsmitteln

«Aktivität» Definition Rollen
Beschreibung: Für jede Methodenanforderung, die eine Rolle fordert, wird ein Element vom Typ «Rolle» im Prozessmodell erstellt. Die Rollen müssen einer Aktivität über die Assoziation Durchführer oder Teilnehmer zugewiesen werden. Fehlt eine entsprechende Aktivität im Prozessmodell, muss das Prozessmodell um die entsprechende Aktivität erweitert werden.
Ergebnis: Prozessmodell mit Phasen, Aktivitäten, Artefakten, Meilensteinen, Hilfsmitteln und Rollen

Sind alle Schritte durchgeführt, ist das Resultat das Prozessmodell des Entwurfsmodells der Soll-Methode.

keine explizite Validierung

Die Validierung des Entwurfsmodells mit Produkt- und Prozessmodell wird nicht explizit definiert. Iterationen und Durchsichten müssen durchgeführt werden. Die Validierung, die bereits bei der Ermittlung der Ist-Methode definiert worden ist (siehe Abbildung 3.18, aus Abschnitt 3.6.1), kann an dieser Stelle ebenfalls Anwendung finden. Darüber hinaus kann es vorkommen, dass die Methodenanforderungen nicht vollständig sind und die Methode nicht in sich geschlossen definiert worden ist. Wird bei der Validierung ein solches Ergebnis erkannt, liegt es in der

Verantwortung des Methodenentwicklers die Methode zu vervollständigen. Dafür wird, wie bei der Erstellung der Ist-Methode, die Checkliste zur Ermittlung von Methodenelementen (siehe Tabelle 3.8 in Abschnitt 3.6.1) genutzt, um fehlende Methodenelemente zu identifizieren.

Der strukturelle Zusammenhang, der in der Phase *Konzeption* genutzten Artefakte, ist im Produktmodell mit den grundlegenden Artefakttypen festgelegt (siehe Abbildung 6.5).

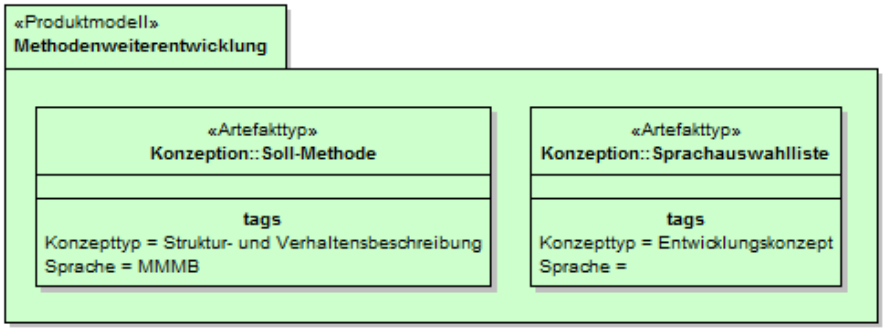


Abbildung 6.5: *MetaMe++-Produktmodell der Konzeption*

Das Produktmodell ist übersichtlich. Es besteht aus der Soll-Methode und der Sprachauswahlliste. Die Sprachauswahlliste ist als Entwicklungskonzept definiert, da es nur als Hilfsmittel bei der Methodenweiterentwicklung genutzt wird, jedoch kein Element der zu erstellenden Methode ist. Die Soll-Methode beschreibt lediglich das Entwurfsmodell der zu erstellenden Methode mit ihren Struktur- und Verhaltenselementen. Die Soll-Methode wird ausschließlich mit der Sprache MMMB beschrieben. Das ist der Grund, warum die Methodenelemente, die im Rahmen der Konzeption als Hilfsmittel erstellt werden, nicht im Produktmodell der Konzeption auftauchen. Sie sind lediglich im Methodenmetamodell definiert.

Aus dem Prozessmodell der Konzeption bei MetaMe++ ist allerdings zu erkennen, dass die Methodenelemente im Methodenmetamodell erweitert werden müssen. Zum einen werden die Zustände *Domänenmodell* und *Produktmodell* für das Element vom Typ Produktmodell definiert, zum anderen werden die Zustände *erstellt*, *vernetzt* und *konzeptionalisiert* für Elemente vom Typ Artefakttyp benutzt. Die Erweiterung wird im Abschnitt 6.2 bei der Integration der Produktmodelle für das gesamte Vorgehen MetaMe++ thematisiert.

6.1.2 Konzeption für das Anwendungsbeispiel der System-Architektur-Spezifikation

Ausgangslage

Dieser Abschnitt stellt das Ergebnis der Konzeption für das Anwendungsbeispiel der System-Architektur-Spezifikation vor. Die Ausgangslage war die Ist-Methode der System-Architektur-Spezifikation, die im Abschnitt 3.6.2 vorgestellt worden ist. Im Rahmen der Verbesserungs-Analyse sind in Abschnitt 5.2.2 Methodenanforderungen definiert worden, die im Rahmen der Methodenweiterentwicklung umgesetzt werden sollen. Im Forschungs- und Entwicklungsprojekt sind drei iterative Methodenweiterentwicklungsprojekte definiert worden. Die resultierende Soll-Methode nach Durchführung des dritten Projekts wird im Folgenden dargestellt.

Produktmodell

Zuerst wird das resultierende Produktmodell der System-Architektur-Spezifikation erläutert (siehe Abbildung 6.6).

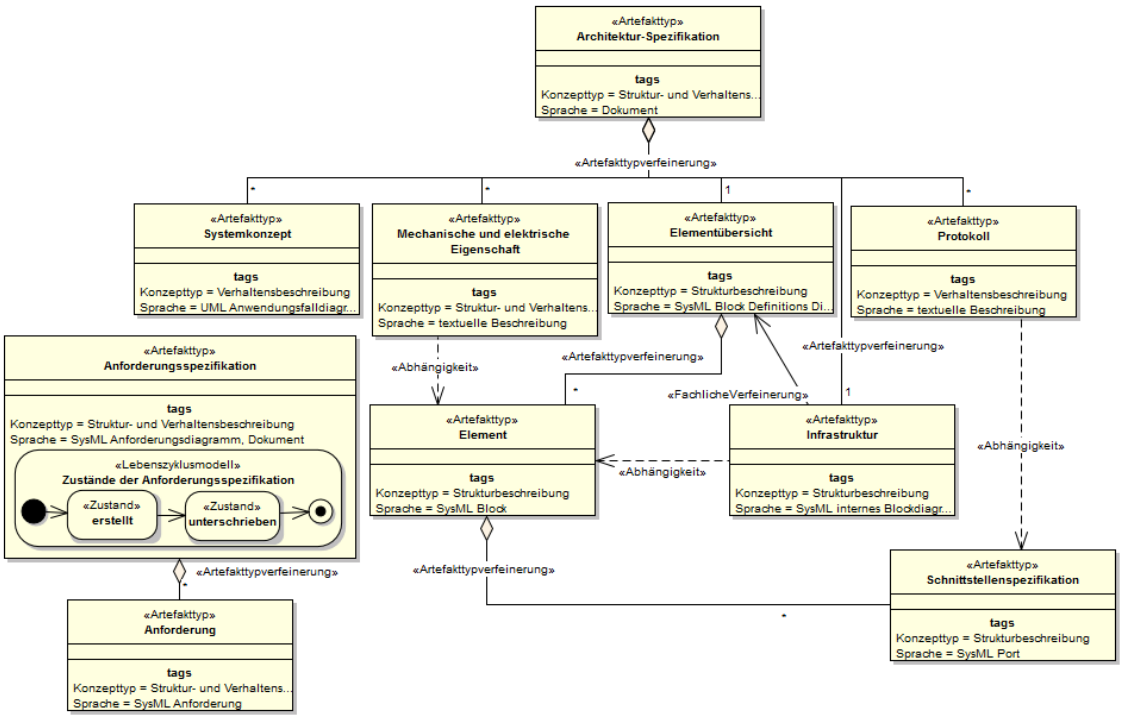


Abbildung 6.6: *Konzeption Produktmodell der Soll-Methode für das Anwendungsbeispiel der System-Architektur-Spezifikation*

Anforderungsspezifikation

Aus den Methodenanforderungen bezüglich der Anforderungserhebung bei der

Server-System-Entwicklung ist der Artefakttyp *Anforderungsspezifikation* entwickelt worden. Das ist ein Dokument, welches die ursprüngliche Rolle des Product Agreement Paper als Produktdefinitionsvereinbarung übernimmt. Dadurch war es notwendig die Zustände *erstellt* und *unterschrieben* für das Dokument zu definieren, da erst das unterschriebene Anforderungsdokument als festgelegte Produktdefinitionsvereinbarung gilt, es aber auch schon frühere Versionen des Dokuments geben kann. Die Anforderungsspezifikation definiert einzelne Anforderungen an das Gesamtsystem. Die Anforderungen werden mittels eines SysML-Anforderungsdiagramms verwaltet. Auf diese Weise können Abhängigkeiten der Anforderungen definiert werden. Mit passenden SysML-Modellierungswerkzeugen ist es möglich die Verwaltung der Anforderungen modellbasiert durchzuführen. Eine einzelne Anforderung wird vom Artefakttyp Anforderung definiert. Als Sprache wird diesem Artefakttyp das Element SysML-Anforderungen zugewiesen. Eine SysML-Anforderung enthält die Metainformationen Status, Priorität, Schwierigkeitsgrad, Herkunft, Relation, ID, Version, Phase, Kurz- und Detailbeschreibung.

Architektur-Spezifikation

Weiterhin mussten gemäß den Methodenanforderungen die Artefakttypen *System-Architektur-Spezifikation* und *Modul-Spezifikation* definiert werden. Im Abschnitt 3.4 ist erkannt worden, dass ein Server-System weitere Module enthält, die wiederum eine dem Server-System ähnliche Struktur besitzen. Aus diesem Grund ist es sinnvoll für ein System und ein Modul die gleichen strukturellen Elemente zu spezifizieren. Daraus resultiert der Artefakttyp *Architektur-Spezifikation*, der auf System-Ebene genutzt wird, um ein Server-System zu spezifizieren, aber auch auf der Modul-Ebene genutzt werden, um ein ein System-Modul eines Server-Systems zu spezifizieren. Eine Architektur-Spezifikation enthält Systemkonzepte, mechanische und elektrische Eigenschaften, eine Elementübersicht sowie eine Infrastruktur und Schnittstellenbeschreibung mit Protokollbeschreibung. Die Inhalte der Architektur-Spezifikation werden in einem entsprechenden Dokument niedergeschrieben.

Systemkonzept

Für die Systemkonzepte ist der Artefakttyp *Systemkonzept* definiert worden. Mit Hilfe von UML-Anwendungsfalldiagrammen können die Systemkonzepte in abstrakter Weise formell beschrieben werden. Weitere für die UML-typischen Möglichkeiten der Verfeinerung der Anwendungsfälle über UML-Aktivitäts-, UML-Zustands- und UML-Sequenzdiagrammen wird ermöglicht, ist aber an dieser Stelle nicht weiter definiert.

mechanische, elektrische Eigenschaft

Der Artefakttyp *mechanische und elektrische Eigenschaft* beschreibt weitere Spezifika der System- oder Modul-Elemente. An dieser Stelle ist ein SysML-Profil angedacht, welches das SysML-Element SysML-Block um die relevanten Eigenschaften erweitert, aber aus zeitlichen Gründen nicht umgesetzt worden ist. Es bleibt zunächst die Spezifikation dieser Eigenschaften informell und textuell durchzuführen.

Elementübersicht

Der Artefakttyp *Elementübersicht* definiert die einzelnen System- oder Modul-Elemente als SysML-Block auf Typebene. Weiterhin wird mit Hilfe des SysML-Block-Definition-Diagramms der Zusammenhang der SysML-Blöcke definiert.

Element

Ein System- oder Modulelement wird über den Artefakttyp *Element* definiert. Die sprachliche Repräsentation eines Elements ist definiert als SysML-Block, der bereits in der Elementübersicht definiert worden ist. Ein SysML-Block kann Elemente des Typs SysML-Port aggregieren, die als Schnittstellenspezifikation genutzt werden. Aus diesem Grund wird der Artefakttyp Schnittstellenspezifikation über die Sprache SysML-Port definiert.

Protokolle

Bei der Schnittstellenspezifikation werden eindeutige Namen für die Schnittstelle und deren Datenfluss beschrieben. Es ist das Problem aufgetreten die unterschiedlichen Arten von Schnittstellen bei der Server-System-Entwicklung, das sind elektrische Schnittstellen unterschiedlichen Typs oder Software-Schnittstellen auf unterschiedlichen Ebenen des ISO-OSI-Modells, einheitlich beschreiben zu können. Darum ist neben der Beschreibung über einen SysML-Port eine zusätzliche textuelle Beschreibung der Protokolle im Architektur-Spezifikations-Dokument vorgesehen. Dafür ist der Artefakttyp *Protokoll* definiert worden, der sich immer auf eine Schnittstellenspezifikation bezieht.

Infrastruktur

Der Artefakttyp *Infrastruktur* verfeinert die Festlegung aus der Elementübersicht mit Hilfe eines internen Blockdiagramms. Instanzen der SysML-Blöcke und deren Verbindungen über Ports definieren eine konkrete Infrastruktur.

In den nächsten Absätzen wird das Prozessmodell der Soll-Methode beschrieben (siehe Abbildung 6.7).

Hilfsmittel

In der Phase *Analyse/Planung* wird mit der Anforderungsdefinition begonnen. Dabei werden Entwickler und der Produktverantwortliche aufgefordert die Anforderungen an das Gesamtsystem zu definieren. Ein erstes Hilfsmittel ist ein Anfor-

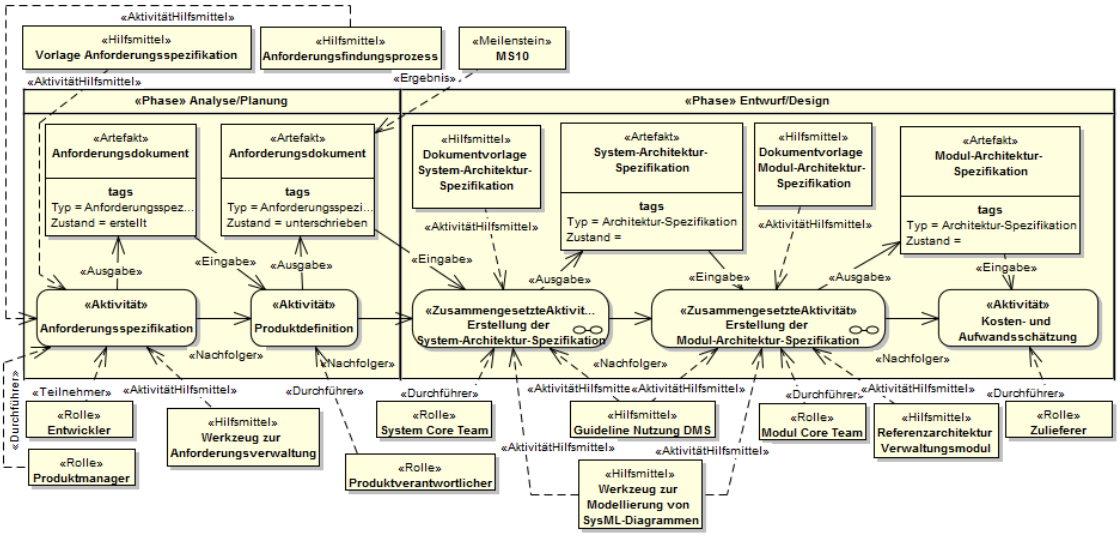


Abbildung 6.7: *Konzeption Prozessmodell der Soll-Methode für das Anwendungsbeispiel der System-Architektur-Spezifikation*

derungsfindungsprozess, der exemplarisch aufzeigt, wie Anforderungen ermittelt werden können. Ein zweites Hilfsmittel ist die Vorlage der Anforderungsspezifikation. Aus dieser Vorlage sind die möglichen Themen und Elemente zu entnehmen, für die Anforderungen bestimmt werden müssen. Weiterhin wird in der Vorlage vorgegeben, welche Informationen für einzelne Anforderungen zu definieren sind. Das dritte Hilfsmittel ist ein Werkzeug zur Anforderungsverwaltung.

«Aktivität» Anforderungsspezifikation

Hilfsmittel: Vorlage Anforderungsspezifikation, Anforderungsfindungsprozess, Werkzeug zur Anforderungsverwaltung

Beschreibung: Das Vorgehen im Anforderungsfindungsprozess soll durchgeführt werden. Die Ergebnisse werden konform zur Vorlage der Anforderungsspezifikation in einem Anforderungsdokument dokumentiert.

Ergebnis: Anforderungsspezifikation im Zustand erstellt.

Produktdefinition

Ist die Anforderungsspezifikation abgeschlossen, findet die Produktdefinition statt. In dieser Aktivität findet das Review der Anforderungsspezifikation durch die Entscheidungsträger statt. Die Produktdefinition wird vom Produktverantwortlichen organisiert. Ist das Review abgeschlossen, wird das Anforderungsdokument offiziell unterzeichnet und als Product Agreement Paper angesehen. Der Meilenstein MS10 ist erreicht und die Phase Entwurf/Design startet.

«Aktivität» Produktdefinition
Eingabe: Anforderungsdokument im Zustand erstellt
Beschreibung: Durchführung des Reviews der Anforderungsspezifikation durch die Entscheidungsträger mit anschließender Unterzeichung der Anforderungsspezifikation.
Ergebnis: Anforderungsspezifikation im Zustand unterzeichnet.

System-Architektur-Spezifikation

In der Phase *Entwurf/Design* wird die zusammengesetzte Aktivität *Erstellung der System-Architektur-Spezifikation* durchgeführt. Dafür wird als Hilfsmittel eine entsprechende Dokumentvorlage für die System-Architektur-Spezifikation bereitgestellt, die detaillierte Hinweise zur Ermittlung der notwendigen Informationen gibt. Des Weiteren wird definiert, dass bestimmte Inhalte mit Hilfe von SysML-Diagrammen beschrieben werden. Zur Vereinfachung der Erstellung dieser Diagramme soll ein Modellierungswerkzeug genutzt werden. Das hat neben der Werkzeug-unterstützten Erstellung der Diagramme einen weiteren Vorteil. Die Spezifikationsinhalte werden in einer modellbasierten Form erstellt und sind somit Maschinen-lesbar zugreifbar. Auf diese Weise können in weiteren Methodenweiterentwicklungsprojekten Konsistenzregeln und automatisierte Konsistenzprüfungen auf den Modellinhalten definiert werden.

«Aktivität» Erstellung der System-Architektur-Spezifikation
Eingabe: Anforderungsdokument im Zustand unterschrieben
Hilfsmittel: Dokumentvorlage System-Architektur-Spezifikation, Guideline Nutzung DMS, Werkzeug zur Modellierung von SysML-Diagrammen
Beschreibung: Durchführung der detaillierten Schritte unter Zuhilfenahme der Dokumentvorlage.
Ergebnis: System-Architektur-Spezifikation

Die detaillierten Schritte der zusammengesetzten Aktivität sind in der Abbildung 6.8 dargestellt.

Als erstes müssen die Systemelemente in der Elementübersicht definiert werden. Des Weiteren werden die Abhängigkeiten zwischen den Elementen spezifiziert. In der Regel unterliegt ein Server-System bestimmten Einschränkungen der Konfiguration, wie beispielsweise die maximale Anzahl der Elemente im Gesamtsystem. Diese Konfigurationseigenschaften sollten aus der Elementübersicht erkennbar sein.

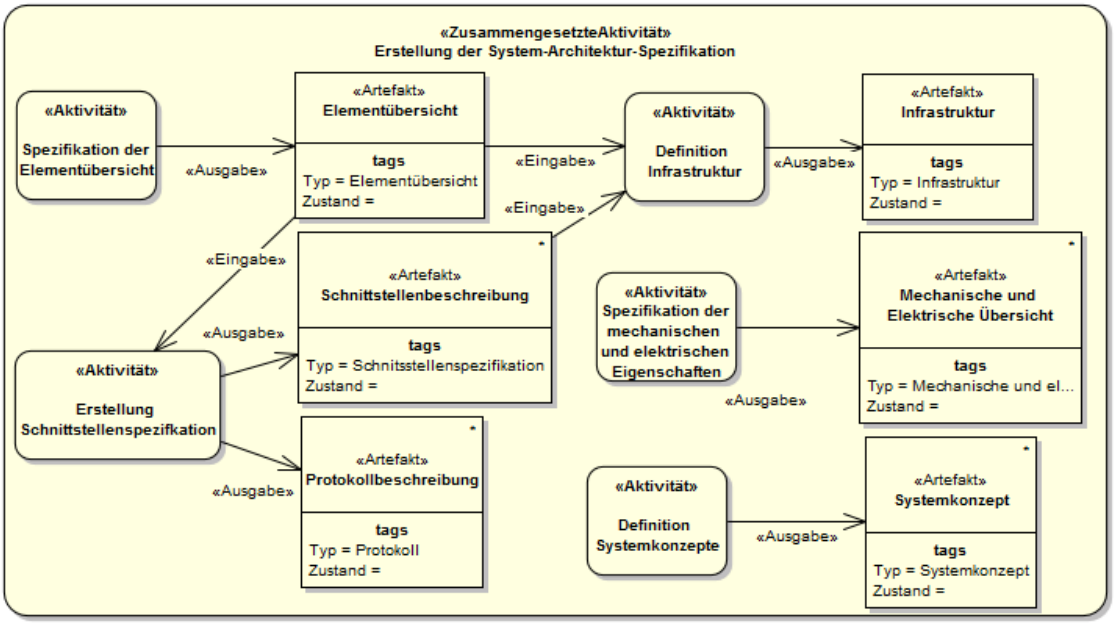


Abbildung 6.8: *Detaillierung der zusammengesetzten Aktivität Erstellung der System-Architektur-Spezifikation*

«Aktivität» Spezifikation der Elementübersicht

Beschreibung: Das Gesamtsystem wird als SysML-Block im SysML-Blockdefinition-Diagramm definiert. Die internen Hardwareelemente, inklusive Chassis, werden ebenfalls als SysML-Blöcke definiert. Zwischen den Elementen können alle Abhängigkeiten, die die SysML zwischen SysML-Blöcken erlaubt, modelliert werden.
Ergebnis: Elementübersicht

In dem weiteren Schritt der Schnittstellendefinition werden die Schnittstellen für jedes Element des Systems definiert.

«Aktivität» Erstellung Schnittstellenspezifikation

Eingabe: Elementübersicht
Beschreibung: Für jedes Element in der Elementübersicht werden SysML-Ports definiert. Jedem Port wird ein eindeutiger Name zugewiesen. Für ein SysML-Port werden die Kommunikationsrichtung sowie Daten- oder Kontrollflusseigenschaften definiert. Ein SysML-Port kann bei Bedarf um ein SysML-Interface erweitert werden, über dem konkrete Funktionen der Schnittstelle spezifiziert werden können. Weiterhin kann bei Bedarf für jede Schnittstelle eine Protokollbeschreibung in textueller Form erfolgen.
Ergebnis: Schnittstellenspezifikation, Protokollbeschreibung

Sind die Systemelemente mit ihren Schnittstellen festgelegt, erfolgt die Definition der Infrastruktur.

«Aktivität» Definition Infrastruktur
Eingabe: Elementübersicht, Schnittstellenbeschreibung
Beschreibung: Die Elementtypen, die im SysML-Blockdefinitions-Diagramm enthalten sind, werden in einem oder mehreren SysML-Internen-Block-Diagrammen instanziiert. Die Schnittstellen werden exemplarisch miteinander verbunden. Dabei wird überprüft, ob die Schnittstellendefinitionen der gewünschten Verbindungen zueinander passen. Die Gesamtheit der SysML-Internen-Block-Diagramme muss alle möglichen Systemkonfigurationen umfassen und definiert dann die komplette System-Infrastruktur.
Ergebnis: Infrastruktur.

Für jedes Element in der Elementübersicht müssen bekannte mechanische und elektrische Eigenschaften beschrieben werden.

«Aktivität» Spezifikation der mechanischen und elektrischen Eigenschaften
Beschreibung: Für jedes Element der Elementübersicht sollen konform zu den Vorgaben aus der Vorlage zur System-Architektur die entsprechenden Informationen erstellt werden.
Ergebnis: Beschreibung der mechanischen und elektrischen Eigenschaften.

Das Gesamtsystem muss geforderte Systemkonzepte umsetzen können. Die Beschreibung der Systemkonzepte in der Anforderungsspezifikation sind sehr abstrakt und müssen im Rahmen der System-Architektur-Spezifikation verfeinert werden.

«Aktivität» Definition Systemkonzepte
Beschreibung: Es soll ein SysML-Anwendungsfalldiagramm für jedes in den Anforderungen genannte Systemkonzept erstellt werden und dadurch die rudimentären funktionalen Anforderungen detailliert werden.
Ergebnis: Systemkonzept

Nachdem die System-Architektur-Spezifikation erstellt ist, muss überprüft werden, inwieweit diese Spezifikation für die Entwicklung des Systems ausreicht. Im Vergleich zur Situation vor der Methodenweiterentwicklung sind bereits detailliere Vorgaben und Spezifikationsinhalte verfügbar. In vielen Fällen können im Rahmen der System-Architektur-Spezifikation weiterführende Spezifikationsdokumente für die Entwicklung der einzelnen Elemente angegeben werden. In der Regel müssen die Spezifikationen der einzelnen Elemente jedoch durch eine

Modul-Architektur-Spezifikation weiter detailliert werden, damit eindeutige Vorgaben für Entwickler zur Verfügung gestellt werden können.

Die Modul-Architektur-Spezifikation umfasst die selben inhaltlichen Themen wie die System-Architektur-Spezifikation, nur ist der Fokus nicht auf das System mit seinen Elementen, sondern auf ein Modul mit dessen Elementen gerichtet. Die zu erledigenden Schritte in der zusammengesetzten Aktivität *Erstellung der Modul-Architektur-Spezifikation* sind die gleichen wie bei der System-Architektur-Spezifikation. Die Dokumentvorlage der Modul-Architektur-Spezifikation erklärt die für die Modulspezifikation zu erstellenden Inhalte. Als Sonderfall gilt die Spezifikation eines Verwaltungsmoduls. In den verschiedenen Forschungs- und Entwicklungsprojekten ist jeweils eine Architektur für ein Verwaltungsmodul entwickelt worden. Die Erkenntnisse aus den Projekten sind in einer Referenzarchitektur dokumentiert worden. Die Referenzarchitektur für Verwaltungsmodule ermöglicht weitergehende Vorgaben zur Spezifikation als bei anderen System-Modulen. Insbesondere die Festlegung auf bestimmte Techniken, beispielsweise die Server-System-Verwaltung mit dem Common Information Model (CIM), bringt konkrete Vorgaben mit sich.

In beiden Fällen, der Erstellung der System- und Modul-Architektur-Spezifikation, gibt es eine Guideline zur Verwaltung der erstellten Dokumente, die ebenfalls das Nutzungskonzept eines DMS zur Ablage der Dokumente festlegt.

«Aktivität» Erstellung der Modul-Architektur-Spezifikation

Eingabe: Anforderungsdokument im Zustand unterschrieben
Hilfsmittel: Dokumentvorlage Modul-Architektur-Spezifikation, Guideline Nutzung DMS, Referenzarchitektur Verwaltungsmodul, Werkzeug zur Modellierung von SysML-Diagrammen
Beschreibung: Durchführung der detaillierten Schritte unter Zuhilfenahme der Dokumentvorlage. In dem Fall der Spezifikation eines Verwaltungsmoduls eines Server-Systems ist die Referenzarchitektur einzuhalten.
Ergebnis: Modul-Architektur-Spezifikation

Im Fall der Fremd-Entwicklung eines Moduls wird dem Zulieferer Zugriff auf die Modul-Architektur-Spezifikation gewährleistet. Auf Basis dieser Dokumente muss der Zulieferer seine Kosten- und Aufwandsschätzung durchführen.

«Aktivität» Kosten- und Aufwandsschätzung

Eingabe: Modul-Architektur-Spezifikation
Beschreibung: Dem Zulieferer wird entsprechend der betrieblichen Vorgaben Zugriff auf die Modul-

Architektur-Spezifikation gewährleistet.

In der Zusammenfassung der angestrebten Soll-Methode ist zu erkennen, dass die Vielfalt der Dokumente im Vergleich zur ursprünglichen Ist-Methode reduziert worden ist. Auf Basis einer Analyse sind die Strukturen der Dokumente neu entwickelt und vereinfacht worden. Die Dokumente unterliegen strukturierten Sichten, einmal auf der System-Ebene und einmal auf der Modul-Ebene. Die Spezifikationsinhalte werden auf Basis der formalisierten Darstellungsform der SysML und konform zu vorgegebenen Dokumentvorlagen erstellt. Das ermöglicht zum einen eine vereinfachte Sicht auf das Gesamtsystem und dessen strukturellen Abhängigkeiten und zum anderen eine gewisse Überprüfung der Vollständigkeit der Spezifikationsinhalte. Das betrifft insbesondere die modellbasiert erhobenen Spezifikationsinhalte auf Basis der SysML. Die Verwaltung der Dokumente wird über ein Konzept zur Nutzung eines Dokumenten-Managements definiert. Die konkrete Realisierung der Soll-Methode wird in Abschnitt 7.1 erklärt.

6.1.3 Konzeption für das Anwendungsbeispiel der CIM-Entwicklung

Ausgangslage

Dieser Abschnitt stellt das Ergebnis der Konzeption für das Anwendungsbeispiel der CIM-Entwicklung vor. Die Ausgangslage für die einzelnen Weiterentwicklungsprojekte der CIM-Entwicklung wird im Abschnitt 5.2.4, mit dem Produktmodell (siehe Abbildung 5.15) und Prozessmodell (siehe Abbildung 5.16) der Ist-Methode der zweiten Iteration, vorgestellt. Darauf aufbauend sind mehrere Iterationen von Projekten, mit separaten Methodenanforderungen, für die CIM-Entwicklung festgelegt worden. Für jede Iteration ist die Phase *Konzeption* durchgeführt worden. Das in diesem Abschnitt vorgestellte Ergebnis ist die Soll-Methode der letzten Iteration und stellt somit das Endergebnis in den Forschungs- und Entwicklungsprojekten mit Bezug zur CIM-Entwicklung dar. Die Anpassungen am Produkt- und Prozessmodell werden für jede Iteration erläutert.

Zuerst wird das Produktmodell der Soll-Methode vorgestellt (siehe Abbildung 6.9).

Projekt 2

Im Projekt 2 ist das CIM-Modell um das Anforderungspaket erweitert worden.

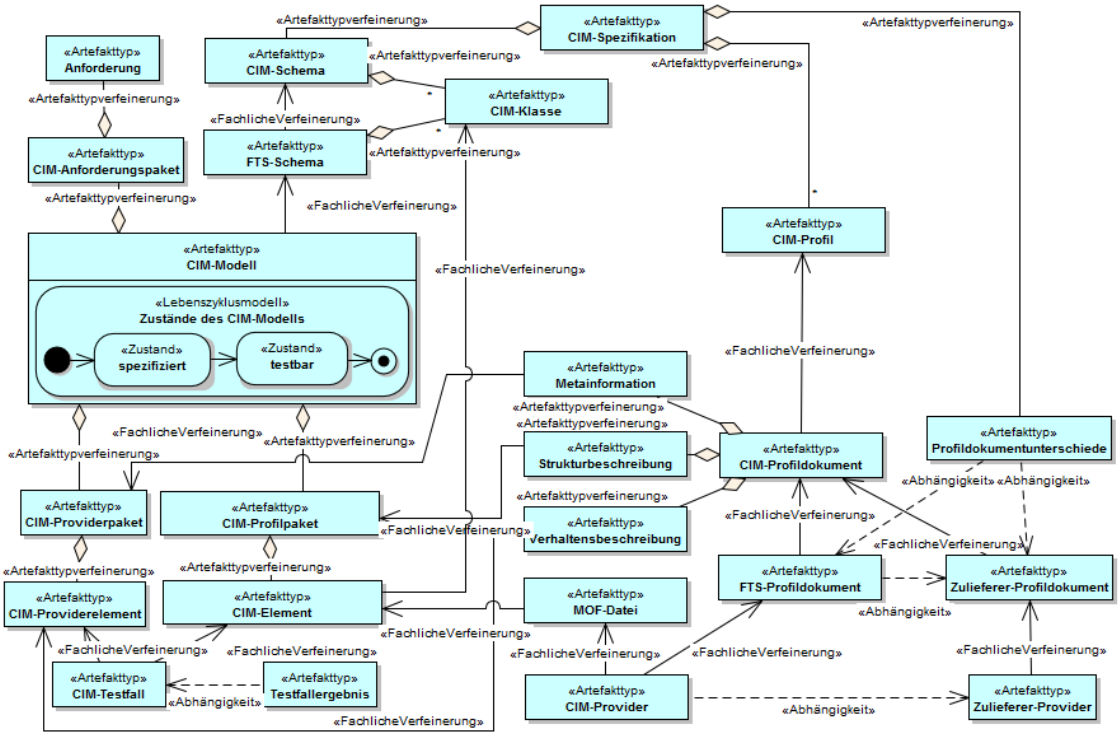


Abbildung 6.9: *Konzeption Produktmodell der Soll-Methode für das Anwendungsbeispiel der CIM-Entwicklung*

Dadurch wird es möglich einzelne Anforderungen innerhalb des Anforderungspakets im CIM-Modell zu erstellen und zu verwalten.

Projekt 3

Im Projekt 3 sind die Artefakttypen Zulieferer-Profildokument und Zulieferer-Provider dem Produktmodell hinzugefügt worden. Das sind die beiden Artefakttypen, die vom Zulieferer bereitgestellt werden. Da FTS die vom Zulieferer bereitgestellten CIM-Provider soweit wie möglich wiederverwenden möchte, und diese über die Profildokumente des Zulieferers spezifiziert sind, bestehen Abhängigkeiten zwischen den von FTS und vom Zulieferer bereitgestellten Profildokumenten und CIM-Providern. Weiterhin müssen die Unterschiede der beiden relevanten Profildokumente nachgehalten werden. In der Regel erweitert das FTS-Profildokument das Profildokument des Zulieferers. Das bedeutet, der CIM-Provider erweitert den Zulieferer-Provider. Es kann aber auch der Fall sein, dass Anpassungen am bestehenden Zulieferer-Provider notwendig sind, beziehungsweise, dass der zugelieferte Provider durch einen eigenen ersetzt werden muss. Es ist notwendig die Unterschiede des zugelieferten und des resultierenden FTS-Providers zu dokumentieren. Im Fall, dass der freigegebene Provider Fehler ent-

hält, ist es erforderlich den Verantwortlichen für die Fehlerbehebung zu identifizieren, entweder FTS oder der Zulieferer. Aus den dokumentierten Unterschieden zwischen den Profildokumenten ist ersichtlich, ob die Funktionalität vom Zulieferer programmiert worden ist oder ob FTS Anpassungen an dem Provider vorgenommen hat. Dieses Vorgehen beschleunigt die Fehlerzuweisung an FTS oder den Zulieferer. Die Alternative ist das Testen, erstens des FTS-Providers und zweitens des Zulieferer-Providers zur Ermittlung des fehlerhaften Programm-Codes. Das zweifache Testen ist aufwendiger als die Dokumentation und Durchsicht der Unterschiede. Für die Dokumentation der Unterschiede ist der Artefakttyp Profildokumentunterschiede im Produktmodell erstellt worden.

Projekt 4

In Projekt 4 sind lediglich Änderungen am Prozessmodell durchgeführt worden, das Produktmodell ist jedoch nicht angepasst oder erweitert worden.

Projekt 5

In Projekt 5 ist das Testkonzept entwickelt worden. Als neuer Artefakttyp ist der *Testfall* ermittelt worden. Es gibt eine Reihe unterschiedlicher Testfälle, sie können strukturelle Eigenschaften oder das Verhalten der CIM-Provider überprüfen. Sie werden zunächst auf Basis der Struktur- und Verhaltensbeschreibung der CIM-Profildokumente erstellt. Weiterhin ist definiert, wie das Ergebnis der ausgeführten Testfälle aussieht, dafür ist der Artefakttyp Testfallergebnis definiert worden.

Projekt 6

Das Projekt 6 hat sich mit der Erweiterung des CIM-Modells um die Metainformationen und Verhaltensbeschreibung befasst. Das CIM-Modell aggregiert daraus resultierend ein CIM-Providerpaket, welches die Metainformationen eines Profils enthält und die einzelnen Provider in Form von CIM-Elementen enthält. Mit Hilfe des CIM-Elements können Verhaltensinformationen im CIM-Modell modelliert werden. Die Informationen des CIM-Providerpakets und der CIM-Elemente werden, in ähnlicher Form wie die strukturellen Informationen des CIM-Profilpakets, in die textuelle Repräsentation des FTS-Profildokuments überführt. Dadurch entstehen die fachlichen Verfeinerungen zwischen Verhaltensbeschreibung und CIM-Providerelement sowie Metainformation und CIM-Providerpaket. Die automatisierte Überführung der Modell-Information in das FTS-Profildokument macht die Aktivität Schreiben des Profildokuments überflüssig.

Projekt 7

Projekt 7 führt dazu, dass die Testfälle ebenfalls aus dem CIM-Modell und nicht mehr manuell auf Basis der Profildokumente abgeleitet werden. Dazu zeigen die fachlichen Verfeinerungen des Testfalls auf die Elemente CIM-Element und CIM-

Providerelement. Darüber hinaus gibt es nun zwei explizite Zustände des CIM-Modells. In einem ersten Schritt werden die Spezifikationsinhalte definiert. In einem zweiten Schritt muss das CIM-Modell um Testdaten erweitert werden. Dafür ist das CIM-Modell um das Lebenszyklusmodell mit den Zuständen *spezifiziert* und *testbar* erweitert worden.

Aus Gründen der Übersicht sind die Artefakttypeigenschaften Konzepttyp und Sprache an dieser Stelle nicht dargestellt. Initiale Festlegungen der Eigenschaften sind in Abschnitt 5.2.4 beschrieben, die vollständige Definition der Eigenschaften ist dem Kapitel 7 zu entnehmen.

Neben dem Produktmodell ist auch das Prozessmodell weiterentwickelt worden (siehe 6.10).

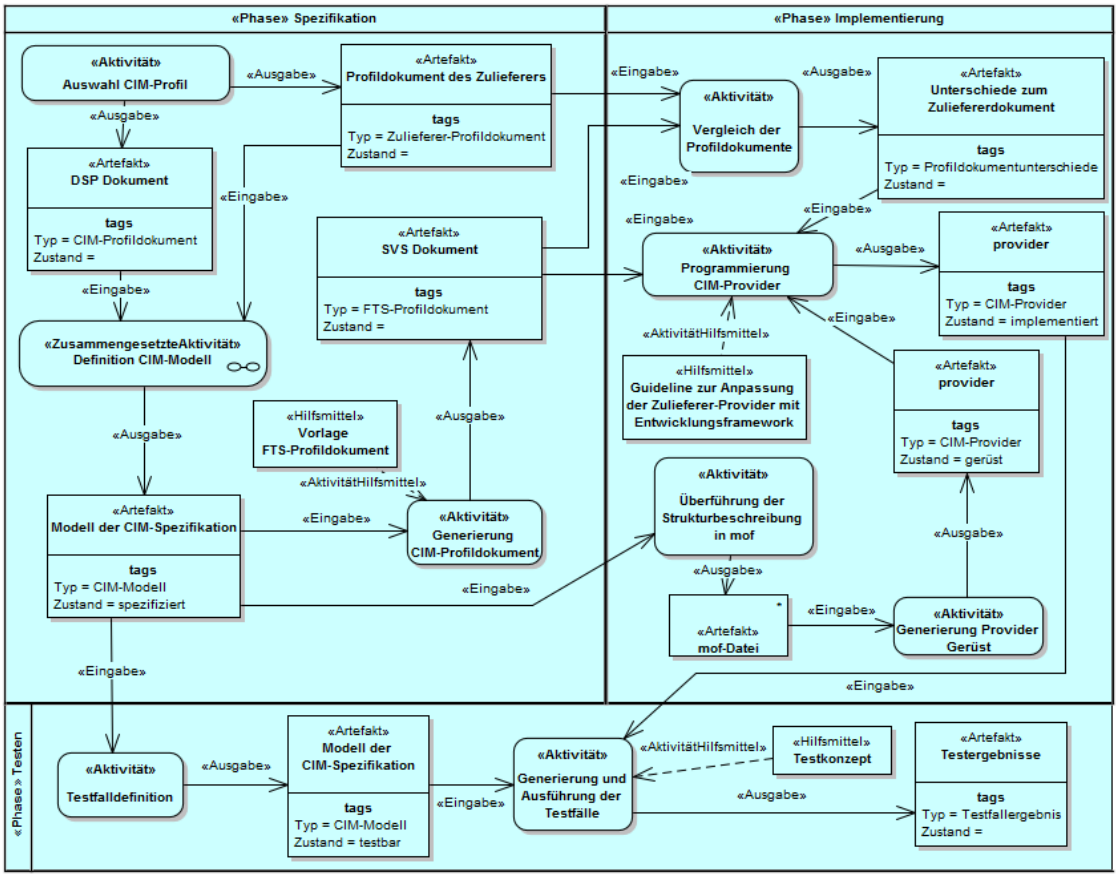


Abbildung 6.10: *Konzeption Prozessmodell der Soll-Methode für das Anwendungsbeispiel der CIM-Entwicklung*

Zusammenfassend sieht der integrierte Entwicklungsprozess für die CIM-Entwicklung nach der Durchführung der einzelnen Projekte die Phasen *Spezifikation*, *Implementierung* und *Testen* vor.

Phasen der CIM-Entwicklung

In der Phase *Spezifikation* wird zunächst das zu implementierende CIM-Profil ausgewählt.

Spezifikation

«Aktivität» Auswahl CIM-Profil
Beschreibung: Die Auswahl eines CIM-Profils besteht in der Aufgabe die Spezifikationen der DMTF zu durchsuchen, ob für die gewünschten Verwaltungsinformationen ein Standard-CIM-Profildokument vorhanden ist. Wenn ja, ist weiterhin zu überprüfen, ob der Zulieferer für das Profil bereits eine Unterstützung anbietet.
Ergebnis: Ausgewähltes CIM- und Zulieferer-Profildokument.

Sind die Vorgaben aus den vorhandenen Profildokumenten bekannt, wird das CIM-Modell spezifiziert. Dafür wird die zusammengesetzte Aktivität *Definition CIM-Modell* ausgeführt (siehe Abbildung 6.11).

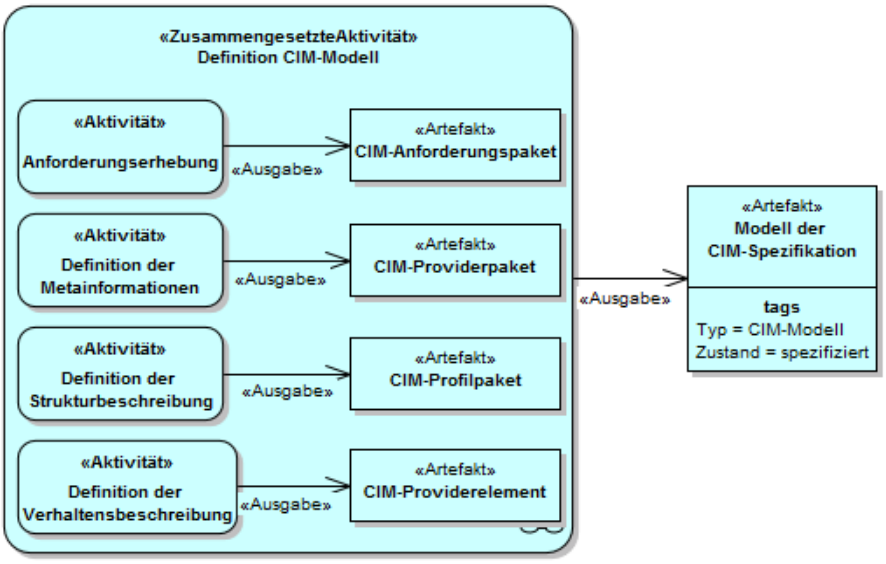


Abbildung 6.11: *Detaillierung der Zusammengesetzten Aktivität Definition CIM-Modell*

Die folgenden Aktivitäten *Anforderungserhebung*, *Definition der Metainformationen*, *Definition der Strukturbeschreibung* und *Definition der Verhaltensbe-*

schreibung werden konform zu den Vorgaben der Guideline zur Spezifikation des CIM-Modells durchgeführt.

«Aktivität» Anforderungserhebung, Definition der Metainformationen, Definition der Strukturbeschreibung, Definition der Verhaltensbeschreibung
Eingabe: CIM- und Zulieferer-Profildokument
Hilfsmittel: Guideline zur Spezifikation des CIM-Modells
Beschreibung: Die für die jeweilige Aktivität definierten Schritte der Guideline zur Spezifikation des CIM-Modells werden durchgeführt.
Ergebnis: CIM-Modell im Zustand spezifiziert

Ist das CIM-Modell vollständig spezifiziert, wird das FTS-Profildokument generiert. Das geschieht konform zur FTS-spezifischen Vorlage des Profildokuments.

«Aktivität» Generierung CIM-Profildokument
Eingabe: CIM-Modell
Hilfsmittel:Vorlage FTS-Profildokument
Beschreibung: Automatisierte Dokumenten-Generierung konform zur Vorlage
Ergebnis: FTS-Profildokument

Ist die Phase *Spezifikation* mit der Erstellung des FTS-Profildokuments für ein Profil abgeschlossen, wird in die Phase *Implementierung* gewechselt.

Implementierung

In der Phase *Implementierung* müssen die Artefakte MOF-Datei und CIM-Provider für jedes CIM-Element des CIM-Modells erstellt werden. Die MOF-Dateien werden aus den Informationen des CIM-Modells generiert.

«Aktivität» Überführung der Strukturinformationen in MOF
Eingabe: CIM-Modell
Beschreibung: Automatisierte Generierung der MOF-Dateien.
Ergebnis: MOF-Dateien

Aus den MOF-Dateien, die die strukturellen Informationen der CIM-Provider enthalten, können CIM-Provider im Zustand *Gerüst* generiert werden.

«Aktivität» Generierung Provider Gerüst
Eingabe: MOF-Datei
Beschreibung: Automatisierte Generierung der CIM-Providergerüste.
Ergebnis: CIM-Provider im Zustand Gerüst

Stehen die CIM-Provider im Zustand *Gerüst* zur Verfügung muss deren Verhalten implementiert werden. Dazu braucht der Programmierer konkrete Vorgaben, die er aus dem FTS-Profildokument herauslesen kann. Aufgrund der bestehenden Zuliefererthematik übernimmt er bereits eine vom Zulieferer bereitgestellte Basis-Implementierung und muss den CIM-Provider lediglich mit einem bereitgestellten Entwicklungsframework anpassen und erweitern. Dafür ist es sinnvoll dem Programmierer Zugriff auf die spezifizierten Unterschiede der CIM-Provider, die in den beiden Profildokumenten, des Zulieferers und FTS, definiert worden sind, zu geben. Aus Gründen der Fehlerzuweisung ist die Dokumentation der Unterschiede bereits unerlässlich. Die Dokumentation der Unterschiede wird vom Entwickler durchgeführt, der das FTS-Profildokument über das CIM-Modell spezifiziert. Diese Person hat in der Regel ein besseres Verständnis über die notwendigen Anpassungen an den zugelieferten Providern als der Programmierer und erkennt die Unterschiede bereits während der Spezifikationsaktivitäten.

«Aktivität» Vergleich der Profildokumente
Eingabe: FTS- und Zulieferer-Profildokument
Beschreibung: Tabellarische Darstellung der Unterschiede beider Profildokumente.
Ergebnis: Profildokumentunterschiede

Die Programmierung der CIM-Provider ist eine manuelle Tätigkeit des Programmierers. In einer Guideline zur Nutzung des Entwicklungsframeworks ist schrittweise erläutert, welche Tätigkeiten durchzuführen sind.

«Aktivität» Programmierung CIM-Provider
Eingabe: FTS-Profildokument, Profildokumentunterschiede
Hilfsmittel: Guideline zur Anpassung der Zulieferer-Provider mit Entwicklungsframework
Beschreibung: Durchführung der Schritte aus der Guideline zur Anpassung der Zulieferer-Provider mit Entwicklungsframework.
Ausgabe: CIM-Provider im Zustand implementiert

Ist die Programmierung der CIM-Provider für ein CIM-Profil abgeschlossen, Testen

muss dessen Qualität überprüft werden. Das ist in der Phase *Testen* definiert. Die gesamte Spezifikation der CIM-Provider ist im CIM-Modell hinterlegt. Daraus sind Testdaten ableitbar. Ein Testfall überprüft, ob ein implementierter CIM-Provider zur Laufzeit konform zu seiner Spezifikation agiert. Das bedeutet, ein CIM-Provider muss Instanzen der entsprechenden CIM-Klassen passend zu den Instanziierungsregeln erstellen, die Werte der CIM-Properties müssen im Rahmen ihrer Spezifikation erstellt werden und Methodenaufrufe auf dem CIM-Modell und Systemfunktionsaufrufe müssen entsprechend ihrem definierten Verhalten ausgeführt werden. Das CIM-Modell enthält bereits die entsprechenden Vergleichswerte und Verhaltensinformationen. Zur Durchführung der Testfälle müssen jedoch zusätzliche Testinformationen definiert werden, so dass ein Testwerkzeug diese Daten verwalten und über definierte Protokolle entsprechende Anfragen an den CIM-Server schicken kann, um deren Antworten mit den Testdaten zu vergleichen. Zur Definition dieser zusätzlichen Testinformationen wird eine Guideline zur Erweiterung des CIM-Modells um Testinformationen bereitgestellt.

«Aktivität» Testfalldefinition
Eingabe: CIM-Modell
Hilfsmittel: Guideline zur Erweiterung des CIM-Modells um Testinformationen
Beschreibung: Durchführung der Schritte aus der Guideline zur Erweiterung des CIM-Modells um Testinformationen.
Ausgabe: CIM-Modell im Zustand testbar

Liegt das entsprechende CIM-Modell mit allen Testinformationen vor, müssen die Testfälle ausgeführt werden.

«Aktivität» Generierung und Ausführung der Testfälle
Eingabe: CIM-Modell
Hilfsmittel: Testkonzept
Beschreibung: Ein Testwerkzeug übernimmt die zum Testkonzept konforme Generierung der Testfälle aus den Modellinformationen, Ausführung der Testfälle und Erstellung der Testfallergebnisse. Der Entwickler muss lediglich die CIM-Modellinformationen exportieren, diese dem Testwerkzeug zur Verfügung stellen und die Konfiguration des Testwerkzeugs für den Zugriff auf den zu testenden CIM-Server festlegen.
Ausgabe: Testfallergebnisse

Zusammenfassung

Die Entwicklungsmethode zur CIM-Entwicklung ist weitestgehend auf modellbasierten Konzepten aufgebaut worden. Das Ergebnis ist eine vollständige modellba-

sierte Spezifikation des CIM-Modells inklusive der Testinformationen. Darauf aufbauend können viele ursprünglich manuelleTätigkeiten automatisiert werden, beispielsweise die Dokumenten-Generierung sowie die Erstellung der Entwicklungsartefakte MOF-Datei und CIM-Providergerüst. Die Programmierung der Funktionalität der CIM-Provider bleibt eine manuelle Tätigkeit. Die Überprüfung der Funktionalität ist wiederum automatisiert durchführbar, nachdem die Testinformationen ebenfalls im CIM-Modell definiert werden. Im Vergleich zur ursprünglichen Entwicklungsmethode (siehe Abschnitt 3.6.3) sind die manuellen Tätigkeiten auf die Spezifikation des CIM-Modells reduziert. Die Spezifikationsinhalte müssen nur einmalig im CIM-Modell erstellt werden. Alle weiteren Sichten und Repräsentationen der Spezifikationsinhalte können automatisiert konvertiert werden. Die vorhandenen Probleme bei der manuellen Konsistenzsicherung redundanter Spezifikationsinhalte konnten eliminiert werden. Die modellbasierte Spezifikation bietet die Grundlage zur weiteren konstruktiven Qualitätssicherung mit Integritätsregeln zur Sicherstellung der korrekten Modellbildung. Die konkrete Realisierung der Soll-Methode wird in Abschnitt 7.2 vorgestellt.

6.2 Integration der Lösungsbausteine

Im Rahmen dieser Arbeit sind die Lösungsbausteine *Ermittlung der Ist-Methode*, *Ermittlung Methodenkontext*, *Verbesserungs-Analyse* und *Konzeption* für die Methodenweiterentwicklung erarbeitet worden. Dieser Abschnitt beschreibt die Integration der Prozess- und Produktmodelle der Lösungsbausteine (siehe Abbildung 6.12). Das Resultat stellt die Weiterentwicklung von MetaMe zu MetaMe++ dar.

Betrachten wir die Lösungsbausteine als Teilmethoden bedeutet dies, dass wir die Integration der Prozess- und Produktmodelle verschiedener Teilmethoden durchführen. Die Lösungsbausteine *Ermittlung der Ist-Methode* und *Ermittlung Methodenkontext* sind beides Elemente der Phase *Ist-Analyse* und werden darum zusammengefasst. Weiterhin werden die Prozessmodelle der *Ist-Analyse*, *Verbesserungs-Analyse* und *Konzeption* zusammengefasst. In dieser Arbeit wird kein Lösungsbaustein für die Phase *Realisierung* definiert. Die Schritte der Phase *Realisierung* sind abhängig von dem Entwurfsmodell und unterscheiden sich für die verschiedenen Methodenweiterentwicklungprojekte. MetaMe++ stellt keine methodischen Vorgaben für die Phase *Realisierung* bereit, beschreibt lediglich ihr Vorhandensein. Darum wird ein Platzhalter für diese Phase betrachtet.

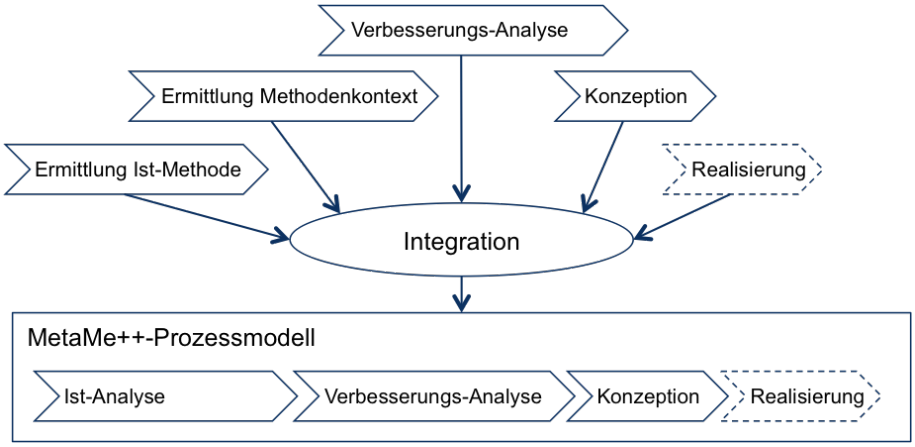


Abbildung 6.12: *Aufgabe: Integration der Prozessbestandteile der Lösungsbausteine*

Integration Prozessmodelle

Bei der Integration der Prozessmodelle müssen die Schnittstellen bei der Zusammenführung der Prozessmodelle zueinander passen. Bei der Entwicklung der Teilmethoden (einzelne Lösungsbausteine) in dieser Arbeit ist schon auf Kompatibilität der Schnittstellen der Teilmethoden geachtet worden. Als Schnittstellen werden die Ein- und Ausgabe-Artefakte einer Aktivität angesehen.

Die zusammengesetzte Aktivität *Ermittlung Ist-Methode* beschreibt den ersten Schritt der Methodenweiterentwicklung und definiert keine Eingabeartefakte. Das Ausgabe-Artefakt ist die Ist-Methode im Zustand validiert. Die zusammengesetzte Aktivität *Ermittlung Methodenkontext* definiert ebenfalls keine Eingabe-Artefakte. Die Ausgabe-Artefakte sind die gültigen Situationsfaktoren, die in ihrer Gesamtheit den Methodenkontext beschreiben.

In dem Lösungsbaustein *Ermittlung Methodenkontext* ist bereits eine *Nachfolger*-Abhängigkeit der beiden zusammengesetzten Aktivitäten definiert (siehe Abbildung 4.5) und die zusammengesetzte Aktivität *Ist-Analyse* beschrieben. Die Artefakte Ist-Methode und Methodenkontext werden wie im entsprechenden Ausschnitt des Produktmodells (siehe Abbildung 4.6 in Abschnitt 4.4.1) auf Typebene über den Artefakttyp Ist-Zustand aggregiert, wobei eine Instanz davon in der Ist-Analyse das Ausgabe-Artefakt darstellt.

Das einzige notwendige Eingabe-Artefakt für die Durchführung der zusammengesetzten Aktivität *Verbesserungs-Analyse* ist der Methodenkontext. Darum wird dieser Aktivität die *Ist-Analyse* vorangestellt werden. Das Ausgabe-Artefakt der *Verbesserungs-Analyse* ist die Definition ein oder mehrerer Projekte der Metho-

denweiterentwicklung, die wiederum als einziges Eingabe-Artefakt der zusammengesetzten Aktivität *Konzeption* definiert ist. Die beiden Aktivitäten können demnach auch nacheinander durchgeführt werden. Die Konzeption beschreibt als Ausgabe-Artefakt das Entwurfsmodell der Soll-Methode. Für den Platzhalter der Aktivität *Realisierung* wird das Entwurfsmodell als einziges Eingabe-Artefakt definiert.

Bei der Integration der Prozessmodelle können darüber hinaus weitere Beziehungen zwischen den einzelnen Aktivitäten definiert werden. Zur Verdeutlichung der Abarbeitungsreihenfolge werden die Nachfolger-Beziehungen für die relevanten Aktivitäten spezifiziert. Dadurch wird auch die iterative Methodenweiterentwicklung (siehe Abschnitt 5.2.4) definiert. Wenn sich der Methodenkontext und die Ziele nach einer Iteration ändern, müssen die Schritte Ist-Analyse und Verbesserungs-Analyse erneut ausgeführt werden. Bleiben beide Eigenschaften robust, wird die *Konzeption*, ohne vorheriges Ausführen der *Ist-Analyse* und *Verbesserungs-Analyse*, für ein weiteres Projekt durchgeführt. Das integrierte Prozessmodell wird um diese Abhängigkeiten erweitert. Darüber hinaus sind die Aktivitäten und Artefakte den entsprechenden Phasen *Ist-Analyse*, *Verbesserungs-Analyse*, *Konzeption* und *Realisierung* zugeordnet worden. Das Resultat wird in Abbildung 6.13 vorgestellt.

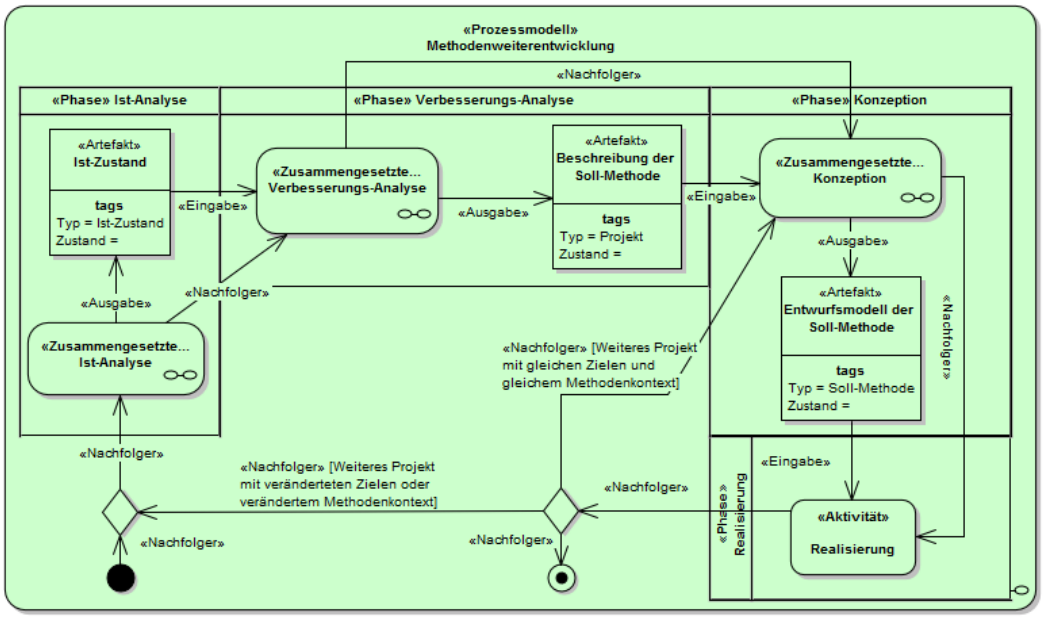


Abbildung 6.13: *Prozessmodell Methodenweiterentwicklung*

Integration der Produktmodelle

Bei der Integration der Teilmethoden müssen nicht nur die Prozessmodelle, sondern auch die Produktmodelle der Teilmethoden integriert werden. Dieser Sachverhalt wird in den nächsten Absätzen beschrieben.

Zuerst werden sämtliche Artefakttypen in einem Produktmodell zusammengefasst. Danach müssen die zusätzlichen Abhängigkeiten der Artefakttypen festgelegt werden. Aufgrund des Umfangs der Produktmodelle werden die entsprechenden Artefakttypen den strukturierenden Modell-Paketen Ist-Analyse, Verbesserungs-Analyse und Konzeption zugeordnet. Die Sprache MMMB definiert ein Methodenmetamodell. Dessen Elemente werden als Artefakttypen bei der Methodenweiterentwicklung benötigt. Die Artefakttypen, die im Methodenmetamodell definiert sind, werden dem Modell-Paket MMMB zugeordnet. Dabei werden die Elemente des Metamodells um die Erkenntnisse, die in den Lösungsbausteinen erarbeitet worden sind, erweitert. Durch die Integration der einzelnen Produktmodelle zu einem Produktmodell für MetaMe++ besteht die Möglichkeit die Abhängigkeiten der Artefakttypen Lösungsbaustein übergreifend zu definieren.

Das Ergebnis wird in der Abbildung 6.14 dargestellt.

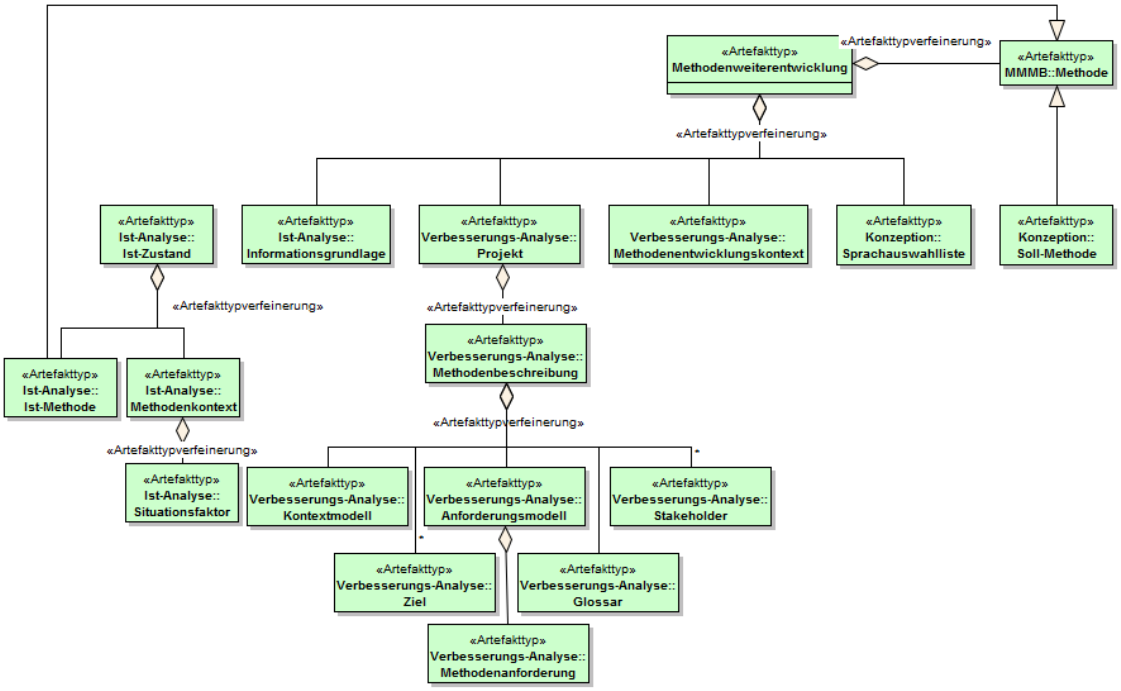


Abbildung 6.14: *Produktmodell Methodenweiterentwicklung*

Bei der Integration der Produktmodelle war es notwendig ein Wurzelelement zu definieren, damit sämtliche Artefakttypen ein aggregierendes Element im Modell der Methodenweiterentwicklung haben. Das ist ein technischer Aspekt bei der Entwicklung einer Werkzeugunterstützung. Dafür ist der Artefakttyp Methodenweiterentwicklung definiert worden.

Die Artefakttypen Ist-Methode und Soll-Methode erben von dem Artefakttyp Methode, der das Wurzelelement des Methodenmetamodells darstellt. Jedes Element des Methodemetamodells ist in einen Artefakttypen überführt worden. Das ist das Resultat einer konzeptionellen Selbstanwendung, bei dem der MetaMe++-Prozess auf sich selbst angewendet und die Methodenelemente des Methodenmetamodells als Artefakttypen definiert worden sind. Somit ist MetaMe++ über sein eigenes Produktmodell typisiert. Weiterhin ist durch die Anwendung von MetaMe++ auch eine Methode für ein weiteres Entwicklungsziel erstellt worden, die wiederum über das MetaMe++-Produktmodell typisiert ist. In der Abbildung 6.14 ist aus Gründen der Übersicht nur das Element Methode und nicht die weiteren Elemente der Sprache MMMB dargestellt.

Im Lösungsbaustein *Ermittlung Ist-Methode* (siehe Abbildung 3.19 in Abschnitt 3.6.1) ist definiert, dass die Ist-Methode in den Zuständen *erstellt* und *validiert* vorliegen kann. Im Lösungsbaustein *Konzeption* (siehe Abschnitt 6.1.1) ist ebenfalls die Validierung einer Soll-Methode diskutiert worden. Die Eigenschaft der Darstellung dieser Zustände wird allgemein auf das Element Methode des Produktmodells übertragen. Weiterhin sind in der Phase *Konzeption* die Zustände Domänenmodell und Produktmodell für das Element Produktmodell definiert worden. Ebenfalls sind die Zustände erstellt, vernetzt und konzeptionalisiert für das Element Artefakttyp definiert. Diese Eigenschaften sind ebenfalls auf die Elemente des Produktmodells übertragen worden.

In der Abbildung 6.15 ist dargestellt, wie das Methodenmetamodell in das Produktmodell der Methodenweiterentwicklung integriert ist. Es werden exemplarisch nur die drei Elemente Methode, Produktmodell und Artefakttyp dargestellt. Sie definieren die entsprechenden Zustände. Die Methode umfasst eine Struktur- und Verhaltensbeschreibung. Die strukturellen Elemente einer Methode werden im Produktmodell definiert. Das Verhalten wird über das Prozessmodell definiert, welches nicht in der Abbildung enthalten ist. Demnach beschreiben das Produktmodell und die Artefakttypen lediglich strukturelle Informationen. Als Sprache werden die Sprachelemente aus der MMMB-Definition angegeben. Im vorgestell-

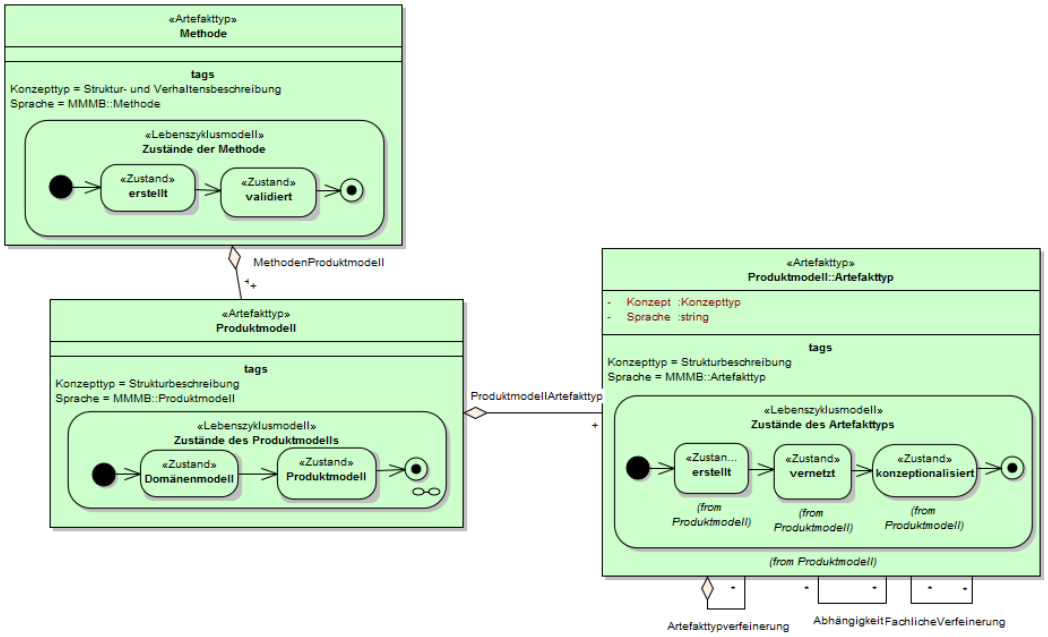


Abbildung 6.15: *Produktmodell Methodenweiterentwicklung (Ausschnitt MMMB)*

ten Produktmodell wird bereits die abstrakte Syntax der Sprachelemente vorgegeben. Das Element Methode im Produktmodell stellt bereits die abstrakte Syntax dar. Dieser Sachverhalt wird beschrieben, indem für die Eigenschaft Sprache der Eintrag MMMB::Methode definiert wird. Das Ganze gilt gleichermaßen für alle weiteren Elemente im MMMB-Paket des Produktmodells für MetaMe++ (bspw. MMMB::Produktmodell, MMMB::Artefakttyp). Die im Anhang A.3 definierte Semantik für die MMMB-Sprachelemente gilt auch für MetaMe++.

Zusammenfassend definiert dieser Abschnitt die Integration der Lösungsbausteine zum Gesamtkonzept MetaMe++ und umfasst das Prozessmodell, welches die Aktivitäten der Phasen *Ist-Analyse*, *Verbesserungs-Analyse*, *Konzeption* und *Realisierung* enthält, sowie das Produktmodell, welches die Artefakttypen, strukturiert über die Pakete Ist-Analyse, Verbesserungs-Analyse, Konzeption und MMMB enthält.

Sprachen

Für die Elemente im MMMB-Paket sind die Sprachen bereits definiert, für die weiteren Artefakttypen sind bisher nur Beispiele in den Anwendungsbeispielen bekannt. Die Definition der weiteren Sprachen wird im folgenden Abschnitt 6.3 vorgestellt.

6.3 Sprachfestlegung im MetaMe++-Produktmodell

Dieser Abschnitt beschreibt die Festlegung der Sprachen, für die einzelnen Artefakttypen für die bisher keine Sprache definiert worden sind. Dafür werden die für die Artefakttypen relevanten Erkenntnisse aus dieser Arbeit zusammengefasst und ein Vorschlag einer geeigneten Sprache für jeden Artefakttyp vorgestellt.

konkrete Syntax

Im Sinne von MetaMe++ beschreibt ein Element des Produktmodells bereits das abstrakte Sprachkonstrukt, also die abstrakte Syntax. Das bedeutet, das Element des Produktmodells muss bereits die inhaltlichen Strukturelemente definieren. Mit der Zuweisung einer Sprache wird die abstrakte Syntax detailliert und eine konkrete Syntax hinzugefügt. Im besten Fall wird ebenfalls die Semantik definiert, sofern die Sprache eine Definition der Semantik enthält.

MMMB

Für die MetaMe++-Artefakttypen im MMMB-Paket sind bereits Sprachen definiert. Die Ist-Methode aus dem Paket Ist-Analyse und die Soll-Methode aus dem Paket Konzeption erben von dem Artefakttyp *Methode* aus dem MMMB-Paket und übernehmen somit dessen Sprachen und zwar MMMB.

Für die übrigen Elemente wird in den folgenden Absätzen jeweils ein Vorschlag unterbreitet.

Ist-Analyse::Ist-Zustand Der Artefakttyp Ist-Zustand wird über die Artefakttypen Ist-Methode und Methodenkontext definiert. Als Sprache wird eine konkrete Syntax benötigt. Die vorgeschlagene Repräsentation des Ist-Zustands ist ein Dokument mit entsprechenden Abschnitten für die Ist-Methode und für den Methodenkontext. Aus den Erkenntnissen im Forschungs- und Entwicklungsprojekt benötigt ein Dokumentationsschriftstück neben den reinen inhaltlichen Informationen auch Strukturierungs- und Verwaltungsinformationen. Die konkrete Repräsentation des Ist-Zustands wird über eine Dokumentenvorlage mit der folgenden Struktur vorgegeben.

- Titel
- Inhaltsverzeichnis
- Einleitung
  - Version des Ist-Zustands
  - Revisionsverfolgung

  - Autoren des Dokuments
  - Leser des Dokuments
  - Inhaltsübersicht
  - Dokument-Eigenschaften
  - Referenzen
- Ist-Methode
- Methodenkontext

Der, um die zusätzlichen Eigenschaften erweiterte, Artefakttyp wird in Abbildung 6.16 vorgestellt.

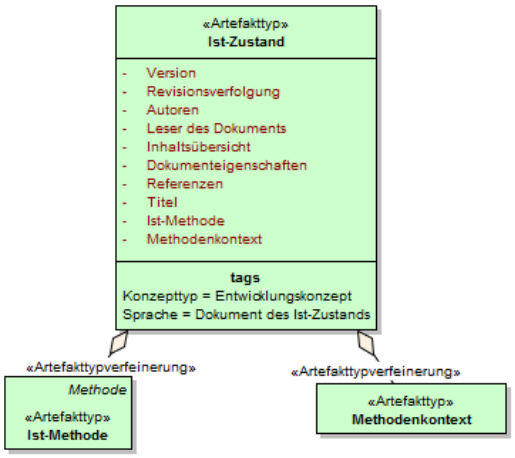


Abbildung 6.16: *Detaillierter Artefakttyp Ist-Zustand*

Ist-Analyse::Ist-Methode Die Ist-Methode wird mit der Sprache MMMB definiert. Die Definition des Ist-Zustands schlägt eine Darstellung als Dokument vor, die die Ist-Methode enthält. Demnach muss eine visuelle Darstellung der Ist-Methode, mit den Vorgaben der konkreten Syntax, möglich sein. Dafür sollen die erweiterten UML-Diagramme, Aktivitätendiagramm für das Prozessmodell und Klassendiagramm für das Produktmodell, genutzt werden.

Ist-Analyse::Methodenkontext Der Methodenkontext ist definiert über die gültigen Situationsfaktoren. Für den Artefakttyp Methodenkontext wird eine strukturierte schriftliche Repräsentation vorgeschlagen. Der Methodenkontext wird strukturiert über die Kategorien der Situationsfaktoren. Das bedeutet, für jede

Kategorie soll ein separater Abschnitt im Dokument des Ist-Zustands erstellt werden. Dieser Abschnitt enthält dann die jeweiligen gültigen Situationsfaktoren. Das bedeutet, der Artefakttyp Methodenkontext muss um den strukturellen Aspekt der Kategorie erweitert werden. Jeder Kategorie werden dann die Referenzen der jeweiligen gültigen Situationsfaktoren zugeordnet. Das soll in einer tabellarischen Darstellung erfolgen. Die Verfeinerung der abstrakten Syntax wird in Abbildung 6.17 vorgestellt.

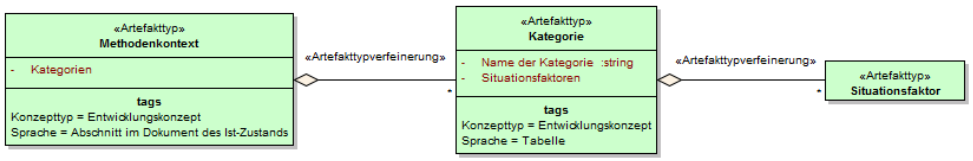


Abbildung 6.17: *Detaillierter Artefakttyp Methodenkontext*

Ist-Analyse::Situationsfaktor Ein Situationsfaktor wird in Abschnitt 4.3 als Tabellenelement mit Name und mit einem booleschen Wert für die Gültigkeit vorgestellt. Im Abschnitt 4.4.3 werden die Situationsfaktoren aufgrund einer schriftlichen Begründung für die Methodenweiterentwicklung als gültig definiert. Demnach benötigt der Artefakttyp Situationsfaktor die inhaltlichen Eigenschaften Name, Begründung und Gültigkeit. Für die konkrete Syntax wird eine schriftliche Darstellung, die passend für einen Tabelleneintrag in die Tabelle des Methodenkontexts ist, vorgeschlagen. Die abstrakte Syntax wird, wie in Abbildung dargestellt, festgelegt.

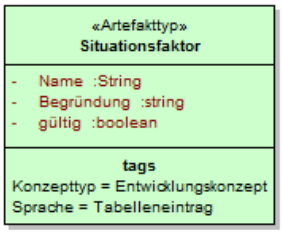


Abbildung 6.18: *Detaillierter Artefakttyp Situationsfaktor*

Ist-Analyse::Informationsgrundlage Der Abschnitt 3.6 stellt die Informationsgrundlage als Sammlung verschiedener Informationen, wie beispielsweise Dokumente, dar. An dieser Stelle ist es schwierig eine konkrete Syntax anzugeben,

denn eine Überführung der Informationen in eine explizite Darstellung als Informationsgrundlage rechtfertigt nicht den Aufwand. Die Informationsgrundlage wird hier demnach als nicht weiter definiertes Nachschlagewerk an Informationen definiert. Das bedeutet, ein Aktenordner, der gedruckte Exemplare der Informationen enthält, ist ebenso geeignet wie eine digitale Ablage der Informationen in einem Dokumenten-Management-System.

Verbesserungs-Analyse::Projekt Ein Projekt soll ebenfalls in einem Dokument verwaltet werden. Das Projekt besteht aus einer Methodenbeschreibung, benötigt jedoch ebenfalls wie der Ist-Zustand Strukturierungs- und Verwaltungsinformationen. Deshalb wird ebenfalls eine Dokumentvorlage ähnlich zum Ist-Zustand mit Titel und Einleitung vorgeschlagen. Synonym zum Ist-Zustand sieht die abstrakte Syntax des Projekts, wie in Abbildung 6.19 definiert, aus.

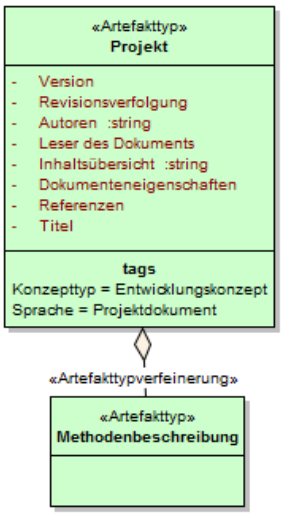


Abbildung 6.19: *Detaillierter Artefakttyp Projekt*

Verbesserungs-Analyse::Methodenbeschreibung Die Methodenbeschreibung aggregiert lediglich die Informationen Kontextmodell, Ziele, Glossar, Stakeholder und Anforderungsmodell. Die Methodenbeschreibung ist ein Abschnitt im Dokument des Projekts. Es werden keine weiteren Eigenschaften für die Methodenbeschreibung definiert. Für die konkrete Syntax wird eine Vorlage des Dokumentenabschnitts Methodenbeschreibung vorgestellt (siehe Abbildung 6.20).

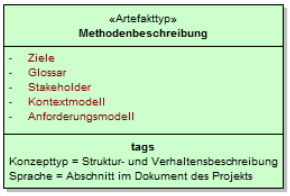


Abbildung 6.20: *Detaillierter Artefakttyp Methodenbeschreibung*

Verbesserungs-Analyse::Kontextmodell Das Kontextmodell besteht aus einem Prozessmodell, welches lediglich die Eingabe- und Ausgabe-Artefakte enthält, welche als Schnittstelle der Methode angesehen werden können. Diese Artefakte können definierten Phasen zugeordnet werden. Wie in weiteren Repräsentationen von Prozessmodellen, wird auch an dieser Stelle das für MMMB erweiterte UML-Aktivitätendiagramm vorgeschlagen (siehe Abbildung 6.21).

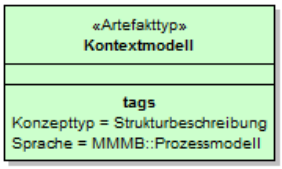


Abbildung 6.21: *Detaillierter Artefakttyp Kontextmodell*

Verbesserungs-Analyse::Ziel Für die Darstellung von Zielen wird ein Name zur Identifikation eines Ziels mit einhergehender detaillierter Beschreibung angegeben (vergleiche mit Abschnitten 5.2.2 und 5.2.3). Die abstrakte Syntax ist wie folgt definiert.

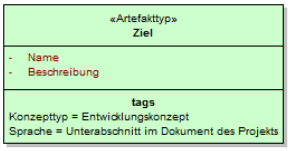


Abbildung 6.22: *Detaillierter Artefakttyp Ziel*

Verbesserungs-Analyse::Glossar Das Glossar beschreibt eine Menge für den Kontext zu definierender Begriffe. Das Glossar besteht aus einer Tabelle mit Tabelleneinträgen. Ein Tabelleneintrag ist ein Glossar-Eintrag. Ein Glossar-Eintrag be-

steht aus einem Begriff und einer Beschreibung. Der Vorschlag für die abstrakte Syntax wird in Abbildung 6.23 vorgestellt.

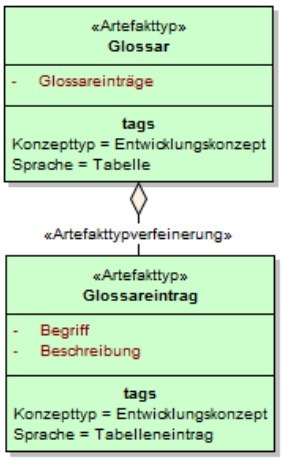


Abbildung 6.23: *Detaillierter Artefakttyp Glossar*

Verbesserungs-Analyse::Stakeholder Stakeholder der Methode sind definiert über Rollen, die von Personen oder Personengruppen eingenommen werden. Zur Darstellung dieser Informationen muss das Element Stakeholder um diese Informationen erweitert werden. Die abstrakte Syntax wird in Abbildung 6.24 definiert.

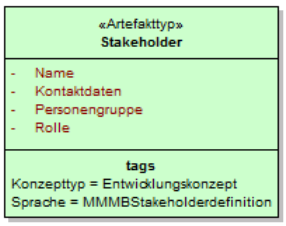


Abbildung 6.24: *Detaillierter Artefakttyp Stakeholder*

Als Sprache wird MMMBStakeholderdefinition genannt. Das ist eine Vorlage zur informellen Festlegung der relevanten Informationen. Bei einer konkreten Person können ein Name und die Kontaktinformationen angegeben werden. Bei einer Personengruppe soll ein Name für die Personengruppe angegeben werden, aber auch hier sollten Kontaktinformationen hinterlegt werden. Bei einer Rolle sollten

in der Organisation bekannte Rollendefinitionen genutzt werden, so dass auch hier eine Kontaktaufnahme möglich ist.

Verbesserungs-Analyse::Anforderungsmodell Das Anforderungsmodell darf mit beliebigen Sprachen erstellt werden, sofern die in Abschnitt 5.2.1 definierte Anforderung, dass Abhängigkeiten zwischen den Methodenanforderungen erstellt werden können, erfüllt werden kann. Die Systems Modeling Language (SysML) enthält die Definition eines Anforderungsdiagramms. Ein Anforderungsdiagramm stellt einzelne Anforderungen dar und erlaubt es verschiedene Relationen zwischen den Anforderungen zu visualisieren. Beispielsweise gibt es die Derive Requirement Relationship. Lässt sich eine Anforderung von einer anderen Anforderung ableiten, wird genau die Derive Requirement Relationship Relation zwischen den beiden Anforderungen erstellt (siehe [Obj12]). Im Rahmen der Methodenweiterentwicklung kann diese Relation genutzt werden, um die Abhängigkeiten der Methodenanforderungen zu beschreiben. Grundlegend definiert das Anforderungsdiagramm eine Sicht auf Anforderungen und deren Relationen. Als Vorschlag wird das Anforderungsmodell von der UML-Metaklasse `Package` abgeleitet. Ein `Package` enthält verschiedene Elemente, unter anderem auch UML-Klassen. Ein SysML-Requirement erweitert wiederum UML-Klassen, so dass das UML-Paket SysML-Requirement enthalten kann. Für die visuelle Repräsentation des Anforderungsmodells wird dann ein SysML-Anforderungsdiagramm genutzt, um es in dem Projekt-Dokument an entsprechender Stelle integrieren zu können. Auf diese Weise wird die abstrakte und konkrete Syntax des Anforderungsmodells über die UML-Paket-Definition beschrieben.

Verbesserungs-Analyse::Methodenanforderung Gemäß den Erläuterungen im vorigen Absatz müssen Methodenanforderungen von der UML-Klasse abgeleitet werden, so dass sie im Anforderungsmodell verwaltet werden können. Wollen wir eine Anforderung im SysML-Anforderungsdiagramm darstellen, muss sie sogar von der SysML-Metaklasse Requirement abgeleitet werden. Auf diese Weise übernimmt das Element Methodenanforderung unter anderem die strukturellen Attribute *id*, zur eindeutigen Identifikation einer Anforderung und das Attribut *text*, welches eine textuelle Beschreibung der Anforderung ermöglicht. Durch diese Festlegung sind semantische Eigenschaften im Anforderungsmodell definiert. Wie bereits erwähnt, können nun die für SysML-Requirements spezifizierten Relationen, wie beispielsweise die Derive Requirement Relation, zur Beschreibung

der Anforderungen und deren Beziehungen untereinander genutzt werden. Auf diese Weise wird die abstrakte und konkrete Syntax einer Methodenanforderung über die SysML-Requirement-Definition beschrieben.

Verbesserungs-Analyse::Methodenentwicklungskontext Für den Methodenentwicklungskontext gelten die gleichen Definitionen wie für den Methodenkontext, nur dass lediglich ein Teil der Kategorien (siehe Abschnitt 4.3.11) berücksichtigt werden muss.

Konzeption::Sprachauswahlliste Die Sprachauswahlliste ist ein Zwischenergebnis in der Phase *Konzeption*. Letztendlich werden mögliche Sprachen für die einzelnen Artefakttypen dokumentiert. Für die abstrakte Syntax bedeutet dies, dass Artefakttypen aus dem Produktmodell referenziert werden und ihnen Sprachen gegenübergestellt werden. Die Sprachen werden lediglich über einen aussagekräftigen Namen beschrieben.

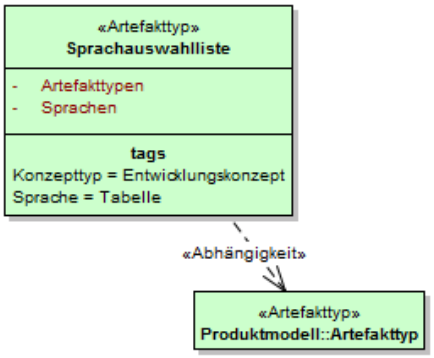


Abbildung 6.25: *Detaillierter Artefakttyp Sprachauswahlliste*

Für eine Dokumentation kann die folgende Vorlage zur Erstellung der konkreten Syntax genutzt werden.

Tabelle 6.1: *Vorlage zur Dokumentation der Sprachauswahlliste*

Artefakttyp	Mögliche Sprachen

6.4 Zusammenfassung

Mit Hilfe von MetaMe++ ist eine Methodenweiterentwicklung durchführbar. In den einzelnen Kapiteln dieser Arbeit sind die Lösungsbausteine für MetaMe++ entwickelt und exemplarisch auf die zwei Anwendungsbeispiele in einem konkreten Forschungs- und Entwicklungsprojekt angewendet worden. Der thematische Abschluss ist die Überführung der Methodenanforderungen aus der Verbesserungs-Analyse in die Konzeption der Soll-Methode für ein Projekt der Methodenweiterentwicklung (siehe Abschnitt 6.1). Dieses Vorgehen ist auf die Anwendungsbeispiele angewendet worden. Die Ergebnisse werden in den Abschnitten 6.1.2 und 6.1.3 vorgestellt. Des Weiteren sind die einzelnen Lösungsbausteine formal im Gesamtkonzept MetaMe++ integriert (6.2). Das Ergebnis ist ein MetaMe++-Prozessmodell und -Produktmodell. Für die Elemente des MetaMe++-Produktmodells ist in Abschnitt 6.3 jeweils ein Vorschlag unterbreitet worden, in welcher Form die Artefakttypen sprachlich und visuell dargestellt werden können. Der Vorschlag definiert Dokumente für die Beschreibung des Ist-Zustands, als Ergebnis der Phase *Ist-Analyse*, und für die Projekte, als Ergebnis der Verbesserungs-Analyse. Das Ergebnis der Konzeption ist ein Modell der Soll-Methode auf Basis der Sprache MMMB.

Ausblick

Im Rahmen der Methodenweiterentwicklung fehlt nun noch die Realisierung der Soll-Methode. Da eine solche Realisierung von den Konzeptions-Ergebnissen und den Methodenanforderungen abhängig ist, definiert MetaMe++ an dieser Stelle keine Vorgaben. Die Realisierung der Soll-Methoden der Anwendungsfälle wird exemplarisch im folgenden Kapitel thematisiert.

7 Realisierung der weiterentwickelten Methoden in den Anwendungsbeispielen

Dieses Kapitel stellt die Ergebnisse der Realisierung der Soll-Methode für das Anwendungsbeispiel System-Architektur-Spezifikation in Abschnitt 7.1 und für das Anwendungsbeispiel der CIM-Entwicklung in Abschnitt 7.2 vor.

Ausgangslage

In dieser Arbeit ist das Vorgehen MetaMe++ definiert worden. Es beschreibt die Methodenweiterentwicklung auf Basis der Konzepte von MetaMe. MetaMe++ definiert die Phasen *Ist-Analyse*, *Verbesserungs-Analyse* und *Konzeption*. Für die zwei Anwendungsfälle System-Architektur-Spezifikation und CIM-Entwicklung aus der Domäne der Server-System-Entwicklung sind die entsprechenden Soll-Methoden konform zum MetaMe++-Vorgehen definiert worden (siehe Kapitel 6). Sie beschreiben jeweils die Konzeption einer weiterentwickelten Methode mit ihren Produkt- und Prozessmodellen.

Aufgabe

Die Methodenweiterentwicklung beschreibt darüber hinaus die Phase *Realisierung*, in der die Soll-Methode implementiert wird, so dass sie auch anwendbar ist. Konkret bedeutet dies, dass die Hilfsmittel zur Verfügung gestellt werden müssen beziehungsweise, dass sie erstellt und entwickelt werden müssen. Die Hilfsmittel können unterschiedlicher Natur sein. In den Anwendungsbeispielen erkennen wir beispielsweise Guidelines, Dokumentvorlagen und Werkzeuge. Aufgrund der Vielfalt der durchzuführenden Aufgaben sind keine konkreten Anweisungen definierbar. Die Phase *Realisierung* ist geprägt von unterschiedlichen individuellen Aufgaben, die konform zum Konzept der Soll-Methode umgesetzt werden müssen.

7.1 Ergebnisse des Beispiels der System-Architektur-Spezifikation

Dieser Abschnitt beschreibt die realisierten Hilfsmittel für die Soll-Methode der System-Architektur-Spezifikation (siehe Abschnitt 6.1.2).

Aufgaben

Es werden die Dokumentvorlagen für die Anforderungsspezifikation sowie für die System- und Modul-Architektur-Spezifikation benötigt (siehe Abschnitt 7.1.1). Des Weiteren muss jeweils ein Werkzeug zur Anforderungsspezifikation (7.1.3) und zur Modellierung von SysML-Diagrammen (7.1.2) ausgewählt werden. Eine Vorgehensbeschreibung für den Anforderungsfindungsprozess unterstützt die Aktivität *Anforderungsspezifikation*. Die Spezifikation einer Referenzarchitektur wird bei der Spezifikation eines Verwaltungsmoduls benötigt. Eine Guideline zur Nutzung des DMSs muss erstellt werden. Damit geht einher, dass ein DMS zur Verfügung stehen muss.

In den folgenden Abschnitten werden die im Forschungs- und Entwicklungsprojekt bereitgestellten Hilfsmittel vorgestellt.

7.1.1 Dokumentvorlagen

Dokumentvorlagen

Die Dokumentvorlagen für die Spezifikationsdokumente agieren als Hilfsmittel zur Erstellung der Artefakte im Entwicklungsprozess. Es gibt sie für die Anforderungsspezifikation, System- und Modul-Architektur-Spezifikation. Die Dokumentvorlagen enthalten Schablonen zur Beschreibung eines oder mehrerer Artefakte.

Beispiel

Als Beispiel dient die Dokumentvorlage der System-Architektur-Spezifikation (siehe Anhang A.4). Sie definiert die Beschreibung der Systemkonzepte (Abschnitt 3.2 und Kapitel 5), der mechanischen und elektrischen Eigenschaften (Abschnitt 4.1), der Elementübersicht (Abschnitt 3.1), der Elemente (Kapitel 4), Schnittstellen (Abschnitt 3.3, 4.2, 4.3 und 4.4) sowie der Protokolle (Abschnitt 3.4) im System. Diese Aggregation von Inhalten findet sich auch im Produktmodell der Soll-Methode für die System-Architektur-Spezifikation wieder. Im Produktmodell sind bereits Sprachdefinitionen festgelegt worden, die in der Vorlage zu übernehmen sind.

Aufbau der Vorlage

Neben den Inhalten sind Meta- und Verwaltungsinformation auszufüllen, wie ein Glossar, Appendix, Veröffentlichungsanweisungen, Autoren, Inhaltsangaben, Referenzen, Einleitung und Weiteres, damit das Dokument in sich geschlossen ist und eventuell über Dokument-Management-Systeme verwaltet werden kann.

Anweisungen

Für jeden auszufüllenden Abschnitt sind Anweisungen oder Beispiele hinterlegt, so dass der Bearbeiter genaue Arbeitsvorgaben vorfindet.

7.1.2 SysML-Modellierungswerkzeug

SysML Werkzeug-unterstützung

In den Dokumentvorlagen werden SysML-Diagramme und SysML-Sprachelemente gefordert. Diese müssen erstellt werden und das ist ohne entsprechende Werkzeugunterstützung eine aufwendige Aufgabe. Darüber hinaus ist es für Anfänger im Bereich der Diagrammerstellung, ohne Möglichkeit der Modellüberprüfung, eine schwierige Aufgabe zum SysML-Standard konforme Diagramme zu erstellen.

SysML-Modelle

Ist der Entwicklungsprozess noch nicht so weit modellbasiert definiert, dass umfassende und vollständige SysML-Modelle als Artefakttypen definiert sind, muss ein Schritt dahingehend unternommen werden SysML-Modellierungswerkzeuge zu nutzen. Die Werkzeuge müssen neben der Diagrammerstellung auch die Informationen der Artefakttypen modellbasiert verwalten können. Dadurch wird neben der grafischen Darstellung auch die Verwaltung sowie eine maschinelle Überprüfung und Weiterverarbeitung der Inhalte möglich.

Evaluierung

Als Ergebnis einer umfassenden Evaluierung ist im konkreten Projekt das UML-Modellierungswerkzeug Enterprise Architect der Firma SparxSystems GmbH genutzt worden. Neben der Anforderung ein kommerzielles Produkt zu nutzen, damit ein robuster und durchgängiger Support gewährleistet ist, sprachen die geringen Kosten im Vergleich zu Konkurrenzprodukten, die Anpassungsmöglichkeiten und das im s-lab – Software Quality Lab vorhandene Know-how bei diesem Produkt für sich.

7.1.3 Werkzeug zur Anforderungsverwaltung

Für die Anforderungsverwaltung gibt es unzählige Softwareprodukte am Markt. Wiederum aufgrund des Know-hows im s-lab – Software Quality Lab mit dem

Werkzeug Enterprise Architect und dessen Anpassungsmöglichkeiten [FBGL+13] wird dieses Werkzeug auch für die Anforderungsdefinition vorgeschlagen. Das hat den Vorteil, bereits lizenzierte Versionen der Software zur Verfügung stehen zu haben.

7.1.4 Anforderungsfindungsprozess

Das eingesetzte Werkzeug zur Anforderungsverwaltung definiert keinen Prozess zur Anforderungsfindung. Es bietet lediglich die Möglichkeit zur Erhebung und Verwaltung der Anforderungen. Wie jedoch Anforderungen im Bereich der System-Architektur-Beschreibung ermittelt werden können, ist zu definieren.

Ergebnisse

Dieser Abschnitt fasst die Ergebnisse aus dem Abschlussbericht [Spi10] zusammen. In einer umfangreichen Evaluierung sind die folgenden Phasen für den Anforderungsfindungsprozess ermittelt worden, die hauptsächlich auf den Erkenntnissen aus [Rup09] basieren.

Ermittlung der Anforderungen: Bei der Ermittlung von Anforderungen müssen zuerst die Ziele an das Produkt, von den Stakeholdern des Produkts, festgelegt werden. Die Stakeholder müssen zuvor identifiziert werden. Des Weiteren ist es wichtig eine Kontextabgrenzung für das zugrunde liegende Projekt zu erstellen, denn es ist genauso wichtig zu wissen, was umgesetzt werden soll, wie was nicht umgesetzt werden soll. Mit Hilfe der Grundlagen muss eine für das Unternehmen passende Ermittlungsmethode vorgeschlagen werden. In dem beschriebenen Projekt hat sich die Kombination der Kreativtechnik *Brainstorming* und der Vergangenheits-orientierten Technik *Systemarchäologie* etabliert. Um das Brainstorming richtungsweisend durchzuführen, wird zuerst eine grobe Idee der Funktionalität mit grundlegenden Anwendungsfällen definiert. Diese Informationen sind bei der Strukturierung der Brainstorming-Ideen nützlich, um die Übersicht zu gewährleisten. Die Technik der Mindmaps ist zur Dokumentation der Brainstorming-Ideen genutzt worden. Mit Hilfe der Systemarchäologie kann danach sichergestellt werden, dass keine notwendige Funktionalität aus vorangegangenen Produkten *vergessen* wird.

Formulierung der Anforderungen: Die Anforderungen können auf unterschiedlichste Art und Weise formuliert werden. Von Prosa über formalisierte Sprachen mit den verschiedensten Abstraktionsgraden bis hin zu stark formali-

sierten Sprachen. Im Forschungs- und Entwicklungsprojekt ist auf die strukturierte Sprache bei der Definition der Darstellungsform, Regeln zur Erstellung der Anforderungen (siehe [Rup03]) sowie auf Anwendungsfalldiagramme und Ablaufdiagramme fokussiert worden.

Validierung der Anforderungen: Letztendlich unterliegen die ermittelten Anforderungen wieder Anforderungen, die eine Validierung der erstellten Inhalte ermöglichen. Darum sollen die Anforderungen die Eigenschaften Verfolgbarkeit, Redundanzfreiheit, Eindeutigkeit, Notwendigkeit, Korrektheit, Konsistenz, Prüfbarkeit, Vollständigkeit, Realisierbarkeit, Priorisierbarkeit, Verständlichkeit und Gültigkeit haben. Die ermittelten Anforderungen können auf diese Weise gegen ihre Soll-Eigenschaften validiert werden.

Verwalten der Anforderungen: Die Verwaltung der Anforderungen wird in dem Projekt durch das ausgewählte Werkzeug zur Anforderungsverwaltung sichergestellt.

Der für FTS vorgeschlagene Prozess wird in der Abbildung 7.1 skizziert.

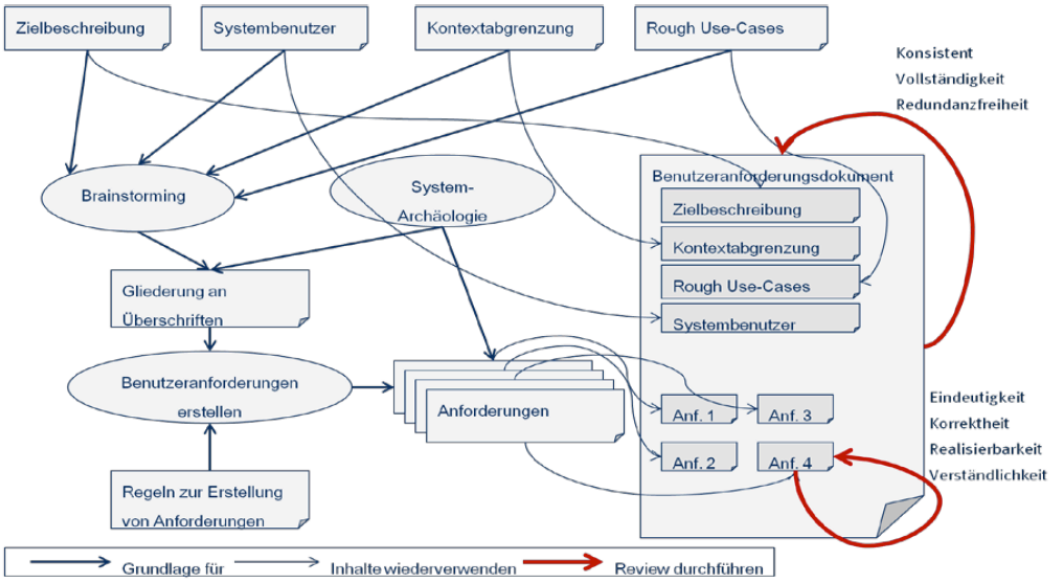


Abbildung 7.1: *Skizzierter Anforderungsdefinitionsprozess bei der System-Entwicklung*

Prozess zur Anforderungsfindung

In einem ersten Schritt werden die Grundlagen vom Projektleiter ermittelt. Das ist die Erstellung der Zielbeschreibung, Ermittlung der Stakeholder, die Kontextabgrenzung und die Definition der grundlegenden Anwendungsfälle (Rough-Use-Cases). Danach wird ein Brainstorming durchgeführt, bei dem eine Mindmap er-

stellt wird. Die Mindmap wird über die grundlegenden Anwendungsfälle strukturiert. In einem weiteren Schritt wird mit Hilfe bekannter Spezifikationsdokumente die Mindmap um notwendige Funktionalität (Systemarchäologie) erweitert. Für die Formulierung der Anforderungen wird sich an den Vorgaben des Werkzeugs zur Anforderungsverwaltung gehalten. Die im Werkzeug erfassten Anforderungen werden in ein Anforderungsdokument überführt. Die Gliederung des Dokuments ist auf Basis der bekannten Quelle *Mastering the Requirements Process* (siehe [RR99]) erstellt worden. Das Dokument ist unterteilt in einen ersten strukturellen Abschnitt, welcher die Grundlagen rekapituliert, und in einen zweiten strukturellen Abschnitt, welcher die Anforderung gemäß ihrer Gliederung aus dem Brainstorming vorstellt. Zum einen können einzelne Anforderungen entgegen ihren Eigenschaften Eindeutigkeit, Korrektheit, Realisierbarkeit und Verständlichkeit begutachtet werden und zum anderen das gesamte Dokument entgegen den Eigenschaften Konsistenz, Vollständigkeit und Redundanzfreiheit.

Erfolge

Das vorgestellte Vorgehen ist bei der Produktentwicklung der BX400 und der CX1000 angewendet worden. Die entstandenen Anforderungsspezifikationen werden bis heute als Best-Practices angesehen.

7.1.5 Referenzarchitektur Verwaltungsmodul

Referenzarchitektur

In dem Fall der Entwicklung eines Verwaltungsmoduls für eine System-Architektur-Spezifikation war eine Aufgabe die Bereitstellung einer Referenzarchitektur für Verwaltungsmodule. Durch Benutzung der Referenzarchitektur ist es einfacher eine entsprechende Modul-Architektur-Spezifikation für ein Verwaltungsmodul in einem konkreten Kontext zu erstellen. Bei der Erstellung des Referenzarchitektur-Dokuments ist die Vorlage für eine Modul-Architektur-Spezifikation wiederverwendet und konkretisiert worden. Das Ergebnis der Realisierung der Referenzarchitektur für Verwaltungsmodule ist ein Dokument mit dem Namen *Architecture Specification for the MMP*. MMP bedeutet Management Platform und beschreibt den Oberbegriff für Verwaltungsmodule in unterschiedlichen Systemarchitekturen. Das Dokument umfasst eine Definition der unterschiedlichen Architekturen (siehe Abbildung 7.2, Nr. 1.). Aufgeführt sind die farblich unterschiedlichen Kontexte, in denen ein Verwaltungsmodul eingebettet werden kann. Diese Kontexte sind ähnlich zu den Varianten der Management-Systeme, die bereits in Kapitel 3.4 definiert worden sind.

Bei Benutzung der Referenzarchitektur zur Erstellung der Modul-Architektur-Spezifikation für das Verwaltungsmodul muss in einem ersten Schritt der vorliegende Kontext ermittelt werden.

Zweitens werden von der Referenzarchitektur interne und externe Schnittstellen vorgegeben (siehe Abbildung 7.2, Nr. 2.).

Drittens stellt die Referenzarchitektur die grundlegenden Bausteine eines Verwaltungsmoduls in einem SysML-Blockdefinitions-Diagramm bereit (7.2, Nr. 3.).

Viertens wird die Architektur in einem internen Block-Diagramm der SysML vorgegeben (7.2, Nr. 4.).

In allen 4 Schritten muss sich der Entwickler bei der Erstellung der Modul-Architektur-Spezifikation nach den Vorgaben richten. Er übernimmt, erweitert oder passt die Definitionen der Referenzarchitektur an. Die Referenzarchitektur dient also als erweiterte Vorlage, gibt gleichermaßen auch entsprechende Anforderungen vor.

7.1.6 Dokument-Management-System

Dokument-Management-System

Die Etablierung eines Dokumenten-Management-Systems war aufgrund einer nicht genehmigten Einführung eines neuen Werkzeugs schwierig. Für das Forschungs- und Entwicklungsprojekt wäre der Aufwand, ein neues Werkzeug firmenweit einzuführen, zu groß gewesen. Eine Insellösung für ein prototypisches Projekt ist abgelehnt worden. Es musste sich mit bestehender Software beholfen werden. Die Wahl fiel dann auf Microsoft Sharepoint, ein Web-basiertes Intranetportal für die verbesserte Zusammenarbeit im Team. Sharepoint ermöglicht die Web-basierte Dokumentenablage und Dokumentenversionierung. Bei FTS werden nun Sharepoint-Seiten für einzelne Projekte erstellt und dort die Dokumente abgelegt, und nicht mehr auf den Firmen-weiten Dokumentenservern. Das reduziert den Such- und Verwaltungsaufwand für Dokumente. Für ein einzelnes Projekt muss sich jeweils eine passende Ablagestruktur überlegt werden. Die Vielzahl der Dokumentenversionen wird durch das integrierte Versionierungskonzept reduziert.

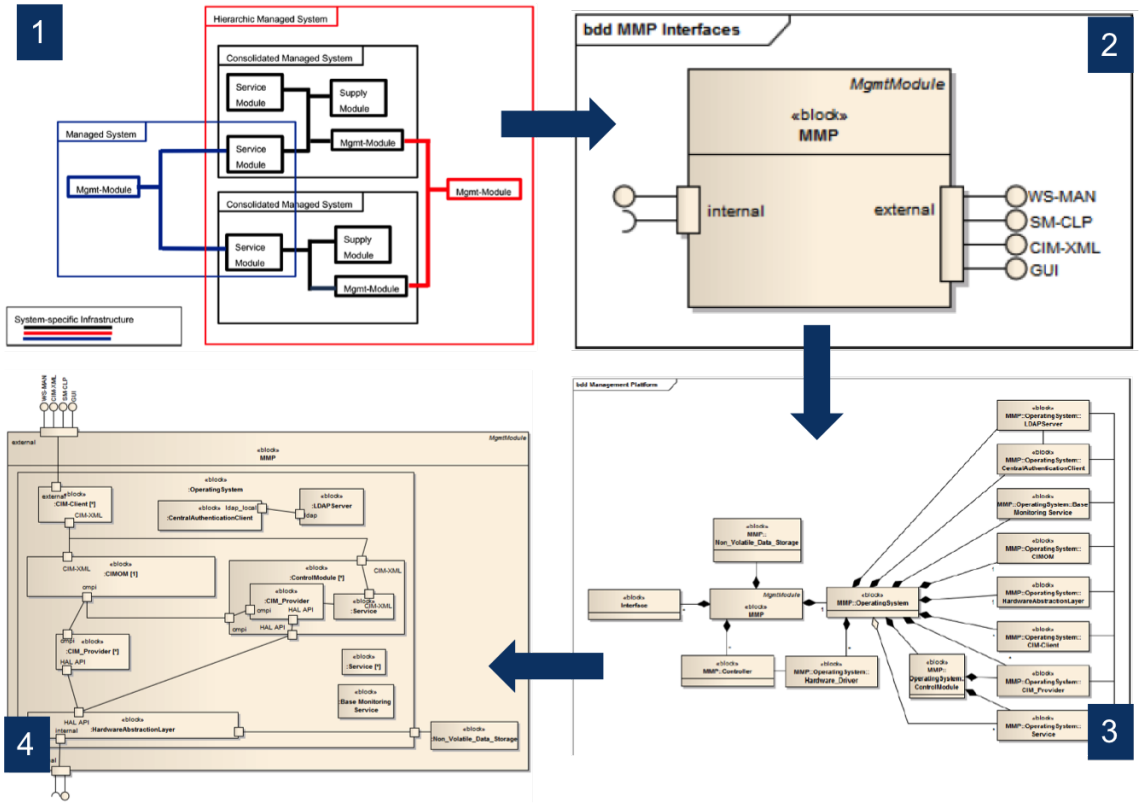


Abbildung 7.2: *Inhalte des Referenzarchitektur-Dokuments der MMP*

7.1.7 Zusammenfassung System-Architektur-Spezifikation

Der umfangreiche Entwicklungsschritt der System-Architektur-Spezifikation ist durch die Realisierung der Weiterentwicklung grundlegend verbessert worden. Sämtliche Schritte von der Produktdefinition und Spezifikation sind über einzelne Prozessaktivitäten definiert worden. Die einzelnen Aufgaben können aufgrund der definierten Entwicklungsmethode beliebig wiederholt werden. Jede Aktivität ist dokumentiert und mit Dokumentvorlagen, Anweisungen, Hilfsmitteln und Werkzeugen angereichert.

modellbasiert

Auch wenn der Entwicklungsprozess der System-Architektur-Spezifikation stark Dokumenten-zentriert ist, was zum Teil durch den darüber liegenden Management-Prozess vorgegeben ist, wird an den wichtigen Stellen ein großer Teil der Spezifikationsinhalte modellbasiert und Werkzeug-unterstützt auf Basis der SysML erstellt. Das birgt Potenzial, weitere Verbesserungen in Richtung der

modellbasierten Entwicklung zu unternehmen. Anschließende Methodenweiterentwicklungen können wiederum mit MetaMe++ durchgeführt werden.

7.2 Ergebnisse des Beispiels der CIM-Entwicklung

Dieser Abschnitt beschreibt die Realisierung der Soll-Methode der CIM-Entwicklung (siehe Abschnitt 6.1.3).

Im Vergleich zum vorangegangen Abschnitt, bei dem es hauptsächlich um eine Werkzeugauswahl und Bereitstellung von Dokumentvorlagen ging, ist der CIM-Entwicklungsprozess bereits so detailliert, dass es mehr um technische Details geht. Er beschreibt die Aktivitäten in einem späteren Zeitpunkt im Entwicklungsprozess, bei dem es detailliertere Vorgaben gibt und auch detailliertere Entwicklungsartefakte definiert sind.

modellbasierte Spezifikation CIM-Schema

Die Aufgabe ist die Erstellung der modellbasierten Spezifikation des CIM-Datenmodells (CIM-Schema), dessen Dokumentation und die automatisierte Überführung des CIM-Datenmodells in Entwicklungsartefakte. Die Erstellung des Modells der CIM-Spezifikation stellt die Basis für weitere Aufgaben dar (siehe Abschnitt 7.2.1).

Ergebnisse

Die Dokumentation des CIM-Schemas wird aufgeteilt in die verschiedenen spezifizierten CIM-Profile. Für die Dokument-basierte Spezifikation eines CIM-Profils wird eine Vorlage des FTS-Profildokuments benötigt. Die Vorlage stellt die Formatvorlage für die Dokumentationsdokumente dar, die automatisiert aus der modellbasierten CIM-Spezifikation erstellt werden (siehe Abschnitt 7.2.2). Ebenso werden die MOF-Dateien aus den modellbasierten Informationen erstellt. Mit einem weiteren Werkzeug werden anschließend aus den MOF-Dateien CIM-Provider-Skeletons erzeugt (siehe Abschnitt 7.2.3). Zur Erweiterung der CIM-Provider-Skeletons, um das spezifizierte Verhalten, muss eine Guideline für die Benutzung des, von einem Zulieferer bereitgestelltem, Entwicklungsframeworks erstellt werden (siehe Abschnitt 7.2.4). Abschließend wird die Realisierung der Testumgebung beschrieben, die automatisiert Testfälle erzeugt und auswertet (siehe Abschnitt 7.2.5).

7.2.1 Modell der CIM-Spezifikation

Zur Erstellung eines Modells der CIM-Spezifikation wird im Sinne der modellbasierten Entwicklung ein Metamodell der abstrakten Syntax, eine konkrete Syntax und die Semantik benötigt. Mit Hilfe dieser Informationen ist eine DSL definiert. Es gibt Werkzeuge und Techniken, mit denen aus der Definition der DSL ein Modellierungswerkzeug erstellt werden kann. Mit einem solchen Werkzeug wäre die Erstellung der CIM-Spezifikation möglich. Die grundlegenden Informationen für die DSL sind bereits von der DMTF definiert (siehe beispielsweise [Dis10a]).

Nachteile DSL

Dieses Vorgehen hat einige Nachteile. Es würde eine Werkzeugentwicklung mit sich ziehen, die Ressourcen bei der Entwicklung und Betreuung benötigt. Aufgrund der regelmäßigen Anpassung an dem von der DMTF vorgeschlagenen Metamodell, circa eine entscheidend geänderte Version pro Jahr, entsteht ein regelmäßiger Änderungsaufwand an dem potenziellem Werkzeug. Es bestehen weitere Anforderungen an das Werkzeug. Beispielsweise eine Multi-User-Unterstützung mit integrierter Versionsverwaltung, also eine technisch anspruchsvolle Aufgabe. Aus diesen Gründen ist eine Eigenentwicklung ein großer Investitionsaufwand. Eine weitere fachliche Schwierigkeit ist eine notwendige proprietäre Anpassung des vorgeschlagenen Metamodells. Die DMTF berücksichtigt keine CIM-Profile im Metamodell, die in der praktischen Anwendung jedoch ein unverzichtbares Strukturierungselement darstellen.

UML-Profile

Das CIM-Schema weist große Ähnlichkeiten mit einem UML-Klassendiagramm auf. Es besteht aus CIM-Klassen und CIM-Assoziationen (siehe auch Abschnitt 3.3.3). Einige Elemente des CIM-Schemas können direkt auf UML-Elemente abgebildet werden. Demnach ist die Erstellung eines UML-Profils für die Erstellung des CIM-Schemas als weitere Lösung in Betracht gezogen worden.

Vorteile UML-Profile

Das Werkzeug Enterprise Architect ist bereits im Unternehmen bekannt. Es ist ein UML-Modellierungswerkzeug, welches die Möglichkeit besitzt UML-Profile zu definieren und Modelle gemäß eines UML-Profils zu erstellen. Der Aufwand zur Erstellung und Verwaltung eines UML-Profils erscheint wesentlich geringer, als die Entwicklung eines eigenen neuen Werkzeugs. Ein weiterer Vorteil ist die Tatsache, dass nicht das ganze Metamodell in seinem kompletten Umfang unterstützt werden muss, sondern nur die Elemente, die tatsächlich gebraucht werden. Die Anforderung der Multi-User-Unterstützung mit integrierter Versionsverwaltung sowie weiterer Schönheitsfeatures sind im EA bereits implementiert.

MetaMe-Eigenschaft

Bei der Aufgabe der Erstellung des UML-Profils hilft weiterhin die Eigenschaft von MetaMe, dass das Produktmodell der Entwicklungsmethode der CIM-Entwicklung das Metamodell der Spezifikation darstellt (siehe Abbildung 2.13). Der in der Abbildung 7.3 dargestellte Ausschnitt aus dem CIM-Produktmodell (siehe 6.9), welches das CIM-Modell beschreibt, stellt die relevanten Metamodellelemente für die Erstellung des UML-Profils dar.

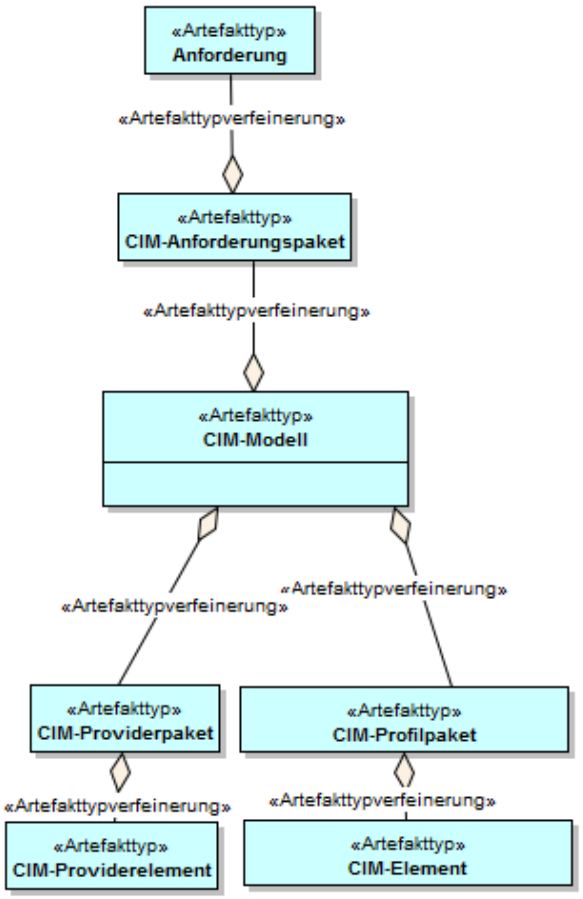


Abbildung 7.3: *Ausschnitt des CIM-Produktmodells mit den relevanten Elementen des CIM-Modells*

Detaillierung

Das dargestellte Metamodell ist in Teilen noch zu abstrakt. Die Metamodell-Elemente CIM-Providerelement und CIM-Element müssen noch weiter detailliert werden. In der Abbildung 7.3 ist jedoch die separate Darstellung von CIM-Providern (CIM-Providerelement) und CIM-Klassen (CIM-Element) zu erkennen. Die nächst höhere Strukturierungsebene ist das CIM-Profil, welches einmal für CIM-Provider (CIM-Providerpaket) und CIM-Klassen (CIM-Profilpaket) dargestellt

wird. Diese strukturelle Trennung ist im DMTF-Metamodell nicht vorgesehen. Wir sehen hier, dass durch Zuhilfenahme des CIM-Produktmodells auch sichergestellt wird, dass eine Lösung auf die bereits herausgestellten Bedürfnisse von FTS zugeschnitten wird, denn insbesondere im definierten CIM-Entwicklungsprozess ist die Strukturierung über CIM-Profile notwendig. Die Situationsgerechtigkeit wird berücksichtigt und genau das soll bei der Anwendung von MetaMe++ der Fall sein.

Realisierung

Eine Aufgabe in der Realisierung des Modells der CIM-Spezifikation ist die Erstellung des UML-Profils. Dafür liegt das CIM-Produktmodell aus der Methodenweiterentwicklung vor, welches um die CIM-spezifischen Eigenschaften des DMTF-Metamodells (CIM-Meta-Schema) detailliert werden muss (siehe Abbildung 3.7). Das Ergebnis wird exemplarisch dargestellt. Zuerst wird das CIM-Element aus dem CIM-Produktmodell um die Metamodell-Elemente CIM-Klasse und CIM-Assoziation konform zum CIM-Meta-Schema erweitert. CIM-Klassen können CIM-Properties enthalten und CIM-Referenzen sind spezialisierte CIM-Properties, wobei diese nur CIM-Assoziationen zugewiesen sein können[1]. Das detaillierte CIM-Produktmodell sieht in den vorgestellten Teilen aus, wie in Abbildung 7.4 dargestellt. Die Möglichkeit der Nutzung von Generalisierungsbeziehungen (Vererbung) im Produktmodell ist möglich, da MMMB ebenfalls als UML-Profil definiert ist.

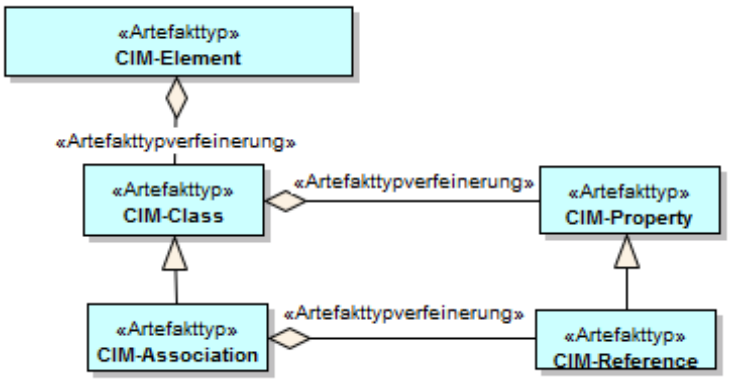


Abbildung 7.4: *Detailliertes CIM-Produktmodell*

Erweiterung

Darüber hinaus müssen semantische und strukturelle Festlegungen, die von

1 An dieser Stelle ist das vorgestellte CIM-Meta-Schema nicht passend modelliert. Die Definition im detaillierten CIM-Datenmodell ist pragmatisch und nicht CIM-Meta-Schema-konform durchgeführt worden.

der DMTF in rein textueller Form definiert worden sind, dem detaillierten CIM-Produktmodell hinzugefügt werden. Das detaillierte CIM-Produktmodell muss dann für die UML-Profil-Erstellung auf das UML-Modell abgebildet werden und die (UML-Profil-)Erweiterungen definiert werden. Im Beispiel wird das Element CIM-Klasse auf das Element UML-Klasse abgebildet. Ein CIM-Property übernimmt in einer CIM-Klasse die gleiche Aufgabe wie ein UML-Attribut in einer UML-Klasse, eine CIM-Reference definiert die Assoziationsenden einer CIM-Assoziation. Auch in diesem Fall wird es als UML-Attribut einer UML-Klasse angesehen. Wir bilden CIM-Property und CIM-Reference auf UML-Attribute ab.

In der Definition eines UML-Profils mit dem Werkzeug Enterprise Architect sieht das Ergebnis wie in Abbildung 7.5 aus.

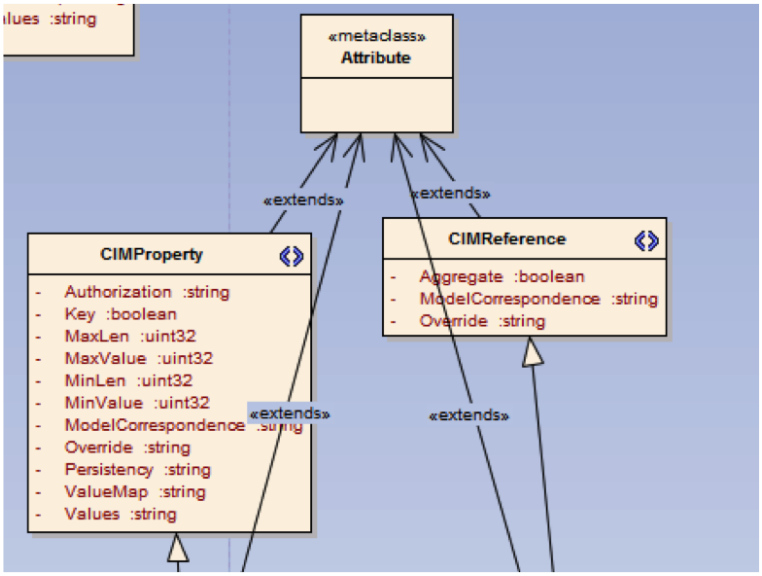


Abbildung 7.5: *Ausschnitt SVS CIM Profile*

Beispiel

In der Abbildung wird lediglich ein kleiner Ausschnitt des nun definierten SVS-CIM-Profils dargestellt, und zwar die Definition der CIM-Properties (CIM-Property) und CIM-Referenzen (CIM-Reference) mit ihren DMTF-spezifischen Eigenschaften.

Zusammenfassung

Auf diese Weise ist das SVS-CIM-Profil definiert. Daraus resultiert das Ergebnis der Phase *Realisierung* den Enterprise Architect mit spezifischen UML-Profile für die Entwicklung des Modells der CIM-Informationen nutzen zu können (siehe auch A.6).

7.2.2 FTS-Profildokument

Aufgabe

Die DMTF stellt eine Dokumentvorlage ([Dis07b]) sowie eine Guideline zur Erstellung eines CIM-Profildokuments ([Dis11]) bereit. Dadurch sind detaillierte Informationen vorhanden, wie ein CIM-Profildokument erstellt werden soll. Jedoch gibt es auch von FTS eine Dokumentenrichtlinie, die ebenfalls Formatierungsvorgaben vorgibt. Die Aufgabe besteht in der Verschmelzung dieser beiden Vorgaben.

Vorlage

Die Lösung ist die Erstellung einer FTS-Profildokumentvorlage unter Berücksichtigung von den FTS-spezifischen Formatrichtlinien und den von der DMTF vorgegebenen Inhalten.

Vereinfachung

Einige Anpassungen zur Vereinfachung der Dokumente sind umgesetzt worden, denn in den verschiedenen Schritten der Methodenweiterentwicklung gab es in einem der ersten CIM-Entwicklungsmethoden nicht die Möglichkeit die Dokumente automatisiert zu erstellen, sondern sie sind manuell erstellt worden. Durch die dort gewonnenen Erkenntnisse ist deutlich geworden, dass nicht alle von der DMTF geforderten Inhalte notwendig für das von FTS anvisierte Ziel sind. Diese nicht notwendigen Inhalte sind jedoch teilweise aufwendig zu erstellen.

Beispiele

Beispiele für weggelassene oder angepasste Inhalte sind zum einen die Requirements-Level. Die DMTF definiert in ihren CIM-Profildokumenten Anforderungen für einzelne Elemente. In den Metadaten dieser Elemente wird definiert, ob diese Elemente bei der Ableitung für produktspezifische CIM-Profile notwendigerweise, konditional oder optional implementiert werden müssen. Diese Information ist bei der Ableitung der produktspezifischen CIM-Profildokumente wichtig, denn bei der Erstellung muss überprüft werden, ob alle notwendigen Elemente vorhanden sind, die Vorbedingungen für die konditionalen Elemente gültig sind und diese somit auch notwendig werden. Das sind die Mindestanforderungen für die Implementierung der CIM-Provider, um Standard-konform zu sein. Die DMTF gibt vor in den produktspezifischen CIM-Profilen diese Informationen ebenfalls anzugeben. Aus Gründen der konstruktiven Qualitätssicherung werden in den FTS-Profildokumenten jedoch nur die zu implementierenden, also notwendigen, Elemente beschrieben. Die konditionalen, deren Vorbedingungen nicht vorherrschen, und optionalen Elemente, die nicht implementiert werden, werden nicht im FTS-Profildokument erwähnt. Diese Inhalte werden nicht erwähnt, da der zusätzliche Spezifikationsaufwand eingespart werden soll und diese Informationen die FTS-Profildokumente verkomplizieren sowie den Entwickler verwirren kön-

nen. Zum anderen sind die Ableitungshierarchien der CIM-Klassen (siehe Abbildung 3.10) laut der DMTF als Tabellen zu beschreiben. Bei der Modellierung des FTS-spezifischen Schemas mit dem Werkzeug Enterprise Architect werden die Ableitungshierarchien für die im Profil enthaltenen Klassen jedoch schon mit einem Diagramm beschrieben. Aus diesem Grund ist es einfacher das Diagramm ins FTS-Profildokument zu kopieren, als die Informationen zusätzlich als Tabelle aufzubereiten, wobei der Informationsgehalt der gleiche ist.

Ergebnis

Das Ergebnis der Realisierung der FTS-Profildokumentvorlage ist ein Hilfsmittel zur Erstellung des FTS-Profildokuments mit Formatierungs- und Inhaltsvorgaben. Bei der automatisierten Dokumentengenerierung mit dem Enterprise Architect, der eine Template-basierte Dokument-Generierungs-Engine enthält, mussten lediglich die Vorgaben in ein Rich-Text-Format überführt werden und mit entsprechenden Platzhaltern für die Modell-Informationen versehen werden. Danach ist es auf Knopfdruck möglich FTS-Profildokumente automatisiert auf Basis der modellbasierten Inhalte zu erstellen. Weitere Details zur automatisierten Dokumentenerstellung sind in [Spi13b] nachzulesen.

7.2.3 Überführung der strukturellen Informationen in MOF-Dateien

Aufgabe

MOF-Dateien werden von dem CIM-Server benötigt (siehe auch 3.3.3). Es müssen dafür die Modell-Information des CIM-Modells für einzelne CIM-Klassen und CIM-Assoziationen in eine für den CIM-Server lesbare Form gebracht werden.

Lösung

Das Werkzeug Enterprise Architect stellt dafür eine Template-basierte Code-Generierungs-Engine bereit, die bei der Realisierung genutzt worden ist. Mit Hilfe der Code-Generierung werden die CIM-Modell-Informationen in die MOF-Datei-Darstellung überführt. Die notwendigen Templates für die MOF-Datei-Generierung sind im Anhang abgebildet (siehe Anhang A.5).

CMPI

Der CIM-Server verwaltet die MOF-Dateien in einem Repository. Im CIM-Server werden die CIM-Provider und die MOF-Dateien registriert. Bei einer Anfrage an den CIM-Server bezüglich CIM-Informationen wird im Repository nachgeschaut, ob eine entsprechende MOF-Datei vorliegt und der für die MOF-Datei registrierte CIM-Provider verfügbar ist. Erst dann wird die Anfrage an den CIM-Provider weitergeleitet. Der CIM-Provider hat in der vorgestellten Architektur (siehe Ab-

bildung 3.9) eine definierte Schnittstelle mit dem Namen Common Manageability Programming Interface (CMPI) [The04]. Diese Schnittstelle wird als Technische Spezifikation von der Arbeitsgruppe *CMPI Working Group*[2] der Organisation *The Open Group* bereitgestellt[3].

Wiederverwendung

CMPI definiert eine C-basierte Schnittstelle zum Austausch von Verwaltungsinformationen innerhalb eines Management-Servers, in diesem Fall zwischen dem CIM-Server und den CIM-Providern. Auf diese Weise können die CIM-Provider in weiteren CIM-Servern wiederverwendet werden.

Werkzeugkette

Mit Hilfe verschiedener Werkzeuge können CIM-Provider-Skeletons aus den MOF-Dateien automatisiert generiert werden, die eine CMPI-konforme Schnittstelle anbieten. Beispielsweise gibt es das Werkzeug CIMPLE [Bra06], mit dem sich C++-konforme, und KonkretCMPI [Bra08] mit dem sich C-konforme CIM-Provider-Skeletons erstellen lassen. Abhängig vom Einsatzgebiet ergibt sich eine Werkzeugkette EA + CIMPLE oder EA + KonkretCMPI. Bei der Realisierung bedarf es lediglich der Auswahl einer der beiden hier genannten Alternativen.

7.2.4 Guideline zur Anpassung der Zulieferer-Provider mit Entwicklungsframework

Ausgangslage

Die Zulieferer-Firma hat ein eigenes Framework zur Erstellung von CIM-Providern erstellt. Sie liefern eine erweiterbare CIM-Implementierung mit der Unterstützung einiger grundlegender CIM-Profile. Die Implementierung umfasst darüber hinaus einige Standard-konforme CIM-Provider. Diese CIM-Provider holen sich die Verwaltungsinformationen ausschließlich über die grundlegenden IPMI-Kommandos (siehe A.2.2). FTS hat in ihren Server-Systemen jedoch eine Menge von OEM-spezifischen IPMI-Kommandos implementiert, die weitergehende Verwaltungsfunktionalität bereitstellen.

Aufgabe

Im Forschungs- und Entwicklungsprojekt war eine der Aufgaben das Framework so zu gestalten, dass die notwendigen Erweiterungen und Anpassungen an der vorhandenen Implementierung durchführbar sind und eine klare Trennung zwischen Zulieferer-Code und Erweiterungs-Code herrscht. Dadurch wird die Zuständigkeit bei der Fehlerbehebung klar definiert.

2 https://collaboration.opengroup.org/tech/management/cmpi/, letzter Aufruf: 2015-01
3 http://www.opengroup.org/aboutus, letzter Aufruf: 2015-01

Diese Trennung ist technisch durch Hook-Methoden realisiert worden. Eine Hook-Methode leitet den Kontrollfluss in ein FTS-spezifisches Programmcode-Paket um und ruft dort ein leere Methode auf. In der leeren Methode wird dann der entsprechende Programmcode zur Anpassung und Erweiterung eingefügt. Lösung

Notwendige Anpassungen und Erweiterungen sind in den folgenden Fällen durchzuführen. Anpassungen

- Neuer CIM-Provider: In dem Fall, dass FTS eine CIM-Klasse definiert, für die es noch keinen CIM-Provider gibt, muss ein neuer CIM-Provider erstellt werden, die entsprechende MOF-Datei ins Repository integriert und der Provider im CIM-Server registriert werden.
- Neues CIM-Profil: In dem Fall, dass FTS ein CIM-Profil definiert, welches noch nicht unterstützt wird, muss ein neues Profil im Programmcode angelegt werden, und für die im CIM-Profil enthaltenen CIM-Klassen neue CIM-Provider erstellt werden.
- Anpassung bestehender CIM-Provider: Diese Aufgabe wird weiter in die folgenden Aufgaben aufgeteilt.
  - Unterstützung weiterer CIM-Properties: In diesem Fall muss ein bestehender CIM-Provider um die neuen Properties angepasst werden. Dafür muss die bestehende Registrierung auf die MOF-Datei umgelenkt werden, die die neuen Properties definiert. Entsprechende Hook-Methoden müssen implementiert werden.
  - Unterstützung weiterer CIM-Methoden: Genau wie für die weiteren CIM-Properties stehen Hook-Methoden für CIM-Methoden zur Verfügung.
  - Anpassung der Werteberechnung für bestehende CIM-Properties: Sind die Werte der CIM-Properties für bestimmte Instanzen der CIM-Klasse anzupassen, ist ebenfalls die Hook-Methode zur Unterstützung neuer CIM-Properties zu nutzen, allerdings muss diese genutzt werden, um bestehende Werte zu überschreiben. Das ist beispielsweise der Fall, wenn Organisations-spezifische Werte wie Firmennamen zurückgegeben werden. FTS möchte ja nicht den Namen des Zulieferers in ihrer CIM-Implementierung nach außen tragen, sondern den eigenen Namen anzeigen.
  - Anpassung der Instanzen: Die Anzahl der Instanzen, die für eine CIM-Klasse konform zu ihrer Definition aus der MOF-Datei erstellt werden, kann unterschiedlich sein. Beispielsweise kennt der Zulieferer nur die

Sensoren im Server-System, die über Standard-IPMI-Kommandos identifiziert werden können. FTS hat aber zusätzliche Sensoren in ihren Server-Systemen, die über OEM-spezifische IPMI-Kommandos identifiziert werden. Beispielsweise bei der Enumerierung der Instanzen der Klasse CIM-NumericSensor muss FTS sicherstellen, dass auch Instanzen für die OEM-spezifischen Sensoren erstellt werden.

Guideline

Für alle oben genannten Anpassungs- und Erweiterungspunkte ist eine detaillierte Guideline für den Programmierer erstellt worden, die die entsprechenden Aufgaben beschreibt, die bei der Auswahl der Hook-Methoden und deren Implementierung mit Beispielcode hilft. Weitere Informationen sind in [Spi13c] nachzulesen.

7.2.5 Testumgebung

Aufgabe

Bei der Realisierung der Testumgebung war die Anforderung *Auf Knopfdruck einen CIM-Provider zu testen* zu erfüllen. Die Vorbedingung war, das CIM-Modell um die Testanforderungen zu vervollständigen.

Lösung

Für einen CIM-Provider sind im CIM-Modell bereits die Informationen hinterlegt, welche Instanzen für die zugehörigen CIM-Klassen zu erstellen sind, welche CIM-Properties unterstützt werden und welche Werte die CIM-Properties bei der Instanziierung zurückliefern dürfen. Auf diese Weise wird für einen CIM-Provider eine Testinstanz erstellt, die die gültigen Werte für eine beliebige Instanz des zu testenden CIM-Providers enthält. Diese Testinstanz definiert dann die Vergleichswerte für einzelne zurückgelieferte Instanzen des CIM-Providers. Dafür sind zusätzliche Informationen ins das CIM-Modell aufgenommen worden. Die Definitionen dieser Informationen sind in der Aktivität *Testfalldefinition* (siehe Abbildung 6.10) beschrieben.

Ergebnis

Auf Basis dieser Vorbedingung ist eine Test Suite in Java prototypisch implementiert worden, deren Konzept in der Abbildung 7.6 dargestellt wird.

JavaCIMTestSuite

Das CIM-Modell ist als EA-Modell vorhanden. Der EA hat die Möglichkeit seine Modellinhalte als XMI zu exportieren. Die Testdaten für ein CIM-Profil werden als XMI exportiert und in einem entsprechenden Ordner abgelegt. Die in Java programmierte Test Suite wird JavaCIMTestSuite genannt. Mit Hilfe der ver-

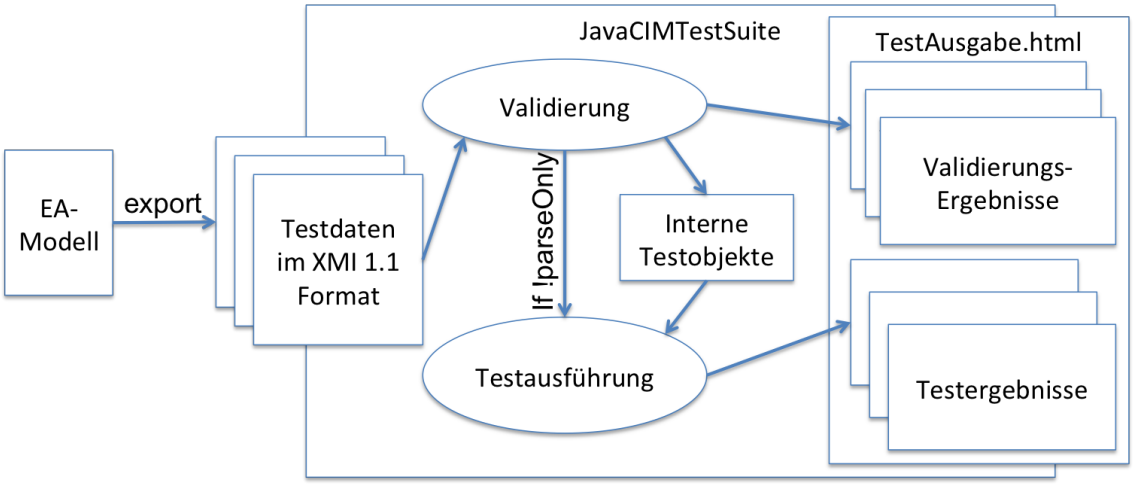


Abbildung 7.6: *Realisierung JavaCIMTestSuite*

fügbaren XML-Parser in Java wird aus den XMI-Daten eine TestInstanz für jede CIM-Klasse im CIM-Profil erstellt (siehe Interne Testobjekte). Die Erstellung der Testinstanzen ist gleichzeitig eine Validierung der Testdaten im CIM-Modell, denn wenn die Testinstanzen nicht erstellt werden, fehlen die entsprechenden Informationen im CIM-Modell. Die Validierungsergebnisse werden in eine Datei mit dem Namen TestAusgabe.html geschrieben. Auf diese Weise erhält der Benutzer Rückmeldung über notwendige Nachbesserungen im CIM-Modell. Die JavaCIM-TestSuite ist im Modus *parse only* ausführbar. Das alleinige Parsen wird so lange wiederholt, bis alle Validierungsergebnisse in Ordnung sind. Danach wird in einem zweiten Schritt die Testfallausführung durchgeführt. Dabei werden für jede Testinstanz die entsprechenden realen Instanzen des Zielsystems abgefragt und jede zurückgelieferte Instanz mit der Testinstanz verglichen. Eine Vielzahl von einzelnen Tests wird durchgeführt. Die Testergebnisse werden ebenfalls in die TestAusgabe.html-Datei geschrieben. Ein Beispiel für die Testfallausführung wird im Anhang A.7 vorgestellt. Weitere Details sind in [Spi14] nachzulesen.

7.2.6 Zusammenfassung der CIM-Entwicklung

Einsparung Aufwand

Die realisierte und weiterentwickelte Methode der CIM-Entwicklung reduziert den Entwicklungsaufwand. Die ursprüngliche Methode war geprägt von aufwendigen Spezifikationsarbeiten. Zuerst ist ein FTS-Profildokument manuell erstellt und mit EA-Modell-Inhalten angereichert worden. Die Erstellung der MOF-Datei war durch die manuelle Erstellung ebenfalls zeitaufwendig und fehlerträchtig.

Das Testen verlief zunächst manuell, danach mit Open-Source-Testwerkzeugen und Testwerkzeugen, die von der Zuliefererfirma des Entwicklungswerkzeugs bereitgestellt worden sind. Das hatte aber den Nachteil, dass die Testfälle wiederum manuell erstellt werden mussten. Letztendlich hat der hier vorgestellte und realisierte Entwicklungsprozess die Aufwände auf ein drittel reduziert. Die tatsächliche Spezifikation wird ausschließlich im EA durchgeführt, die FTS-Profildokumente und die Testfälle werden automatisiert erstellt. Die Testfälle sogar automatisiert durchgeführt. Dadurch ließen sich zwei Drittel des Gesamtaufwands eliminieren. Weiterhin ist der Wissensaustausch effizienter, da alle durchzuführenden Aktivitäten in der CIM-Entwicklung dokumentiert sind und entsprechende Guidelines mit detaillierten Vorgaben vorhanden sind. Das ermöglicht neue Entwickler schnell mit CIM-Entwicklungsaufgaben zu beauftragen, ohne dass eine intensive persönliche Hilfestellung notwendig ist, sondern diese auf spezifische fachliche Fragen reduziert werden kann.

Ergebnis

Im Forschungs- und Entwicklungsprojekt sind mit der weiterentwickelten Methode in den letzten zwei Projektjahren, neben der Weiterentwicklung der Methode selber, über 35 CIM-Profile mit mehr als 300 CIM-Provider entwickelt, getestet und für am Markt erhältliche Produkte freigegeben worden – das mit einer durchschnittlichen Ressourcenverteilung von weniger als 1 Entwickler pro Tag.

Erfolg

Die Nutzung der Entwicklungsmethode der CIM-Entwicklung ist bei FTS etabliert und gilt als ein anschauliches Beispiel für die modellbasierte Entwicklung. Die Akzeptanz dieser Entwicklungsmethode ist auch der situationsgerechten Anpassung und Weiterentwicklung mit MetaMe++ geschuldet.

7.3 Zusammenfassung der Realisierung

Notwendigkeit Realisierung

Die Phase *Realisierung* im MetaMe++-Vorgehen ist die notwendige Konsequenz eine weiterentwickelte Methode im Unternehmen zu etablieren. Die Definition einer Methode reicht als Vorgabe nicht aus, um diese anzuwenden. In der Realisation werden die notwendigen Hilfsmittel in Form von Dokumentvorlagen, Werkzeugen und Guidelines erstellt und den Entwicklern, die nach der Methode arbeiten sollen, bereitgestellt.

heterogene Aufgaben

Die Aufgaben der Realisierung sind unterschiedlich und schlecht planbar. Sie

hängen von den in der Methode definierten Hilfsmitteln ab, sind also erst nach der Phase *Konzeption* bekannt. Meist ist dann erst bekannt, ob lediglich Dokumentvorlagen und Guidelines erstellt werden müssen oder ob auch entsprechende Werkzeuge evaluiert und ausgesucht oder sogar neu entwickelt werden müssen. Wie die beiden Anwendungsbeispiele zeigen, sind beide Realisierungsphasen durch unterschiedliche Aufgaben geprägt.

8 Zusammenfassung und Ausblick

In diesem Kapitel werden die wesentlichen Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf weitere Fragestellungen und Aufgaben gegeben.

8.1 Ergebnisse der Arbeit

MetaMe++

In dieser Arbeit ist die Aufgabe bearbeitet worden eine Methode zur Methodenweiterentwicklung zu entwickeln, die problemspezifische Anforderungen berücksichtigt, eine iterative Weiterentwicklung ermöglicht und dabei unternehmensspezifische Eigenschaften in die Betrachtung mit einbezieht. Dabei ist die Meta-Methode zur Entwicklung von Softwareentwicklungsmethoden MetaMe verallgemeinert und um die Eigenschaften der Methodenweiterentwicklung angereichert worden. Das Ergebnis ist die Meta-Methode zur Methodenweiterentwicklung MetaMe++.

Methoden-weiterentwicklung

Zur Lösung der Aufgabe ist das Vorgehen der Methodenweiterentwicklung beschrieben worden, welches die Phasen *Ist-Analyse*, *Verbesserungs-Analyse*, *Konzeption* und *Realisierung* definiert (siehe Abschnitt 1.3 und 2.3).

Ist-Analyse

Bei der *Ist-Analyse* muss die bestehende Ist-Methode definiert werden. Es hat sich herausgestellt, dass es zwar Sprachen zur Beschreibung von Methoden gibt, aber kein zu den Sprachen passendes Vorgehen zur Ermittlung der Ist-Methode (siehe Problemdefinition der Methodenentwicklung 1). Darüber hinaus stellen bekannte Sprachen die Bestandteile der Methode, Prozessmodell und Produktmodell, nicht explizit getrennt dar, wie es das MetaMe-Konzept fordert. Aus diesem Grund ist die Sprache MMMB (siehe Abschnitt 2.5) definiert worden, die lediglich die rudimentären Methodenelemente darstellt, für die mittels des in dieser Arbeit definierten Vorgehens (siehe Abschnitt 3.6.1) Ermittlungsanweisungen be-

reitgestellt werden. Auf diese Weise ist es möglich die Ist-Methode zu ermitteln und darzustellen.

Situationsfaktoren

Während der Ist-Analyse ist es ebenfalls wichtig die für das Unternehmen spezifischen Eigenschaften zu ermitteln, damit diese in den weiteren Phasen der Methodenweiterentwicklung berücksichtigt werden können (siehe Problemdefinition der Methodenentwicklung 4). Dafür ist zuerst eine Liste von möglichen Situationsfaktoren erstellt worden (siehe Abschnitt 4.3) und beschrieben, wie eine Bewertung der Situationsfaktoren durchgeführt wird (siehe Abschnitt 4.2).

Verbesserungs-Analyse

In der anschließenden Phase *Verbesserungs-Analyse* ist es notwendig Methodenanforderungen zu definieren, die für die anschließende Konzeption der weiterentwickelten Methode eine fundierte und dokumentierte Basis bereitstellen (siehe Problemdefinition der Methodenentwicklung 2). Zuerst ist definiert worden, welche Rolle Methodenanforderungen bei der Methodenweiterentwicklung einnehmen. Danach ist ein aus der Praxis entnommenes Vorgehen zur Ermittlung von Anforderungen auf die Ermittlung von Methodenanforderungen spezialisiert worden. Abschließend ist dargestellt worden, welche Einflüsse Methodenanforderungen auf die (Weiter-)Entwicklung einer Methode haben (siehe Abschnitt 5.1). Eine wichtige Erkenntnis ist die Trennung von Methodenanforderungen, die auf Basis der Ziele des Methodenweiterentwicklungsprojekts erstellt werden, und den situativen Methodenanforderungen, die auf Basis der gültigen Situationsfaktoren ermittelt werden. Die ersteren Methodenanforderungen beschreiben die Optimierungsmöglichkeiten, die zweiten jedoch die situationsgerechte Anpassung der Methode und machen beide Aspekte explizit.

Iterationen

Wie die Problemdefinition der Methodenentwicklung 3 beschreibt, führt das Projektrisiko zur gewünschten schrittweisen Einführung von Verbesserungen. Dafür sind die Standardeigenschaften der Anforderungsdefinition, Priorisierung und Abhängigkeiten, bei den Methodenanforderungen genutzt worden. Mit Hilfe dieser Eigenschaften ist es prinzipiell möglich Teilmengen der Methodenanforderungen zu bilden und daraus inkrementelle Projekte der Methodenweiterentwicklung zu definieren. Die Lösung zur Problemdefinition ist implizit im Vorgehen der Verbesserungs-Analyse integriert worden (siehe Abschnitt 5.2.4).

MMMB

Die Phasen *Ist-Analyse* und *Verbesserungs-Analyse* sind in ein methodisches Vorgehen überführt und die Ergebnisse mit der Sprache MMMB dargestellt worden (siehe Abschnitt 3.6, 4.4 und 5.2).

Konzeption

In der Phase *Konzeption* muss auf Basis der Ist- und Verbesserungs-Analyse die Lösung für die weiterentwickelte Methode konzipiert werden. Dafür ist das Prozess- und Produktmodell von MetaMe mit ein paar wenigen Einschränkungen übernommen und die im MetaMe-Prozessmodell bekannten Aktivitäten detailliert worden. Die angepassten Modelle sind ebenfalls mit MMMB formalisiert worden (siehe Abschnitt 6.1).

Integration der Lösungsbausteine

Schlussendlich sind die einzelnen Lösungsbausteine in das Gesamtvorgehen MetaMe++ integriert worden, welches nun die Eigenschaften der Methodenweiterentwicklung berücksichtigt (siehe Abschnitt 6.2).

weitere Sprachvorgaben

Im Konzept von MetaMe sowie MetaMe++ stellt die Methode selber eine Meta-Methode dar und darüber hinaus auch eine Sprache. Die Sprache wird definiert über das Produktmodell als abstrakte Syntax. Die konkrete Syntax für MetaMe++ muss in Teilen noch definiert werden. Ein Teil des Produktmodells stellt das Meta-Modell der Sprache MMMB dar, für das bereits die konkrete Syntax und Semantik definiert worden ist. Für die weiteren Produktmodell-Elemente ist ebenfalls ein Vorschlag für die Sprachfestlegung getätigt worden. An dieser Stelle gibt es für einige Elemente des Produktmodells keine formalisierte Festlegung. Für die Darstellung der Anforderungen und Anforderungsmodelle sucht sich der Nutzer von MetaMe++ seine bevorzugte Darstellungsform aus (siehe Abschnitt 6.3).

Evaluierung

Der allgemeine Teil dieser Arbeit beschreibt die Methodenweiterentwicklung. Als Lösung ist eine Meta-Methode für die Methodenweiterentwicklung entwickelt worden. Diese Meta-Methode ist in verschiedenen Methodenweiterentwicklungsprojekten nutzbar. In dieser Arbeit hat sich die Anwendung auf die Server-System-Entwicklung beschränkt. Zwei Anwendungsfälle aus einem realen Forschungs- und Entwicklungsprojekt sind betrachtet worden. In beiden Fällen sind neue Erkenntnisse aus den Anwendungsfällen in die Entwicklung von MetaMe++ eingeflossen und das weiterentwickelte MetaMe++ ist wiederum auf die Anwendungsbeispiele angewendet worden. Somit zeigen die Anwendungsbeispiele die Anwendbarkeit von MetaMe++. Mit Hilfe von MetaMe++ sind zwei konkrete Methoden, die System-Architektur-Spezifikation und die CIM-Entwicklung, weiterentwickelt worden.

konkrete Verbesserungen

In den konkreten Anwendungsfällen der Server-System-Entwicklung konnten einige Schwächen erkannt werden. Ein Teil der Schwächen wurde in den Grundla-

gen dieser Arbeit bereits behoben. Ein anderer Teil der Schwächen wurde jedoch erst im Zuge der Methodenweiterentwicklung behoben.

Definition Struktur von Server-Systemen

Für die Lösung der Herausforderung der fehlenden Festlegung des Domänenwissens und der Systemarchitekturen im Bereich der Server-System-Entwicklung sind im Kapitel 3 die notwendigen Definitionen getätigt worden. Darüber hinaus werden im Abschnitt 3.2 die unterschiedlichen Arten von Server-Systemen beschrieben.

verbesserte Methoden

Die weiteren Herausforderungen der Server-System-Entwicklung konnten exemplarisch in den zwei Anwendungsfällen und die darauf angewendete Methodenweiterentwicklung gelöst werden. In beiden Fällen ist eine verbesserte Methode entwickelt worden.

Die folgende Tabelle 8.1 stellt die erarbeiteten Lösungen den Teilaufgaben dieser Dissertation gegenüber.

Tabelle 8.1: *Zusammenfassung der Teilaufgaben*

Teilaufgabe der Dissertation	**Lösung**
1 Detaillierung des Vorgehens der Methodenweiterentwicklung, so dass ein Methodenentwickler das Vorgehen versteht und durchführen kann	MetaMe++ (siehe Kapitel 6)
2 *Definition einer MetaMe kompatiblen Methodenbeschreibungssprache*	MMMB (siehe Abschnitt 2.5)
3 *Bereitstellung der Hilfsmittel zur Ermittlung des Methodenkontexts*	Checkliste möglicher Situationsfaktoren (siehe Abschnitt 4.3)
4 *Dokumentation des Methodenkontexts*	Sprachfestlegung im MetaMe++-Produktmodell (siehe Abschnitt *Ist-Analyse::Methodenkontext* im Abschnitt 6.3)

5 *Durchführung einer Methodenweiterentwicklung für die Server-System-Entwicklung*	Anwendungsbeispiel System-Architektur-Spezifikation • Ist-Analyse::Ermittlung Ist-Methode (siehe Abschnitt 3.6.2) • Ist-Analyse::Methodenkontext (siehe Abschnitt 4.4.3) • Verbesserungs-Aanalyse (siehe Abschnitt 5.2.2) • Konzeption (siehe Abschnitt 6.1.2) • Realisierung (siehe Abschnitt 7.1) Anwendungsbeispiel CIM-Entwicklung • Ist-Analyse::Ermittlung Ist-Methode (siehe in Abschnitt 3.6.3) • Ist-Analyse::Methodenkontext (siehe Abschnitt 4.4.3) • Verbesserungs-Analyse (siehe Abschnitt 5.2.3 und 5.2.4) • Konzeption (siehe Abschnitt 6.1.3) • Realisierung (siehe Abschnitt 7.2)

Mit der Anwendung von MetaMe++ auf die beiden Anwendungsfälle konnte die Hypothese dieser Arbeit *Mit MetaMe++ ist eine Methodenweiterentwicklung durchführbar* bestätigt werden.

8.2 Ausblick

In dieser Arbeit sind grundlegende Erkenntnisse zur Methodenweiterentwicklung beschrieben worden. Der Ansatz dieser Arbeit ist praxisbezogen und anwendbar, was beides gezeigt worden ist.

MMMB vs. Methodenbeschreibungssprachen

Eine erste Annahme dieser Arbeit ist, dass es immer eine Ist-Methode gibt. In den Anwendungsbeispielen war die Ist-Methode implizit vorhanden und ist mit Hilfe von MMMB und dem MetaMe++-Vorgehen der Ist-Analyse explizit dokumentiert worden. In der Industrie ist die Festlegung und Beschreibung von Methoden und Prozessen jedoch schon eine seit längerem durchgeführte Tätigkeit. Die erneute Beschreibung mit MMMB, wenn doch schon eine umfassende Be-

schreibung besteht, scheint einen unnötigen Mehraufwand einzufordern. Es ist zu untersuchen, inwieweit bestehende Methodenbeschreibungen in MMMB überführt werden können, wenn ihnen notwendiger Weise eine feste Syntax und Semantik unterliegt. SPEM wäre ein erster Kandidat für Kompatibilitätsbetrachtungen, da erfahrungsgemäß zu erwarten ist, dass SPEM einen der weitverbreitesten Vertreter einer formalisierten Sprache zur Beschreibung von Methoden darstellt. Gefühlsmäßig erscheinen mir (Geschäfts-)Prozessmodelle sowie Workflowmodelle ebenfalls eine geeignete Grundlage zur Weiterentwicklung einer Methode zu sein. Es bleibt zu untersuchen, inwiefern solche Modelle wirklich im Rahmen der Methodenweiterentwicklung zu handhaben sind. Wobei ich davon ausgehe, dass das MetaMe++-Prozessmodell ohne Weiteres direkt durch einen Vertreter der allgemeinen Prozessmodelle ersetzt werden kann. Hauptsächlich ist es notwendig ein Sprachkonstrukt der Prozessbeschreibungssprachen auf Artefakte abbilden zu können. Bei UML-Aktivitätendiagrammen dürften das die UML-Objekte und bei der BPMN die Data Objects sein. Damit ist es möglich eine Referenz zwischen Prozessmodellen und den für MetaMe++ notwendigen Produktmodell zur Definition der Artefakttypen zu gewährleisten. Der Vorteil von MMMB ist jedoch durch die Tatsache gewährleistet, speziell für MetaMe++ entwickelt zu sein. Weiterhin ist MMMB einfacher handhabbar, da weitaus weniger Sprachkonstrukte enthalten sind.

weitere Situationsfaktoren

Die Erstellung von situationsgerechten Entwicklungsmethoden war eine der Hauptaufgaben dieser Arbeit. Durch die Anwendung von MetaMe++ werden neben Methodenanforderungen auch speziell situative Methodenanforderungen ermittelt, deren Implementierung zur situationsgerechten Anpassung der Soll-Methode führen. Die situativen Methodenanforderungen werden auf Basis einer Liste von möglichen Situationsfaktoren ermittelt. Diese Liste stellt selbstverständlich nur eine Momentaufnahme der bis dato gesammelten Erkenntnisse dar. Zum einen muss die Liste weiterentwickelt werden. Zum anderen müssen weitere Forschungserkenntnisse bezüglich der Auswirkungen bestimmter Situationsfaktoren ermittelt werden. Es gibt lediglich einige Guidelines, wie auf Basis einer bestimmten Situation die zu entwickelnde Methode gestaltet werden muss. Weiterhin schiebt der in dieser Arbeit vorgestellte Ansatz die Verantwortung auf den Methodenentwickler, der für die Implementierung der situativen Methodenanforderungen zuständig ist. Zur Verbesserung dieses Sachverhalts kann das Konzept der Methodenanforderungen weiter verfolgt werden, indem weitere und bessere Guidelines entwickelt werden, die bei der konkreten Detaillierung und Um-

setzung der situativen Methodenanforderungen fundierte und konkrete Vorgaben zum Methodendesign machen.

Messung der Verbesserung

Eine der wichtigsten Fragen bei der Methodenweiterentwicklung ist die nach der erreichten Verbesserung der Methode. Die erreichten Verbesserungen im Rahmen dieser Arbeit durchgeführten Methodenweiterentwicklungen sind argumentativ dargestellt worden. In den Forschungs- und Entwicklungsprojekten war das Verständnis vorhanden, dass es die zwei Optimierungspotenziale Sprachentwicklung und Methodenentwicklung gibt. Aufgrund der Anwendung von MetaMe++ konnten jeweils weitere Festlegungen und Detaillierungen bezüglich der Methode sowie für die (vorbereitende Tätigkeiten der) Sprachdefinition dokumentiert werden. Durch die Etablierung von Konzepten der Modell- und komponentenbasierten Entwicklung, Grundkonzepte von MetaMe++, konnten teilweise Entwicklungswerkzeuge erstellt oder angepasst werden, die Entwicklungstätigkeiten automatisieren oder einen Beitrag zur konstruktiven Qualitätssicherung machen. In den beiden Aspekten sind die Optimierungspotenziale Entwicklungszeit und Qualität verborgen. Mit Hilfe dieser beiden Optimierungspotenziale konnte eine Verbesserung der Methoden belegt werden, indem gezeigt worden ist, dass mit der weiterentwickelten Methode die Entwicklungszeit verkürzt oder die Qualität gesteigert werden konnte.

weitere Evaluierung

Bezüglich der Nachweisbarkeit von Optimierungen mangelt es zum einen an der Verfügbarkeit von geeigneten Metriken und zum anderen an der Durchführung von geeigneten Versuchen. Ein Projekt verläuft nie genau gleich wie ein vorhergehendes. Darum ist ein direkter Vergleich mit der Vorher- und Nachher-Methode einer Methodenweiterentwicklung schlecht machbar. Die parallele Durchführung der selben Entwicklungsaufgabe mit zwei unterschiedlichen Entwicklungsmethoden zur Untersuchung und Ermittlung der besseren Entwicklungsmethode ist in der Realität mit hohem Ressourcenaufwand verbunden. In den dieser Arbeit zugrunde liegenden Anwendungsbeispielen konnte dieser Aufwand nicht geleistet werden.

Metriken

Bezüglich der Messung der Verbesserung ist es sinnvoll einen Blick auf die herausgestellten Methodenanforderungen zu werfen. Diese beschreiben letztendlich die gewünschte Ideal-Methode. Mit Hilfe von Überdeckungsmetriken von Anforderungen ist es möglich eine Aussage zu treffen, inwieweit die Ideal-Methode nach einer Methodenweiterentwicklung erreicht werden konnte. Allerdings haben wir in dieser Arbeit auch gesehen, dass die Menge der Methodenanforderun-

gen in den einzelnen Iterationen der Methodenweiterentwicklung unterschiedlich sein kann. Das liegt zum einen an den Kenntnissen, die bei einer Iteration der Methodenweiterentwicklung gesammelt werden, zum anderen können sich auch die übergeordneten Ziele im Laufe der Zeit ändern.

psychologische Faktoren

Eine Methodenweiterentwicklung und die Etablierung der weiterentwickelten Methode hängt neben technischen und konzeptionellen Herausforderungen auch stark von psychologischen Aspekten ab. Einige menschliche Situationsfaktoren sind in der Arbeit bereits thematisiert worden. Dennoch ist es besonders schwierig eine Methodenweiterentwicklung durchzuführen, wenn die Mitarbeiter im Projekt generell Ängste haben, wenn eine Analyse ihrer Arbeitsweise und Arbeitsergebnisse erfolgen soll. Aber auch der Widerstand gegen Neuerungen gefährdet den Erfolg einer Methodenweiterentwicklung. Die Untersuchung dieser psychologischen Faktoren mit dem Ziel, deren negativen Einfluss zu reduzieren, konnte in dieser Arbeit nicht thematisiert werden, muss in Zukunft aber sicherlich in die Betrachtung miteinbezogen werden.

Werkzeugunterstützung

Das Thema Werkzeugunterstützung ist ebenfalls wichtig für den Erfolg dieser Methode. MMMB ist als UML-Profil definiert worden und MetaMe++ formalisiert. Der weitere konsequente Schritt ist eine umfassende Werkzeugunterstützung bereitzustellen. Momentan muss der MetaMe++-Prozessanteil manuell durchgeführt werden. Es ist aber auch denkbar ihn durch eine Workflow-Engine maschinell auszuführen. Die Definition und Dokumentation der durch MetaMe++ erstellten Ist- und Soll-Methode wird mittels des MMMB-UML-Profils und des Werkzeugs EA durchgeführt. Für die Phase der *Verbesserungs-Analyse* mit den Aufgaben der Erstellung von Anforderungs-, Stakeholder- und Kontext-Modellen sind lediglich Vorschläge für die Darstellung und Dokumentation gemacht worden. Hier sollen jedoch keine Festlegungen gemacht werden, da Darstellungen und Sprachen genutzt werden sollen, die im Unternehmen und beim Methodenentwickler bekannt sind und tatsächlich auch genutzt werden. Ein generischer Werkzeugansatz mit Integrationsmöglichkeit unterschiedlicher weiterer Sprachen muss in Betracht gezogen werden.

Neben der Meta-Ebene, auf der die Methodenweiterentwicklung durchgeführt wird, gibt es die konkrete Ebene der (weiterentwickelten) Methoden.

Formale Sprachen zur System-Architektur-Spezifikation

Bei der Server-System-Entwicklung ist der Fokus auf die grundlegenden Spezifikationsarbeiten und deren Dokumentation gelegt worden. Insbesondere die

Dokumenten-Organisation und -Struktur sind definiert worden. Im Sinne der modell- und komponentenbasierten Entwicklung sind entsprechende strukturelle Eigenschaften erarbeitet worden, die hauptsächlich auf SysML-Sprachkonstrukte abgebildet worden sind, damit eine formalisierte Sprache bei der Spezifikation genutzt werden kann. Die Ergebnisse zeigen jedoch weiteres Potenzial auf, die SysML speziell für die Server-System-Spezifikation bei FTS zu erweitern. Bei der Spezifikation des Gesamtsystems müssen physikalische Eigenschaften wie Maße, elektrotechnische Spezifikationen und thermodynamische Vorgaben direkt in den zu erstellenden Modellen integriert werden. Erst dann ist es möglich ein Produktdesign vollständig modellbasiert zu definieren.

MechatronicUML

Ein Server-System kann als mechatronisches System angesehen werden, denn eine Reihe der Eigenschaften mechatronischer Systeme finden sich in Server-Systemen wieder. Beispielsweise enthalten Server-Systemen (siehe Abschnitt 3.4) verschiedene Sensoren, Aktoren und Steuerungskomponenten. Kontinuierliche Regelung von physikalischen Attributen wie Lüftung und Stromversorgung, sich verändernde physikalische Ausstattungen eines Server-Systems sowie der Zusammenschluss mehrerer autonom agierender Server-Systeme in einer größeren Verwaltungsinstanz sind weitere Eigenschaften. Die Entwicklung von Server-Systemen ist darüber hinaus eine interdisziplinäre Aufgabe der Mechanik, Elektrotechnik und Softwareentwicklung (vergleiche mit Kapitel 1 in [BDG+14b]). Es liegt nahe die Entwicklungsmethodik MechatronicUML [BDG+14a] in der Entwicklung der Server-Systeme in Betracht zu ziehen. Sie definiert einen Entwicklungsprozess, Sprachen und Werkzeuge speziell für mechatronische Systeme. Allerdings liegt der Fokus im Besonderen auf Real-Zeit und sicherheitskritische Eigenschaften, die bei Server-Systemen nicht im Vordergrund stehen. Als Grundtenor dieser Arbeit ist es schwierig die bestehende Entwicklungsmethode durch die der MechatronicUML zu ersetzen. Der MechatronicUML-Entwicklungsprozess kann nicht eins zu eins auf den bestehenden Entwicklungsprozess abgebildet werden. Es fehlt das Know-how im Einsatz von stark formal definierten Sprachen, wie denen, die von der MechatronicUML bereitgestellt werden (Component Model, Real-Time Statecharts, Real-Time Coordination Protocols). Dennoch sollte für weitere Methodenweiterentwicklungsprojekte im Rahmen der Server-System-Entwicklung die Nutzung von MechatronicUML als weiteres Ziel definiert werden. Für einige Systemkomponenten ist der Einsatz der detaillierten Konzepte der komponentenbasierten Entwicklung sinnvoll, denn es wird ein Vorteil sein die von der MechatronicUML gelieferten Werkzeuge zur Spezifikation und Verifikation von Systemkomponenten nutzen zu können. Allerdings muss dafür in vorher-

gehenden Methodenweiterentwicklungsprojekten erst eine solide Grundlage für die Einführung von MechatronicUML geschaffen werden.

Produktlinien

Ein weiterer erkannter Anwendungsfall ist die Wiederverwendung von Systemspezifikationen. Wie beispielsweise im Sinne der in dieser Arbeit vorgestellten Referenzarchitektur eines Verwaltungsmoduls muss eine Trennung von allgemeiner wiederverwendbarer System-Architektur und konkreter Produkt-Architektur etabliert werden. Die Systembausteine eines Server-Systems weisen in den verschiedenen Produkten große Gemeinsamkeiten auf, das liegt zum einen an der Produktvielfalt. Verschiedene Systeme werden für verschiedene Marktgruppen entwickelt, haben jedoch alle einen Kern an Grundfunktionalitäten. Zum anderen müssen neue Systeme einen Großteil von am Markt etablierten alten Verwaltungsschnittstellen aus Gründen der Kompatibilität bereitstellen. Durch diese beiden Eigenschaften ist die Wiederverwendung eine Grundlage für die effiziente und effektive Systementwicklung. Im Sinne der modell- und komponentenbasierten Entwicklung ist es notwendig sicherzustellen, dass es allgemeine wiederwendbare Spezifikationen für System-Module und -Komponenten gibt, die für ein spezielles Produkt konkretisiert werden können. Aus Sicht der Forschungsgebiete ist eine Überarbeitung der System-Architektur-Spezifikation im Sinne des Produktlinien-Konzepts sinnvoll.

Ideal-Methode in der CIM-Entwicklung

Im Anwendungsbeispiel der CIM-Entwicklung ist die Methodenweiterentwicklung bereits so weit fortgeschritten, dass man sich bereits sehr nah an der Ideal-Methode bewegt und kaum noch Optimierungspotenzial besteht.

Literaturverzeichnis

[ABG09] Alter, Stefanie ; Börner, René ; Goeken, Matthias: Operationalisierung der IT-Governance-Kernbereiche für die Identifizierung und Gestaltung von Services. In: *Informatik 2009: Im Focus das Leben, Beiträge der 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 28.9.-2.10.2009, Lübeck, Proceedings*, 2009, 3660–3674

[AK03] Atkinson, Colin ; Kühne, Thomas: Model-Driven Development: A Metamodeling Foundation. In: *IEEE Softw.* 20 (2003), September, Nr. 5, S. 36–41. – ISSN 0740–7459

[ARB07] Aharoni, Anat ; Reinhartz-Berger, Iris: Representation of Method Fragments. In: Ralyté, Jolita (Hrsg.) ; Brinkkemper, Sjaak (Hrsg.) ; Henderson-Sellers, Brian (Hrsg.): *Situational Method Engineering: Fundamentals and Experiences* Bd. 244. Springer US, 2007. – ISBN 978–0–387–73946–5, S. 130–145

[ASB04] Amelunxen, Carsten ; Schürr, Andy ; Bichler, Lutz: Codegenerierung für Assoziationen in MOF 2.0. In: Rumpe, Bernhard (Hrsg.) ; Hesse, Wolfgang (Hrsg.): *Modellierung 2004, Proceedings zur Tagung, 23.-26. März 2004, Marburg, Proceedings* Bd. 45, GI, 2004 (LNI), S. 149–168

[Bad11] Badenfelt, Ulrika: Fixing the contract after the contract is fixed: A study of incomplete contracts in IT and construction projects. In: *International Journal of Project Management* 29 (2011), Nr. 5, S. 568–576. – ISSN 0263–7863

[Bal09] Balzert, Helmut: *Lehrbuch Der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Springer, 2009. – ISBN 9783827417053

[BDG+14a] Becker, Steffen ; Dziwok, Stefan ; Gerking, Christopher ; Heinzemann, Christian ; Schäfer, Wilhelm ; Meyer, Matthias ; Pohlmann, Uwe: The MechatronicUML Method: Model-driven Software Engineering of Self-adaptive Mechatronic Systems. In: *Companion Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA : ACM, 2014 (ICSE Companion 2014). – ISBN 978–1–4503–2768–8, S. 614–615

[BDG+14b] Becker, Steffen ; Dziwok, Stefan ; Gerking, Christopher ; Schäfer, Wilhelm ; Heinzemann, Christian ; Thiele, Sebastian ; Meyer, Matthias ; Priesterjahn, Claudia ; Pohlmann, Uwe ; Tichy, Matthias: The MechatronicUML Design Method - Process and Language for Platform-Independent Modeling / Heinz Nixdorf Institute, University of Paderborn. 2014 (tr-ri-14-337). – Forschungsbericht. – Version 0.4

[Bec99] Beck, Kent: Extreme Programming: A Discipline of Software Development. In: *ESEC / SIGSOFT FSE*, 1999, S. 1

[BKKW07] Bucher, Tobias ; Klesse, Mario ; Kurpjuweit, Stephan ; Winter, Robert: Situational Method Engineering: On the Difference of "Context" and "Projekt Type". In: Ralyté, Jolita (Hrsg.) ; Brinkkemper, Sjaak (Hrsg.) ; Henderson-Sellers, Brian (Hrsg.): *Situational Method Engineering: Fundamentals and Experiences* Bd. 244. Springer Boston, 2007. – ISBN 978–0–387–73946–5, S. 33–48

[Boe79] Boehm, Barry W.: Guidelines for Verifying and Validating Software Requirements and Design Specifications. In: Samet, P. A. (Hrsg.): *Euro IFIP 79*, North Holland, 1979, S. 711–719

[Bra06] Brasher, Michael E. and Schopmeyer, Karl: *CIMPLE: An Embeddable CIM Provider Engine*. `http://simplewbem.org/whitepaper.pdf`. Onlinedokument, letzter Aufruf: 2015-01, 2006)

[Bra08] Brasher, Michael E.: *KonkretCMPI*. `http://konkretcmpi.org/KonkretCMPI.pdf`. Onlinedokument, letzter Aufruf: 2015-01, 2008)

[Bri96] Brinkkemper, Sjaak: Method Engineering: Engineering of Information Systems Development Methods and Tools. In: *Information and*

Software Technology 38 (1996), Nr. 4, S. 275–280

[BS98] BEA Systems, Inc: *BEA Manager: Agent Integrator Reference Manual*. `http://docs.oracle.com/cd/E13192_01/manager/mgr20/`. Onlinedokument, letzter Aufruf: 2015-01, 1998

[BWBM08] Bekkers, Willem ; Weerd, Inge van d. ; Brinkkemper, Sjaak ; Mahieu, Alain: The Influence of Situational Factors in Software Product Management: An Empirical Study. In: *Proceedings of the 2008 Second International Workshop on Software Product Management*. Washington and DC and USA : IEEE Computer Society, 2008 (IWSPM '08). – ISBN 978–0–7695–3625–5, S. 41–48

[CATP10] Cervera, Mario ; Albert, Manoli ; Torres, Victoria ; Pelechano, Vicente: A Methodological Framework and Software Infrastructure for the Construction of Software Production Methods. In: Münch, Jürgen (Hrsg.) ; Yang, Ye (Hrsg.) ; Schäfer, Wilhelm (Hrsg.): *New Modeling Concepts for Today's Software Processes* Bd. 6195. Springer Berlin / Heidelberg, 2010. – ISBN 978–3–642–14346–5, S. 112–125

[CATP11] Cervera, Mario ; Albert, Manoli ; Torres, Victoria ; Pelechano, Vicente: Turning Method Engineering Support into Reality. In: Ralyté, Jolita (Hrsg.) ; Mirbel, Isabelle (Hrsg.) ; Deneckère, Rébecca (Hrsg.): *Engineering Methods in the Service-Oriented Context* Bd. 351. Springer Boston, 2011. – ISBN 978–3–642–19996–7, S. 138–152

[CATP12] Cervera, Mario ; Albert, Manoli ; Torres, Victoria ; Pelechano, Vicente: The MOSKitt4ME Approach: Providing Process Support in a Method Engineering Context. In: Atzeni, Paolo (Hrsg.) ; Cheung, David (Hrsg.) ; Ram, Sudha (Hrsg.): *Conceptual Modeling* Bd. 7532. Springer Berlin / Heidelberg, 2012. – ISBN 978–3–642–34001–7, S. 228–241

[Chr12] Christ, Fabian: *Automatische Kompatibilitätsprüfung Framework-basierter Anwendungen*, Universität Paderborn, Diss., 2012

[Coc00] Cockburn, Alistair: Selecting a Project's Methodology. In: *IEEE Software* 17 (2000), Nr. 4, S. 64–71. – ISSN 0740–7459

[CPL09] Chung, Lawrence ; Prado Leite, Julio do: On Non-Functional Requirements in Software Engineering. In: Borgida, Alexander (Hrsg.) ; Chaudhri, Vinay (Hrsg.) ; Giorgini, Paolo (Hrsg.) ; Yu, Eric (Hrsg.): *Conceptual Modeling: Foundations and Applications* Bd. 5600. Springer Berlin / Heidelberg, 2009, S. 363–379

[CS05] Cossentino, Massimo ; Seidita, Valeria: Composition of a New Process to Meet Agile Needs Using Method Engineering. In: Choren, Ricardo (Hrsg.) ; Garcia, Alessandro (Hrsg.) ; Lucena, Carlos (Hrsg.) ; Romanovsky, Alexander (Hrsg.): *Software Engineering for Multi-Agent Systems III* Bd. 3390. Springer Berlin / Heidelberg, 2005. – ISBN 978–3–540–24843–9, S. 36–51

[DH07] Drews, Günter ; Hillebrand, Norbert: *Lexikon der Projektmanagement-Methoden*. Haufe-Mediengruppe, 2007 (Haufe Projektmanagement). – ISBN 9783448080520

[DH08] Dinger, Jochen ; Hartenstein, Hannes: *Netzwerk- und IT-Sicherheitsmanagement: Eine Einführung*. Universitätsverlag Karlsruhe, 2008. – ISBN 978–3–8664–209–2

[Dis07a] Distributed Management Task Force, Inc.: *CIM Query Language Specification, DSP0202, v1.0.0*. `http://www.dmtf.org/sites/default/files/standards/documents/DSP0202_1.0.0.pdf`. Onlinedokument, letzter Aufruf: 2015-01, 2007)

[Dis07b] Distributed Management Task Force, Inc.: *Management Profile Specification Template, DSP1000, v1.0.0*. `http://www.dmtf.org/sites/default/files/standards/documents/DSP1000.pdf`. Onlinedokument, letzter Aufruf: 2015-01, 2007)

[Dis07c] Distributed Management Task Force, Inc.: *Server Management Command Line Protocol (SM CLP), DSP0214, v1.0.2*. `http://www.dmtf.org/sites/default/files/standards/documents/DSP0214.pdf`. Onlinedokument, letzter Aufruf: 2015-01, 2007)

[Dis09a] Distributed Management Task Force, Inc.: *CIM Operations over HTML, DSP0200, v1.3.1*. `http://www.dmtf.org/sites/default/files/standards/documents/DSP0200_1.3.1.pdf`. Onlinedokument,

letzter Aufruf: 2015-01, 2009)

[Dis09b] Distributed Management Task Force, Inc.: *Representation of CIM using XML, DSP0201, v2.3.1*. http://www.dmtf.org/sites/default/files/standards/documents/DSP0201_2.3.1.pdf. Onlinedokument, letzter Aufruf: 2015-01, 2009)

[Dis09c] Distributed Management Task Force, Inc.: *UML Profile for CIM, v1.0.0*. http://www.dmtf.org/sites/default/files/standards/documents/DSP0219_1.0.0.pdf. Onlinedokument, letzter Aufruf: 2015-01, 2009)

[Dis10a] Distributed Management Task Force, Inc.: *CIM Infrastructure Specification, DSP0004, v2.6.0*. http://www.dmtf.org/sites/default/files/standards/documents/DSP0004_2.6.0_0.pdf. Onlinedokument, letzter Aufruf: 2015-01, 2010

[Dis10b] Distributed Management Task Force, Inc.: *Web Services for Management, DSP0226, v1.1.0*. http://www.dmtf.org/sites/default/files/standards/documents/DSP0226_1.1.pdf. Onlinedokument, letzter Aufruf: 2015-01, 2010)

[Dis11] Distributed Management Task Force, Inc.: *Profile Usage Guide, DSP1001, v1.1.0*. http://www.dmtf.org/sites/default/files/standards/documents/DSP1001_1.1.0_0.pdf. Onlinedokument, letzter Aufruf: 2015-01, 2011)

[Dis13] Distributed Management Task Force, Inc.: *Model Object Format (MOF), v3.0.0*. http://www.dmtf.org/sites/default/files/standards/documents/DSP0221_3.0.0.pdf. Onlinedokument, letzter Aufruf: 2015-01, 2013)

[DW06] Distributed Management Task Force, Inc. ; WBEM Solutions, Inc.: *CIM Tutorial*. http://www.wbemsolutions.com/tutorials/DMTF/. Webseite letzter Aufruf: 2015-01, 2006

[ESS08] Engels, Gregor ; Sauer, Stefan ; Soltenborn, Christian: Unternehmensweit verstehen – unternehmensweit entwickeln: Von der Modellierungssprache zur Softwareentwicklungsmethode. In: *Informatik-*

Spektrum 31 (2008), Nr. 5, S. 451–459. – ISSN 0170–6012

[FBGL+13] Fazal-Baqaie, Masud ; Güldali, Baris ; Luckey, Markus ; Sauer, Stefan ; Spijkerman, Michael: Maßgeschneidert und werkzeugunterstützt Entwickeln angepasster Requirements Engineering-Methoden. In: *OBJEKTspektrum (Online Themenspecials)* (2013), June, Nr. RE/2013, S. 1–5

[FHKS09] Friedrich, Jan ; Hammerschall, Ulrike ; Kuhrmann, Marco ; Sihling, Marc: *Das V-Modell® XT: Für Projektleiter und QS-Verantwortliche kompakt und übersichtlich*. Springer Berlin Heidelberg, 2009. – ISBN 978–3–642–01488–8

[FV04] Fuentes, Lidia ; Vallecillo, Antonio: An Introduction to UML Profiles. In: *UPGRADE, The European Journal for the Informatics Professional* 5 (2004), Nr. 2, S. 5–13

[Gan10] Ganthier, Jim : *Top 10 Server Trends for 2010: Understanding these ten server trends can help CIOs succeed despite an unsteady economy*. `http://esj.com/articles/2010/01/12/server-trends-2010.aspx`. Version: 2010

[HLL12] Hansmann, Holger ; Laske, Michael ; Luxem, Redmer: Einführung der Prozesse - Prozess-Roll-out. In: Jörg Becker, Martin Kugeler Michael R. (Hrsg.): *Prozessmanagement - Ein Leitfaden zur prozessorientierten Organisationsgestaltung*, Springer Berlin Heidelberg, 2012, S. 277–307

[Hob04] Hobbs, Chris: *A Practical Approach to WBEM/CIM Management /// A practical approach to WBEM/CIM management*. Boca Raton and FL and USA /// Boca Raton and Fla. : CRC Press, Inc. and Auerbach, 2004. – ISBN 0849323061

[HS05] Henderson-Sellers, Brian: Creating a comprehensive agent-oriented methodology - using method engineering and the OPEN metamodel. In: *the OPEN metamodel, Chapter 13 in Agent-Oriented Methodologies (eds. B. Henderson-Sellers and P. Giorgini), Idea Group*, 2005, S. 11

[HS06] Henderson-Sellers, Brian: Method Engineering: Theory and Practice. In: Karagiannis, Dimitris (Hrsg.) ; Mayr, Heinrich C. (Hrsg.): *Information Systems Technology and its Applications, 5th International Conference ISTA'2006, May 30-31, 2006, Klagenfurt, Austria* Bd. 84, GI, 2006 (LNI). – ISBN 3–88579–178–1, S. 13–23

[HSR10] Henderson-Sellers, Brian ; Ralyté, Jolita: Situational Method Engineering: State-of-the-Art Review. In: *j-jucs* 16 (2010), Nr. 3, S. 424–478

[Int99] Intel Corporation, Hewlet-Packard Company, NEC Corporation, Dell inc.: *Platform Managment FRU Information Storage Definition, v2.0, Document Revision 1.1*. `http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/information-storage-definition.pdf`. Onlinedokument, letzter Aufruf: 2015-01, 1999)

[Int06] Intel and Hewlett-Packard and Dell and Avocent: *IPMI - IPMI CIM Mapping Guideline, v0.6*. `http://www.intel.de/content/dam/www/public/us/en/documents/product-briefs/cim-mapping-guideline-.6.pdf`. Onlinedokument, letzter Aufruf: 2015-01, 2006)

[Int09] Intel Corporation, Hewlet-Packard Company, NEC Corporation, Dell inc.: *Intelligent Platform Management Interface Specification Second Edition, v2.0*. `http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/second-gen-interface-spec-v2.pdf`. Onlinedokument, letzter Aufruf: 2015-01, 2009)

[JBR99] Jacobson, Ivar ; Booch, Grady ; Rumbaugh, James: *The unified software development process*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1999. – ISBN 0–201–57169–2

[KA04] Karlsson, Fredrik ; Agerfalk, Pär J.: Method configuration: adapting to situational characteristics while creating reusable assets. In: *Information and Software Technology* 46 (2004), Nr. 9, 619–633. `http://www.sciencedirect.com/science/article/pii/S0950584903002507`

[KDS07] Kornyshova, Elena ; Deneckère, Rébecca ; Salinesi, Camille: Method Chunks Selection by Multicriteria Techniques: an Extension of the Assembly-based Approach. In: Ralyté, Jolita (Hrsg.) ; Brinkkemper, Sjaak (Hrsg.) ; Henderson-Sellers, Brian (Hrsg.): *Situational Method Engineering: Fundamentals and Experiences* Bd. 244. Springer Boston, 2007. – ISBN 978–0–387–73946–5, S. 64–78

[Luc13] Luckey, Markus: *Adaptivity engineering : Modeling and quality assurance for self-adaptive software systems*, Universität Paderborn, Diss., 2013

[MR06] Mirbel, Isabelle ; Ralyté, Jolita: Situational method engineering: combining assembly-based and roadmap-driven approaches. In: *Requir. Eng.* 11 (2006), Nr. 1, S. 58–78

[MS05] Mauro, Douglas R. ; Schmidt, Kevin J.: *Essential SNMP, Second Edition*. O'Reilly Media, Inc., 2005. – ISBN 0596008406

[NE00] Nuseibeh, Bashar ; Easterbrook, Steve: Requirements engineering: a roadmap. In: *Proceedings of the Conference on The Future of Software Engineering*. New York, NY, USA : ACM, 2000 (ICSE '00). – ISBN 1–58113–253–0, S. 35–46

[NFG+06] Northrop, Linda M. ; Feiler, Peter H. ; Gabriel, Richard P. ; Goodenough, John B. ; Linger, Richard C. ; Longstaff, Thomas A. ; Kazman, Rick ; Klein, Mark H. ; Schmidt, Douglas ; Sullivan, Kevin ; al, et: *Ultra-Large-Scale Systems - The Software Challenge of the Future*. Software Engineering Institute, 2006. – ISBN 0–9786956–0–7

[Obj08] Object Management Group: *Software and Systems Process Engineering Meta-Model*. `http://www.omg.org/cgi-bin/doc?formal/08-04-01.pdf`. Onlinedokument, letzter Aufruf: 2015-01, 2008

[Obj11a] Object Management Group: *OMG Object Constraint Language (OCL), v2.3.1*. `http://www.omg.org/spec/OCL/2.3.1/PDF/`. Onlinedokument, letzter Aufruf: 2015-01, 2011)

[Obj11b] Object Management Group: *OMG Unified Modeling Language (OMG UML, Infrastructure), v2.4.1*. `http://www.omg.org/spec/UML/2.4.1/`

Infrastructure/PDF. Onlinedokument, letzter Aufruf: 2015-01, 2011)

[Obj11c] Object Management Group: *OMG Unified Modeling Language (OMG UML, Superstructure), v2.4.1*. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF. Onlinedokument, letzter Aufruf: 2015-01, 2011)

[Obj12] Object Management Group: *OMG Systems Modeling Language (OMG SysML), v1.3*. http://www.omg.org/spec/SysML/1.3/PDF. Onlinedokument, letzter Aufruf: 2015-01, 2012)

[Obj13] Object Management Group: *Meta Object Facility (MOF) core specification, v2.4.1*. http://www.omg.org/spec/MOF/2.4.1/PDF. Onlinedokument, letzter Aufruf: 2015-01, 2013)

[Ral02] Ralyté, Jolita: Requirements Definition for the Situational Method Engineering. In: *Proceedings of the IFIP TC8 / WG8.1 Working Conference on Engineering Information Systems in the Internet Context*. Deventer and The Netherlands and The Netherlands : Kluwer, B.V., 2002. – ISBN 1–4020–7217–1, S. 127–152

[RJ00] Rising, Linda ; Janoff, Norman S.: The Scrum software development process for small teams. In: *Software, IEEE* 17 (2000), Nr. 4, S. 26–32. – ISSN 0740–7459

[RR99] Robertson, Suzanne ; Robertson, James: *Mastering the requirements process*. Harlow : Addison-Wesley, 1999 http://www.gbv.de/dms/ilmenau/toc/267718810.PDF. – ISBN 0201360462

[RR01] Ralyté, Jolita ; Rolland, Colette: An Assembly Process Model for Method Engineering. In: Dittrich, Klaus (Hrsg.) ; Geppert, Andreas (Hrsg.) ; Norrie, Moira (Hrsg.): *Advanced Information Systems Engineering* Bd. 2068. Springer Berlin / Heidelberg, 2001, S. 267–283

[Rup03] Rupp, Chris: Reden Sie Klartext, Psychotherapie für Anforderungen in der Softwareentwicklung. In: *manage it* 6 (2003), S. 38–42

[Rup09] Rupp, Chris: *Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis.* 5., aktualisier-

te und erweiterte Aufl. München : Carl Hanser Verlag GmbH und CO. KG, 2009. – ISBN 3446418415

[Sau11a] Sauer, Stefan: Applying Meta-Modeling for the Definition of Model-Driven Development Methods of Advanced User Interfaces. In: Hussmann, Heinrich (Hrsg.) ; Meixner, Gerrit (Hrsg.) ; Zuehlke, Detlef (Hrsg.): *Model-Driven Development of Advanced User Interfaces* Bd. 340. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–14561–2, S. 67–86

[Sau11b] Sauer, Stefan: *Systematic Development of Model-based Software Engineering Methods*. Paderborn, University of Paderborn, Diss., 2011

[SB01] Schwaber, Ken ; Beedle, Mike: *Agile Software Development with Scrum*. 1st. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2001. – ISBN 0130676349

[Sch06] Schmidt, Douglas C.: Model-Driven Engineering. In: *IEEE Computer* 39 (2006), Nr. 2. `http://www.truststc.org/pubs/30.html`

[SH96] Slooten, Kees van ; Hodes, Bert: Characterizing IS development projects. In: *Proceedings of the IFIP TC8, WG8.1/8.2 working conference on method engineering on Method engineering : principles of method construction and tool support: principles of method construction and tool support*. London and UK and UK : Chapman & Hall, Ltd, 1996, 29–44

[Som07] Sommerville, Ian: *Software Engineering*. 8. Pearson Studium, 2007. – ISBN 978–3–8273–7257–4

[SOP09] SOPHISTEN, Die: *IVENA Gliederung - Nichtfunktionale Anforderungen*. `http://www.sophist.de/infopool/downloads/offener-downloadbereich.html`, Website, last visit: 2011-08, 2009

[Spi08] Spijkerman, Michael. and Meyer, Matthias.: *Beratende Begleitung bei der MMB-Softwareentwicklung – Bericht über die erste Phase*. 2008

[Spi09] Spijkerman, Michael and Meyer, Matthias: *Beratende Begleitung bei der MMB-Softwareentwicklung – Bericht über die zweite Phase*. 2009

[Spi10] Spijkerman, Michael and Meyer, Matthias: *Beratende Begleitung bei der Spezifikation von Blade-Systemen – Bericht über die dritte Phase*. 2010

[Spi11] Spijkerman, Michael and Sauer, Stefan: *Entwicklung einer Spezifikationsmethodik für Server-Systeme am Beispiel der Management Platform MMP – Abschlussbericht des Projekts Entwicklung einer Spezifikationsmethodik für Blade-Server-Systeme anhand des MMB-NXT*. 2011

[Spi12] Spijkerman, Michael and Sauer, Stefan.: *Entwicklungsmethodik fu¨r Server- Management-Komponenten basierend auf dem Common Information Model – Abschlussbericht des Projekts Entwicklungsmethodik für Management-Komponenten von Server-Systemen*. 2012

[Spi13a] Spijkerman, Michael: Ein pragmatischer Ansatz zur Entwicklung situationsgerechter Entwicklungsmethoden. In: Kuhrmann, M. (Hrsg.) ; D. M. Fern'andez, D. M. (Hrsg.) ; Linsen, O. (Hrsg.) ; Knapp, A. (Hrsg.): *Proceedings of Modellierung von Vorgehensmodellen - Paradigmen, Sprachen, Tools (MVV2013) at SE 2013*, 2013, S. 425–434

[Spi13b] Spijkerman, Michael and Sauer, Stefan: *Entwicklungsmethodik für Management-Komponenten von Server-Systemen – Abschlussbericht des Projekts 2013 Step1*. 2013

[Spi13c] Spijkerman, Michael and Sauer, Stefan: *Entwicklungsmethodik für Server- Management-Komponenten basierend auf dem Common Information Model – Abschlussbericht des Projekts Entwicklungsmethodik für Management-Komponenten von Server-Systemen*. 2013

[Spi14] Spijkerman, Michael. and Sauer, Stefan.: *Entwicklungsmethodik für Management-Komponenten von Server-Systemen – Abschlussbericht des Projekts 2013 Step2*. 2014

[Sta09] Stadtler, Dilek: *Eine generische Methode zur unternehmens- bzw. projektspezifischen Festlegung von Vorgehensmodellen zur Entwicklung von Software*. Paderborn, Universität Paderborn, Diss., 2009

[The04] The Open Group: *CMPI, Issue 1.0*. https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?publicationid=12003. Onlinedokument, letzter Aufruf: 2015-01, 2004)

[Wei08] Weilkiens, Tim: *Systems Engineering mit SysML/UML: Modellierung, Analyse, Design*. 2., aktualis. u. erw. dpunkt Verlag, 2008. – ISBN 3898645770

A Anhang

A.1 Beschreibung Verwaltungsfunktionalität

Dieser Abschnitt gibt eine detaillierte Beschreibung der Begriffe des System Managements an.

Tabelle A.1: *Begriffe des System Managements*

Asset Managment
Das Asset Management dient zur Inventarisierung von Hardwareelementen. Das Setzen und Auslesen von Inventarnummern in einem nichtflüchtigen Speicher eines Hardware-Elements wird vorausgesetzt.
Redirection
Ein Service für die Umleitung von Datenströmen, wie Audiodaten, Videodaten, Kommandozeile oder Eingabegeräten auf externe Geräte, wird angeboten. Der Anwendungsfall ist die Möglichkeit des entfernten Zugriffs auf ein zu verwaltendes Hardwareelement. Beispielsweise ist es möglich über ein Verwaltungsmodul eines Server-Systems zwischen der Bildschirmausgabe einzelner Server umzuschalten. Die Infrastruktur muss eine physikalische Verbindung mit ausreichender Bandbreite bereitstellen.
Configuration Management
Das Konfigurationsmanagement ist der Oberbegriff für das allgemeine Lesen und Schreiben von Konfigurationsparametern auf der zu verwaltenden Hardware.
Inventory Management / Hardware Discovery

Die Inventarverwaltung beschreibt das Lesen von hardwarespezifischen Daten und das Identifizieren von Hardwarekomponenten eines Servers oder Hardwaremodule eines Server-Systems. Modulare Server-Systeme können ihre Konfiguration zur Laufzeit verändern. Der Austausch von Systemmodulen zur Laufzeit ist eine zu unterstützende Funktion. Dabei ist sicher zu stellen, dass die Verwaltungsfunktionalität die Hardwareänderungen erkennt. Hierfür werden Erkennungs- und Identifikationsmechanismen benötigt. Obwohl eine ähnliche Funktionalität mit Hot-Plug jedoch aus der Server-Entwicklung bekannt ist, sind die, auch mit dem Begriff Hot-Swapping zusammengefassten, Funktionen Hot-Add und Hot-Remove beim modularen Server-System als risikobehaftet anzusehen. Betrachten wir keine mechanischen und elektrotechnischen Schwierigkeiten, die zu Schäden an der Hardware führen können, bleibt das Problem, dass aufgrund von unterschiedlichen Ausprägungen der Serverhardware keine robusten Verwaltungsschnittstellen vorausgesetzt werden können. Unterschiedliche Server-System-Module bieten unterschiedliche Konfigurationseigenschaften über unterschiedliche Verwaltungsschnittstellen an. Die elektronischen Schwierigkeiten können in dem Fall der robusten Identifikation der Hardware-Elemente über electronic keying (E-Keying) Mechanismen überwunden werden. E-Keying überprüft vor der Inbetriebnahme des Hardware-Moduls, ob die elektrischen Schnittstellen kompatibel zueinander sind, damit beispielsweise Hardwaredefekte durch nicht passende elektrische Spannungen ausgeschlossen werden können.

Cooling and Noise Management

Diese Funktionalität umfasst alle Systemeigenschaften, die Auswirkungen auf die Betriebstemperatur eines Server-Systems haben. Insbesondere die Abfuhr der entstehenden Abwärme wird fast ausschließlich über mechanische Lüfter gewährleistet. Eine hohe Last der mechanischen Lüfter produziert Geräuschemission. Betriebsumgebungen definieren Grenzwerte für die Geräuschemission. Technische Voraussetzungen schränken die erlaubte Betriebstemperatur der Server-System-Hardware ein. Eine Optimierungsfunktion über verschiedene Sensoren zur Messung von Temperatur, aktuellen Lüfterdrehzahlen, Lastwerten, Luftdruck, Stromaufnahme berechnen die Sollwerte für verschiedene Aktoren zur Änderung der Lüfterdrehzahlen, Lastverteilung oder Stromaufnahme. Bei einzelnen Servern wird die Lüftersteuerung über Regelungstechnik in Hardware umgesetzt. Große modulare Server-Systeme benötigen Unterstützung durch Verwaltungsmodule mit komplexer Software.

Deployment

Deployment beschreibt die Bereitstellung und Installation von Betriebssystemsoftware auf einzelne Server. Aufgrund der hohen Zahl von Servern werden Anforderungen bezüglich der automatisierten Installation eines Betriebssystems auf mehrere Server erhoben. Diese Anforderung bedarf Speicherplatzgrößen bis hin zu mehreren Gigabyte und Infrastrukturen, die den Durchsatz des Transfers dieser Datengrößen in einem vertretbaren Zeitrahmen ermöglichen. Gemeinhin wird die Verwaltungsumgebung jedoch durch die geringen Ressourcen eingebetteter Systeme charakterisiert. Entweder werden aktuellere Hardware-Architekturen der Verwaltungsumgebung benötigt oder der Installationsprozess wird gestartet und nutzt danach für die Datenübertragung autonom die Infrastruktur der Produktivumgebung.

Eventing / Logging

Eventing umfasst die Definition von Ereignissen, die autonom vom System festgestellt werden können. Darunter fallen Statusänderungen der Hardware und Software sowie Auditing-Informationen. Hierbei steht die Umsetzung von Informationsklassifizierung und Filtermechanismen im Vordergrund, damit ein Interessent der Systemereignisse genau konfigurieren kann, welche Events über welche Schnittstelle versendet werden. Das Speichern und Weiterleiten dieser Ereignisse fällt ebenfalls unter den Begriff Eventing.

Fault Management

Fehlermanagement bedeutet neben dem Erkennen von Fehlerzuständen auch die Umsetzung von Fehlerreaktionsmaßnahmen. Es können Redundanzkonzepte greifen, indem redundante Elemente die fehlerhaften Elemente ersetzen. Das System oder die Systemelemente können durch Ausschalten in einen sicheren Zustand oder durch Neustart in einen definierten Zustand überführt werden. Bei Fehlern, die zeitweise toleriert werden können oder einen manuellen Eingriffen erfordern, informieren Systemereignisse über den Fehlerzustand mit Zusatzinformation für die Fehlerbehebung und Wiederherstellung eines sicheren Systemzustands.

Health and Status Management

Das Health and Status Management erweitert Fault Management und Eventing dahingehend, dass die Auswirkung von Fehlern oder Zustandsänderungen auf weitere Systemkomponenten oder das Gesamtsystem betrachtet und analysiert wird. Wird beispielsweise eine redundante Komponente nach einem Fehlerfall als Ersatz für eine fehlerhafte Komponente eingesetzt, ist der aktuelle fehlerfreie Betrieb des Systems weiterhin sichergestellt. Die Ausfallsicherheit des Systems wird jedoch beeinträchtigt. Das Status Management übernimmt die Anpassung des Systemstatus.

Time Management

Die Verwaltung von Zeit in einem modularen Server-System hat Auswirkungen auf die Benutzerfreundlichkeit. Eine einheitliche Zeit in der Verwaltungsarchitektur über alle zu verwaltenden Systemmodule erleichtert die Konfiguration. In Szenarien, bei denen jedes Systemmodul seine eigene Zeit verwaltet, ist es schwierig einen mit Zeit behafteten Parameter zu setzen oder den Zusammenhang von Ereignissen über Zeitstempel zu erkennen. Die Möglichkeit verschiedene Uhren im System über die Verwaltungsschnittstellen zu synchronisieren wird vorausgesetzt.

Power Management

Mit Power Management wird die Eigenschaft beschrieben Hardwaremodule über die Verwaltungsumgebung an- und auszuschalten. Insbesondere bei komplexer Hardware ist das Einschalten ein zeitbehafteter Vorgang. Bei einem modularen Server-System gibt es Abhängigkeiten zwischen einzelnen Hardwaremodulen, die eine Einschaltreihenfolge vorgeben. Der Einschaltalgorithmus eines einzelnen Servers (Boot-Vorgang) muss konfiguriert werden (Power-Cycle-Management). Weitere Konfigurationseigenschaften bezüglich des Energieverbrauchs werden angeboten.

- Power Limiting: Softwarebasierte Einschränkung des Stromverbrauchs eines Hardwaremoduls.
- Power Capping: Hardwarebasierte Einschränkung des Stromverbrauchs eines Hardwaremoduls.

Darauf aufbauend werden zeit-, termin- und umgebungsgesteuerte Konfigurationsmöglichkeiten angeboten. So können Stromkosten eingespart, bedarfsgerechte Rechenkapazitäten bereitgestellt und Rahmenbedingungen der Betriebsumgebung eingehalten werden. Die Optimierung der Konfigurationsmöglichkeiten wird ermöglicht, wenn das zu optimierende System die Möglichkeiten zur Messung und Speicherung von Stromverbrauchsdaten anbietet.

Redundancy Management

Redundanz bedeutet die Bereitstellung von funktional und technisch gleichen Hardwareelementen, damit die Warscheinlichkeit eines Ausfalls des Server-Systems reduziert wird. Im Fehlerfall wird ein redundantes Element die Aufgaben des defekten Elements übernehmen. Verschiedene Arten von Redundanz werden hier benutzt. Bei der heißen Redundanz (Hot-Spare) sind alle redundanten Hardwareelemente aktiv und der Ausfall ein oder mehrerer Elemente kann durch die Ressourcen weiterer aktiver Elemente kompensiert werden. Kalte Redundanz stellt redundante Hardwareelemente bereit, die jedoch deaktiviert oder im Stand-by-Zustand sind. Sie werden erst im Fehlerfall, in Folge von Fehlerreaktionsmaßnahmen, aktiviert. Dabei müssen die Aktivierungszeiten vom System akzeptiert werden. Das numerische Verhältnis von redundanten Hardwareelementen wird durch $n + m$ beschrieben, wobei n die Anzahl der aktiven und m die Anzahl der redundanten Hardwareelemente beschreibt. Festplatten, Stromversorgungsmodule, Lüfter oder Server sind die typischen redundant ausgelegten Elemente in einem Server-System. Die konfigurierbaren Eigenschaften der Redundanz hängen von der technischen Infrastruktur des Systems ab.

User Management

User Management umfasst das Anlegen und Verwalten von Benutzern und deren Zugriffsrechte in der Verwaltungsumgebung eines Server-Systems. Hierbei stehen die Aufgaben Authentifizierung und Autorisierung im Vordergrund. Authentifizierung bedeutet der Nachweis einer definierten Identität (engl.: Identity Management) und Authentifizierung reglementiert Zugriffsrechte einer Identität auf Hardwareelemente oder Konfigurationsdaten. Üblich ist die Anwendung von Rollenkonzepten (engl.: Role Based Access Control).

Update Management

Aufgrund der Weiterentwicklung der Verwaltungssoftware, auch nach der Markteinführung der Server-Systeme, ist es notwendig diese während des Produktivbetriebs im System zu aktualisieren. Die Problematik durch den Umfang der Installationsdaten, wie beim Deployment, besteht hier nicht, da der Umfang der Installationsdaten der Verwaltungssoftware wesentlich geringer als bei der Betriebssystemsoftware der Produktivumgebung ist. Der Update-Prozess der Verwaltungssoftware ist jedoch risikobehaftet, da bei einem fehlerhaften Update die Gefahr des Kontrollverlusts bis hin zur irreparablen Schädigung der Hardwareelemente besteht. Für den Fehlerfall werden Fallback-Routinen zur Wiederherstellung vorheriger Softwareversionen benötigt.

Localization

Der Server-System-Markt ist global. Server-Systeme kommen weltweit zum Einsatz. Aufgrund dessen muss die Verwaltungssprache des Systems konfigurierbar sein. Die Informationsinhalte, die über die Verwaltungsschnittstellen zur Verfügung stehen, müssen in mehreren Sprachen zur Verfügung stehen. Die Sprache muss eingestellt werden können.

Security Management

Unter dem Begriff Sicherheit fallen hier sämtliche Konfigurationseigenschaften mit sicherheitsrelevanten Auswirkungen. Während Autorisierung und Authentifikation schon vom User Management betrachtet werden, sind hier die technischen Eigenschaften der Schnittstellen und der Infrastruktur gemeint. Das Aktivieren und Deaktivieren von Schnittstellen, das Verschlüsseln von Datenströmen oder das Setzen von betriebsspezifischen Richtlinien für Passwörter sind relevante Beispiele. Diese Konfigurationen beziehen sich auf die Verwaltungs- und Produktivumgebung.

Network Management

Unter dem Begriff des Netzwerkmanagements fallen alle Konfigurationsmöglichkeiten für die Netzwerkumgebung. Die Einstellung von IP-Adressen und Netzwerkprotokollen zur Bildung der administrativen, logischen Netzwerkinfrastruktur steht hier im Vordergrund. In dieser Arbeit ist mit Netzwerkmanagement die Definition aus diesem Abschnitt gemeint, nicht die Definition nach ISO, wie in der Einleitung des Abschnitts 3.3 beschrieben.

Backup - Restore

Die Konfigurationsmöglichkeiten einzelner Server sind vielfältig. Wird ein Server-System mit einer großen Menge an Servern betrachtet, steigt der Konfigurationsaufwand proportional an. Oft unterscheiden sich die Konfigurationen zwischen einzelnen Servern in einer gemeinsamen administrativen Umgebung nicht wesentlich voneinander. Mit Backup-Restore-Funktionalität wird hier die einfache Übertragung einer Konfiguration eines Servers oder eines Server-Systems beschrieben. Auf diese Weise wird die gesamte lauffähige Konfiguration eines Servers oder Server-Systems gesichert und auf einem weiteren Server oder auf einem weiteren Server-System eingespielt.

Virtualization

Unter Virtualization wird im Allgemeinen die Bereitstellung logischer Elemente als Simulation physikalischer Elemente auf Basis realer Hardware verstanden. Beispielsweise gibt es virtuelle Server, die auf Basis von Abstraktions-Schnittstellen einen Hardware-Server simulieren. Auf diese Weise führt ein realer Hardware-Server verschiedene virtuelle Server aus. Das Konzept der Virtualisierung wird auch im Bereich der Speichermedien angewendet, indem ein physikalisches Medium verschiedene virtuelle Medien darstellen kann. Im Bereich des Netzwerk-Managements ist es möglich physikalische Netzwerke durch logische Netzwerke zu strukturieren. Eine Virtualisierung benutzt eine Middleware, die entsprechende Schnittstellen zur Abstraktion von physikalischen Elementen anbietet.

A.2 Die Verwaltungsarchitekturen SNMP und IPMI

Dieser Abschnitt beschreibt die beiden Verwaltungsarchitekturen SNMP und IPMI und stellt dadurch zusätzliche Grundlagen für die Erkenntnisse in Abschnitt 3.4 dar, die allgemeine strukturelle Eigenschaften in Server-Systemen definieren.

A.2.1 SNMP

Aufgabe von SNMP

Das Simple Network Management Protocol (SNMP) ist 1988 entstanden und wird von der Internet Engineering Task Force (IETF) standardisiert und verwaltet. Mit SNMP wird die entfernte Verwaltung von IP-basierten Netzwerkkomponenten ermöglicht. Ursprünglich war das Protokoll für die Verwaltung von Netzwerkroutern gedacht, jedoch wurde der Standard ausgeweitet und weitere Systemelemente können definiert werden. Das sind jegliche Art physikalischer Elemente

wie Drucker, Modems, Netzteile und Server aber auch Softwareentitäten wie Betriebssysteme, Webserver oder Datenbanken (siehe auch [MS05]).

Funktionalität von SNMP

Die Systeminformationen können verwaltet und überwacht werden, indem Informationen ausgelesen und verändert werden können. Dadurch wird es indirekt möglich einfache Zustandswechsel im verwalteten Element herbeizuführen.

Historie von SNMP

SNMP ist in der Version 1, 2 und 3 spezifiziert. Die Spezifikation ist in Form von Request for Comments[1] (RFC) Dokumenten vorhanden. RFCs beschreiben Diskussionen über das Internet in einer formalen Form und können neben informativen Informationen auch Standards definieren[2]. RFC1157 stellt den historischen Standard SNMPv1 dar, SNMPv2 wird in den Standards RFC 3416, RFC 3417 und RFC 3418 erläutert. 2002 wird SNMPv3 schließlich in den Standards RFC 3410, RFC 3411, RFC 3412, RFC 3413, RFC 3414, RFC 3415, RFC 3416, RFC 3417, RFC 3418 und RFC 2576 zusammengefasst. Die aktuelle Entwicklung ist der Support von SNMPv2 mit Kompatibilität zu SNMPv1. SNMPv3 verfolgt hauptsächlich die Weiterentwicklung im Hinblick auf Sicherheit und der sicheren Kommunikation zwischen den SNMP-Entitäten.

SNMP Architektur

Die Architektur von SNMP besteht hauptsächlich aus Network Management Stations (NMS) und Agenten. Die NMS übernehmen einfache Verwaltungsaufgaben. Agenten stellen entsprechende Informationen der zu verwaltbaren Komponenten bereit (siehe Abbildung A.1).

NMS und Agenten sind über ein TCP-basiertes Netzwerk miteinander verbunden. Für jede Verwaltungskomponente gibt es mindestens einen Agenten. Es ist ein Masteragent vorhanden, der mehrere Subagenten hat. Die Kommunikation zwischen Masteragent und Subagenten wird proprietär oder auf Basis von Quasi-Standards wie dem SNMP Multiplex (SMUX) Protokoll, beschrieben in RFC 1227, realisiert. Ein Agent überwacht mehrere Managed Objects (MO). Die MOs werden in Form einer Management Information Base (MIB) beschrieben. Eine MIB ist eine textbasierte Beschreibung auf Basis der Structure of Management Information (SMI). SMI definiert die Spezifikation einzelner MOs wiederum über die Abstract Syntax Notation One (ASN.1). Es werden ausschließlich strukturelle Informationen festgelegt.

1 http://www.rfc-editor.org, letzter Aufruf: 2015-01
2 http://www.rfc-editor.org/rfcxx00.html, letzter Aufruf: 2015-01

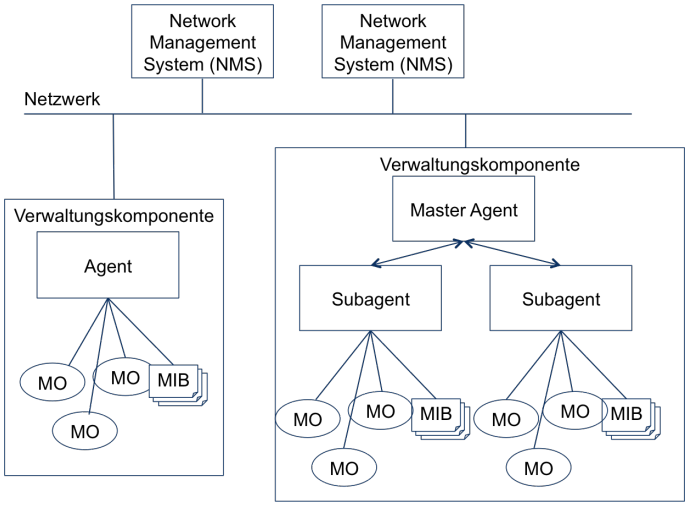


Abbildung A.1: *SNMP-Architektur (siehe [BS98])*

Managed Objects

MOs werden beschrieben durch einen Object Identifier (OID), welcher ein MO eindeutig identifiziert. Der Datentyp (SYNTAX), die Zugriffsrechte (ACCESS) und der Status (STATUS) werden zusammen mit einer Beschreibung (DESCRIPTION) des MOs festgelegt.

Object Identifier

OIDs werden innerhalb einer MIB eindeutig verteilt. Zusammen mit der Hierarchie der MIB, die in einem Wurzelbaum organisiert wird, wird die eindeutige OID-Vergabe realisiert. Für die MIB-Identifikation gibt es einen numerischen Wert sowie eine Zeichenkette in Klartext (siehe Abbildung A.2).

Abbildung A.2: *Struktur der OIDs*

Management Information Base

Ein Agent unterstützt mehrere MIBs. Beispielsweise ist die Unterstützung der MIB-II jedoch obligatorisch. Sie beschreibt notwendige TCP/IP-Informationen. Die Abbildung A.2 beschreibt den Pfad zur MIB-II (numerisch: 1.3.6.1.2.1, Klartext iso.org.dod.internet.mgmt.mib-2). Es gibt auch die Möglichkeit eigene OEM-spezifische MIBs zu definieren. OEM-MIBs können unter iso.org.dod.internet.private.enterprises.IANA-Nummer (numerisch: 1.3.6.1.4.1.x) eingetragen werden. Jeder Beliebige darf bei der Internet Assigned Numbers Au-

thority[3] (IANA) eine IANA-Nummer beantragen. Beispielsweise sind die IANA-Nummern 231 und 10368 für FTS eingetragen.

Der folgende Ausschnitt der OEM-spezifischen s31.mib von Fujitsu Technology Solutions GmbHbeschreibt die Informationen, die ein Blade-Server über SNMP bereitstellt.

```
1  fts                                                    OBJECT IDENTIFIER ::= { enterprises 10368 }
2  server                                                 OBJECT IDENTIFIER ::={quanta1 }
3  high-density                                           OBJECT IDENTIFIER ::= {server 1
4  } s31MgmtAgent                                         OBJECT IDENTIFIER ::= { high-density 1 }
5
6          s31AgentInfo                                   OBJECT IDENTIFIER ::= { s31MgmtAgent 1 }
7  ...
8          s31ManagementBlade                             OBJECT IDENTIFIER ::= { s31MgmtAgent 2 }
9          s31SystemInfo                                  OBJECT IDENTIFIER ::= { s31MgmtAgent 3 }
10 ...
11         s31ServerBlade                                 OBJECT IDENTIFIER ::= { s31MgmtAgent 4 }
12                 s31SvrCtrlInfo                         OBJECT IDENTIFIER ::= { s31ServerBlade 1 }
13                 s31SvrBlade                            OBJECT IDENTIFIER ::= { s31ServerBlade 2 }
14 ...
15
16 s31SvrBladeTable OBJECT-TYPE
17         SYNTAX  SEQUENCE OF S31SvrBladeEntry
18         ACCESS  not-accessible
19         STATUS  mandatory
20         DESCRIPTION
21                 "Table with data of server blade basic information"
22         ::={ s31SvrBlade 1 }
23
24 s31SvrBladeEntry OBJECT-TYPE
25         SYNTAX  S31SvrBladeEntry
26         ACCESS  not-accessible
27         STATUS  mandatory
28         DESCRIPTION
29                 ""
30         INDEX   { s31SvrBladeId }
31         ::={ s31SvrBladeTable 1 }
32
33 S31SvrBladeEntry ::=
34         SEQUENCE {
35                 s31SvrBladeId                  INTEGER,
36 ...
37                 }
38
39 s31SvrBladeId OBJECT-TYPE
40         SYNTAX  INTEGER (1..20)
41         ACCESS  read-only
42         STATUS  mandatory
43         DESCRIPTION
44                 "Server blade number, index into the server blade table (1 based)"
45         ::={ s31SvrBladeEntry 1 }
```

Listing A.1: *Auszüge aus der s31.mib*

Beispielsweise wird die OID s31SvrBladeId aus dem obigen Beispiel[4] textuell wie folgt referenziert: OID Beispiel

iso.org.dod.internet.private.enterprises.quanta.server.high-density.
s31MgmtAgent.s31ServerBlade.s31SvrBlade.s31SvrBladeTable.s31SvrBladeId

3 http://www.iana.org/, letzter Aufruf: 2015-01
4 Der MIB-Eintrag 31SvrBladeEntry ist als not-accessible definiert und besitzt keine OID.

Die äquivalente numerische Referenzierung lautet:
1.3.6.1.4.1.7244.1.1.1.4.2.1.1

Ausführung von SNMP

SNMP definiert einfache Kommandos für die Abfrage der Agenten vom NMS nach bestimmten Informationen, die in einer MIB über OIDs definiert werden. Der Agent sammelt diese Informationen und schickt die Antwort an den NMS. Durch diese Eigenschaft muss der NMS seine Agenten über einen Poll-Mechanismus abfragen, um Statusänderungen zeitnah mitzubekommen. Eine Besonderheit wird durch Traps beschrieben, das sind nicht aufgeforderte Nachrichten der Agenten an den NMS. Auf diese Weise können dem NMS beliebige Systemereignisse mitgeteilt werden. Traps werden in der Regel für die Eskalation von Fehlersituationen eingesetzt.

Vor- und Nachteile

Der Vorteil von SNMP ist die einfache Struktur und Definition. Ein wesentlicher Nachteil ist die Sicherheit[5]. SNMPv1 und die ersten Revisionen von SNMPv2 bieten keine Möglichkeit des autorisierten Zugriffs auf die Verwaltungsinformationen sowie die fehlende Möglichkeit SNMP-Datenpakete verschlüsselt zu übertragen. SNMPv3 ermöglicht Autorisierung, ist jedoch nicht sehr weit verbreitet. Das hängt mit der Performance zusammen. Aufgrund des Polling-Mechanismus wird ein sehr großer Datenoverhead erzeugt. Weiterhin ist die Abfrage von MOs via SNMP nicht mehr State-of-the-Art. SNMPv2 wird meist noch aus Kompatibilitätsgründen unterstützt. Der Trend SNMPv3 zu unterstützen wird erwartet, sobald der Markt eine erhöhte Sicherheit von Verwaltungsnetzwerken wünscht. Weiterhin bleibt zu bemerken, dass SNMP nicht in der Lage ist komplexe Verwaltungsfunktionen aufzurufen.

A.2.2 IPMI

Das Intelligent Platform Management Interface (IPMI) standardisiert eine Verwaltungsarchitektur zur Out-Of-Band-Verwaltung von Server-Systemen. Darauf aufbauend werden standardisierte Schnittstellen zum Zugriff auf Systeminformationen bereitgestellt. Die aktuelle Version des Standards ist IPMIv2.0[Int09] und wird von den Server-System-Herstellern Intel Corporation, Hewlet-Packard Company, NEC Corporation und Dell inc. entwickelt.

Die IPMI-Architektur wird in Abbildung A.3 dargestellt. Das Hauptelement ist der *Baseboard Management Controller (BMC)*. Ein BMC wird direkt auf dem Mainboard einer Server-Hardware bereitgestellt. Der BMC besitzt einen *seriellen Controller*, der über einen *seriellen Schalter* mit dem *seriellen Konnektor* des Mainboards verbunden wird. Auf diese Weise wird über die externe Schnittstelle auf den BMC zugegriffen. Weiterhin wird der BMC über einen *SMBus* an den *LAN Controller* des Mainboards angeschlossen,

5 https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02144.html, Webseite, letzter Aufruf: 2015-01

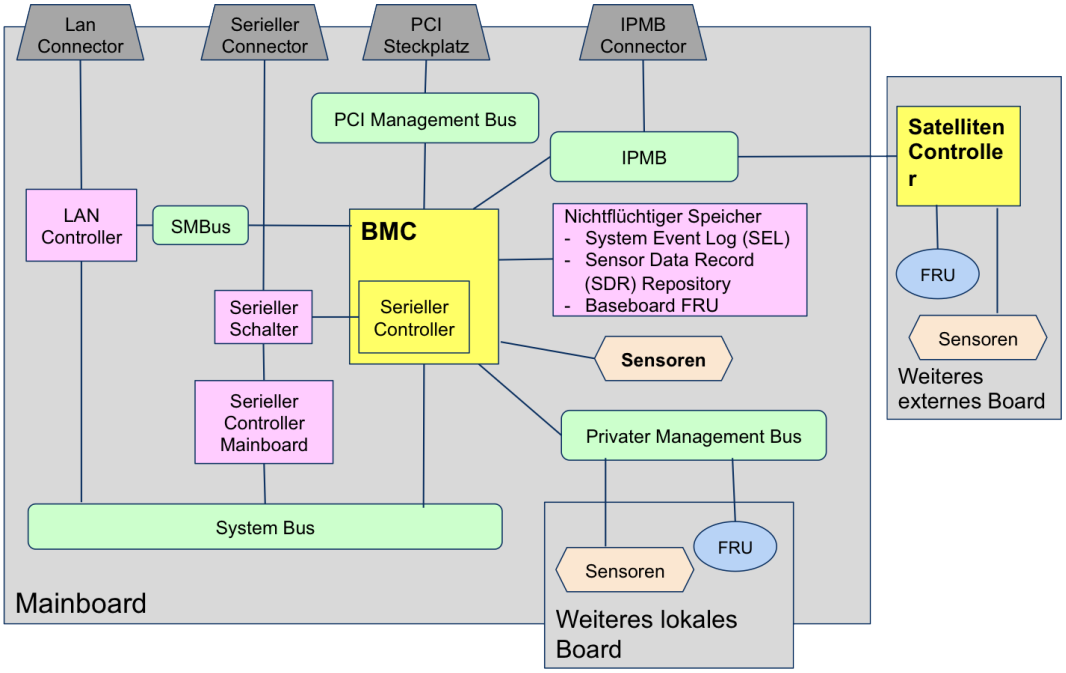


Abbildung A.3: *IPMI-Architektur (siehe [Int09] Abschnitt 1.7.3)*

dadurch wird es möglich über die externe LAN-Schnittstelle auf den BMC zuzugreifen. Eine weitere implizite externe Schnittstelle ist der *PCI Management Bus*, über den Erweiterungskarten an das Mainboard angeschlossen werden können und vom BMC verwaltet werden. Interne Boards, wie beispielsweise Speicher- oder Prozessor-Boards, können über *private Management Busse* an den BMC angeschlossen werden. Externe Boards, wie beispielsweise ein Chassis-Board oder beliebige weitere Module über *IPMB-Connector*, können über den *Intelligent Platform Management Bus (IPMB)* an den BMC geklemmt werden. Bei diesen externen Modulen wird jedoch ein *Satelliten Controller* erwartet, der die Systeminformationen des externen Moduls bereitstellt. Weitere interne oder externe Boards können *Sensoren* oder *Field Replacement Unit (FRU)* Daten enthalten. FRU ist ein standardisierter Datensatz in nicht flüchtigen read-only-Speichern. Der BMC verwaltet intern eigene Sensoren für Spannungen, Temperaturen, Lüfter und weitere. Der BMC muss Zugriff auf einen *nichtflüchtigen read-write fähigen Speicher* erhalten. In diesem Speicher liegen die FRU-Daten des BMC und Sensor Data Records, die eine Definition aller vom BMC verwaltbaren Sensoren enthalten. Der System Event Log (SEL), der sämtliche vom BMC erkannten Systemereignisse in Form von SEL-Einträgen enthält, ist ebenso in diesem Speicher hinterlegt.

Der BMC ist ein eigenständiger Controller, der mit der Standby-Spannung des Servers versorgt wird. Dadurch wird die Verwaltung des Servers im ausgeschalteten Zustand gewährleistet.

IPMI definiert die externen Zugriffsprotokolle über serielle und LAN-Schnittstellen sowie die internen Zugriffsprotokolle SMBus und IPMB. Weiterhin sind standardisierte Kommandos zum Zugriff auf die Verwaltungsfunktionalität, die ebenfalls in [Int09] definiert werden, vorhanden. Verwaltungsinformationen werden von FRU-Daten, Sensoren und dem Server über den System-Bus bereitgestellt.

Ein FRU ist eine Baugruppe, die im Feld (Field) ausgetauscht werden kann. Unterstützt das FRU IPMI, muss es einen standardisierten Datensatz zur eindeutigen Identifikation enthalten. Dieser standardisierte Datensatz wird in [Int99] definiert. In Abbildung A.4 wird die Datensatzdefinition der *Board Info Area* vorgestellt. Das sind die Daten, die nach IPMI-Standard von internen oder externen Boards bereitgestellt werden müssen, beispielsweise Datum der Herstellung, Name des Herstellers, Produktname, Seriennummer und weitere.

BOARD INFO AREA

field length	field
1	Board Area Format Version 7:4 - reserved, write as 0000b 3:0 - format version number = 1h for this specification.
1	Board Area Length (in multiples of 8 bytes)
1	Language Code (See section 15)
3	**Mfg. Date / Time** Number of minutes from 0:00 hrs 1/1/96. LSbyte first (little endian)
1	**Board Manufacturer** type/length byte
P	Board Manufacturer bytes
1	**Board Product Name** type/length byte
Q	Board Product Name bytes
1	**Board Serial Number** type/length byte*
N	Board Serial Number bytes*
1	**Board Part Number** type/length byte
M	Board Part Number bytes
1	**FRU File ID** type/length byte*
R	FRU File ID bytes* The FRU File version field is a pre-defined field provided as a manufacturing aid for verifying the file that was used during manufacture or field update to load the FRU information. The content is manufacturer-specific. This field is also provided in the Product Info area. Either or both fields may be 'null'.
xx	Additional custom Mfg. Info fields. Defined by manufacturing. Each field must be preceded by a type/length byte
1	C1h (type/length byte encoded to indicate no more info fields).
Y	00h - any remaining unused space
1	Board Area Checksum (zero checksum)

Abbildung A.4: *Beispiel FRU-Datendefinition Board Info Area (siehe [Int99] Kapitel 11)*

Ein Sensor Data Record (SDR) beschreibt Sensoren, die vom BMC verwaltet werden können. Die Abbildung A.5 zeigt einen Ausschnitt der generischen Definition eines kompakten Sensors. Ein SDR beschreibt einen *Header*, der allgemeine Verwaltungs- und

Versionierungs-Informationen enthält. *Key* Information beschreiben die Adressierung eines konkreten Sensors über einen Verwaltungsbus (I^2C oder IPMB). Der *Body* definiert die Daten des Sensors. Beispielsweise beschreibt die *EntityID* den Sensor-Typ (Voltage, Fan, Temperature, etc.) und die *EntityInstance* einen eindeutigen Identifier. Weitere Bytes, die nicht in der Abbildung enthalten sind, beschreiben beispielsweise Schwellwerte, Datentypen und Einheiten. Es wird ebenfalls ein Sensor Offset definiert, der letztendlich die Laufzeitdaten, also die konkreten Werte des Sensors beschreibt.

byte	Field Name	size	Description
	SENSOR RECORD HEADER		
1:2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. It is *not* related to the sensor ID.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **51h** for this specification. *BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits.*
4	Record Type	1	Record Type Number = 02h, Compact Sensor Record
5	Record Length	1	Number of remaining record bytes following.
	RECORD KEY BYTES		
6	Sensor Owner ID	1	[7:1] - 7-bit I^2C Slave Address, or 7-bit system software ID[1] [0] - 0b = ID is IPMB Slave Address, 1b = system software ID
7	Sensor Owner LUN	1	[7:4] - Channel Number The Channel Number can be used to specify access to sensors that are located on management controllers that are connected to the BMC via channels other than the primary IPMB. (Note: In IPMI v1.5 the ordering of bits 7:2 of this byte have changed to support the 4-bit channel number) [3:2] - FRU Inventory Device Owner LUN. LUN for Write/Read FRU commands to access FRU information. 00b otherwise. [1:0] - Sensor Owner LUN. LUN in the Sensor Owner that is used to send/receive IPMB messages to access the sensor. 00b if system software is Sensor Owner.
8	Sensor Number	1	Unique number identifying the sensor behind the given slave address and LUN. Code FFh reserved.
	RECORD BODY BYTES		
9	Entity ID	1	Indicates the physical entity that the sensor is monitoring or is otherwise associated. See *Table 43-13, Entity ID Codes.*
10	Entity Instance	1	[7] - 0b = treat entity as a physical entity per Entity ID table 1b = treat entity as a logical container entity. For example, if this bit is set, and the Entity ID is 'Processor', the container entity would be considered to represent a logical 'Processor Group' rather than a physical processor. This bit is typically used in conjunction with an Entity Association record. [6:0] - Instance number for entity. (See section *39.1, System- and Device-relative Entity Instance* Values for more information) 00h-5Fh system-relative Entity Instance. The Entity Instance number must be unique for each different entity of the same type Entity ID in the system. 60h-7Fh device-relative Entity Instance. The Entity Instance number must only be unique relative to the management controller providing access to the Entity.

Abbildung A.5: *Beispiel SDR-Datendefinition Compact Sensor (siehe [Int09] Abschnitt 43.2)*

Der System Event Log (SEL) ist ebenfalls über IPMI spezifiziert. Der SEL ist ein Logbuch für Systemereignisse. Der BMC verfügt über Monitoring-Funktionalität und erkennt Systemereignisse. Beispielsweise wird das Überschreiten von Schwellwerten durch Sensoren definiert. Ein entsprechender textueller Eintrag wird in der SEL gespeichert.

Der IPMI-Standard definiert Standard-Kommandos auf den verschiedenen Schnittstellen auf die der BMC zugreifen kann. Diese Kommandos sind über die serielle oder die LAN-Schnittstelle ausführbar. Ein Beispiel des Get SEL Info Commands wird in Abbildung A.6 vorgestellt. Das Kommando definiert die Abfrage des Status des SEL. Jedes Kommando liefert einen Completion Code zurück, der den Status des Kommandos beschreibt. Das zurückgegebene Byte 2 beschreibt die SEL-Version, Byte 3 und 4 die Anzahl der Log-

Einträge, Byte 5 und 6 den verfügbaren freien Speicher des Logbuchs und weitere Informationen.

	byte	data field
Request Data	-	-
Response Data	1	Completion Code 81h = cannot execute command, SEL erase in progress
	2	SEL Version - version number of the SEL command set for this SEL Device. **51h** for this specification. (BCD encoded).*BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits.*
	3	Entries LS Byte - number of log entries in SEL, LS Byte
	4	Entries MS Byte - number of log entries in SEL, MS Byte
	5:6	Free Space in bytes, LS Byte first. FFFFh indicates 65535 or more bytes of free space are available.
	7:10	Most recent addition timestamp. LS byte first. Returns FFFF_FFFFh if no SEL entries have ever been made or if a component update or error caused the retained value to be lost.
	11:14	Most recent erase timestamp. Last time that one or more entries were deleted from the log. LS byte first.
	15	Operation Support [7] - Overflow Flag. 1=Events have been dropped due to lack of space in the SEL. [6:4] - reserved. Write as 000 [3] - 1b = Delete SEL command supported [2] - 1b = Partial Add SEL Entry command supported [1] - 1b = Reserve SEL command supported [0] - 1b = Get SEL Allocation Information command supported

Abbildung A.6: *Beispiel: Get SEL Info Command (siehe [Int09] Abschnitt 31.2)*

IPMI ist der momentane Quasi-Standard bei der Verwaltung von Server-Systemen. IPMI ermöglicht mit den genannten Eigenschaften die Überwachung der Hardware (Monitoring), Wiederherstellung und Neustarten des Servers (Recovery Control), Protokollierung von „out-of-range“-Zuständen der Hardware (Logging) und Inventarisierung der Hardware (Inventory).

A.3 MetaMe-Methodenbeschreibungssprache MMMB

Dieser Abschnitt beschreibt die Sprachdefinition der MetaMe-Methodenbeschreibungssprache MMMB.

abstrakte Syntax

Das Methodenmetamodell stellt die abstrakte Syntax in Form eines Metamodells dar. Für die Sprachdefinition fehlen nun eine Semantikbeschreibung der Elemente der abstrakten Syntax und eine konkrete Syntax, die das Aussehen der instanziierten Elemente des Metamodells definieren.

MMMB-UML-Profil

MMMB stellt eine graphische Modellierungssprache dar, die spezifisch für die Domäne der Methodenentwicklung ist. In dieser Arbeit wird MMMB als UML-Profil definiert, eine leicht gewichtige UML-Erweiterung (siehe Kapitel 18 in [Obj11c]).

UML-Profil-Erstellung

Ein Vorgehen für die Definition von UML-Profilen wird in [FV04] dargestellt. Es umfasst die folgenden Schritte.

1 Definition des Domänenmodells als Metamodell
2 Erstellung eines Stereotyps für jedes Metamodellelement mit dem Namen aus dem Metamodell
3 Anwendung jedes Stereotyps auf eine Meta-Klasse des UML-Metamodells
4 Definition der Attribute des Stereotyps, die die erweiterten Eigenschaften darstellen.
5 Erstellung domänenspezifischer Einschränkungen für das UML-Profil auf Basis von OCL-Constraints.

Für die Erstellung des „MMMB UML-Profils“ wurden die 5 Schritte nach [FV04] durchgeführt.

Das im Schritt 1) definierte Metamodell findet sich in Abschnitt 2.5 in Abbildung 2.17. Die Ergebnisse der Schritte 2), 3), 4) und 5) werden in den folgenden Abschnitten vorgestellt.

Für jedes Sprachelement werden die folgenden Informationen angegeben.

Profil-Definition: Die Ableitung vom UML-Sprachelement. Wenn ein Profil-Diagramm angegeben wird, wird das UML-Metamodell-Element stereotypisiert als «metaclass» angegeben. Es enthält lediglich die für das abgeleitete UML-Profil-Element relevanten Attribute. Weitere Informationen sind der UML-Superstructure-Spezifikation 2.4.1 [Obj11c] zu entnehmen. Das abgeleitete UML-Profil-Element wird stereotypisiert als <stereotype> angegeben. Das UML-Profil-Element übernimmt die Attribute des UML-Metamodell-Elements.

Beschreibung und Semantik: Das UML-Profil-Element übernimmt die semantischen Eigenschaften des UML-MetaModell-Elements (Metaklasse). In diesem Abschnitt werden konkrete Referenzen zur UML-Spezifikation angegeben. Weiterhin wird deskriptiv die semantische Erweiterung oder Einschränkung der UML-Semantik beschrieben.

Attribute: Definiert das UML-Profil-Element neue Attribute, werden Name, Typ und Beschreibung des Attributs beschrieben.

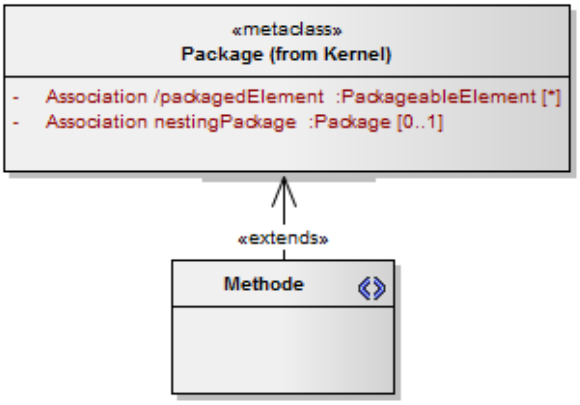
Konkrete Syntax: Dieser Abschnitt definiert die konkrete Syntax des UML-Profil-Elements.

Einschränkungen: Dieser Abschnitt benennt explizit die Einschränkungen der Metamodell-Elemente im UML-Profil und gibt diese zusätzlich in Form von OCL-Constraints an.

Beispiele: Der letzte Abschnitt gibt grafische Beispiele für die Benutzung des UML-Profil-Elements. Die konkrete Syntax wird dargestellt.

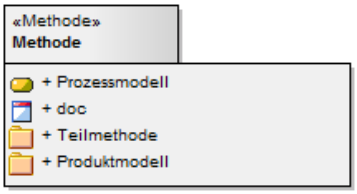
A.3.1 MMMB::Methode

Profil Definition:



Beschreibung und Semantik:	Eine «Methode» wird von der Metaklasse `Package` abgeleitet. Eine «Methode» beschreibt ein Methodenmodell, in dem der Namensraum für die konkreten Methodenelemente definiert wird. Das Element «Methode» erweitert die Metaklasse `Package`. Die Definitionen in Abschnitt 7.3.38 aus [Obj11c] gelten soweit wie in diesem Abschnitt keine Einschränkungen definiert werden.
Attribute:	keine
Konkrete Syntax:	Die konkrete Syntax des UML-Elements `Package` gilt (siehe 7.3.38 aus [Obj11c]). Zusätzlich zum Paketnamen soll die Zeichenfolge „«Methode»“ vorangestellt werden.
Einschränkungen:	Eine «Methode» darf nur Elemente vom Typ «Prozessmodell», «Produktmodell» und «Methode» enthalten. `context Methode inv: self.packagedElement -> forAll(i | i.oclIsTypeOf(Produktmodell) or i.oclIsTypeOf(Prozessmodell) or i.oclIsTypeOf(Methode)`

Beispiele:

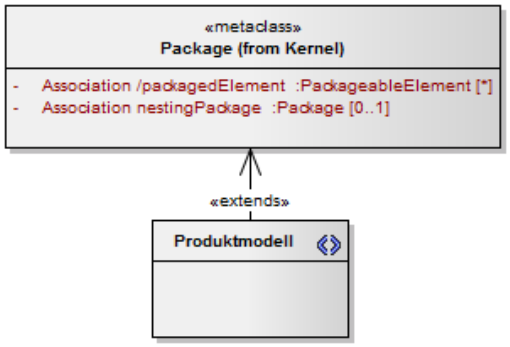


A.3.2 MMMB::Methodenverfeinerung

Profil Definition:	Diese Assoziation wird implizit über das Attribut «Methode».packagedElement definiert (siehe Abschnitt 10.4.1 [Obj11b])
Beschreibung und Semantik:	Pakete ermöglichen die Gruppierung weiterer Pakete. Die Assoziation «Methodenverfeinerung» definiert die Gruppierung einer (Teil-)«Methode» in der «Methode».
Attribute:	keine
Konkrete Syntax:	Die Gruppierung der (Teil-)«Methode» und «Methode» wird implizit über die konkrete Syntax der Pakete definiert (siehe «Methode»). Die Darstellung ist konform zur UML-Paket-Definition (siehe Abschnitt 7.3.38 aus [Obj11c]).
Einschränkungen:	Siehe Einschränkungen von «Methode».
Beispiele:	Siehe Beispiele von «Methode».

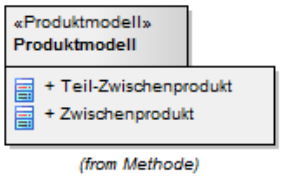
A.3.3 MMMB::Produktmodell

Profil Definition:



Beschreibung und Semantik:	Das Element «Produktmodell» wird von der Metaklasse `Package` abgeleitet. Ein «Produktmodell» beschreibt ein Produktmodell, in dem der Namensraum für die konkreten Artefakttypen definiert wird. Das «Produktmodell» enthält die Artefakttypen. Die Definitionen in Abschnitt 7.3.38 aus [Obj11c] gelten soweit in diesem Abschnitt keine Einschränkungen definiert werden.
Attribute:	keine
Konkrete Syntax:	Die konkrete Syntax des UML-Package-Elements gilt (siehe 7.3.38 aus [Obj11c]). Dem Paketnamen soll die Zeichenfolge «Produktmodell» vorangestellt werden.
Einschränkungen:	Das «Produktmodell» darf lediglich «Artefakttypen» enthalten. `context Produktmodell inv: self.packagedElement -> forAll(i | i.oclIsTypeOf(Artefakttyp))`

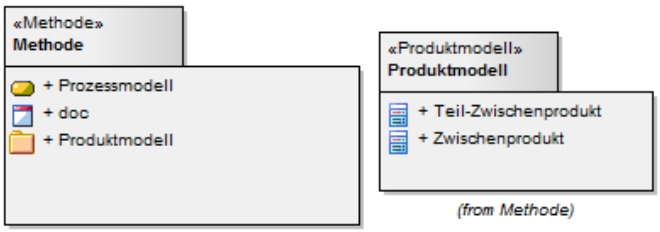
Beispiele:



A.3.4 MMMB::MethodenProduktmodell

Profil Definition:	Diese Assoziation wird implizit über das Attribut «Methode».packagedElement definiert (siehe Abschnitt 10.4.1 [Obj11b])
Beschreibung und Semantik:	Pakete ermöglichen die Gruppierung weiterer Pakete. Die Assoziation «MethodenProduktmodell» definiert die Gruppierung von «Produktmodell» in der «Methode».
Attribute:	keine
Konkrete Syntax:	Die Gruppierung der «Methode» und «Produktmodell» wird implizit über die konkrete Syntax der Pakete definiert (siehe «Methode» und «Produktmodell»). Die Darstellung ist konform zur UML-Paket-Definition (siehe Abschnitt 7.3.38 aus [Obj11c]).
Einschränkungen:	Siehe Einschränkungen von «Methode».

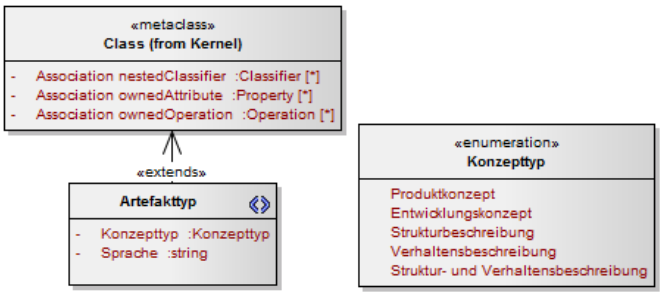
Beispiele:



Auf der linken Seite wird die Assoziation «MethodenProduktmodell» implizit durch das enthaltene «Produktmodell» Produktmodell innerhalb der «Methode» Methode dargestellt. Auf der rechten Seite wird der Namensraum vom «Produktmodell» Produktmodell unterhalb des Pakets angezeigt. Beide Beispiele repräsentieren den Zusammenhang der Pakete, die durch «MethodenProduktmodell» definiert werden.

A.3.5 MMMB::Produktmodell::Artefakttyp

Profil Definition:



Beschreibung und Semantik:

Ein «Artefakttyp» beschreibt ein Artefakttyp innerhalb des «Produktmodells». Ein «Artefakttyp» beschreibt den Typ eines Zwischen- oder End-Produkts der «Methode». Der «Artefakttyp» ist weiter typisierbar. Zwei Zustände sind angedacht. Wird dem «Artefakttyp» eine Sprache zugewiesen, ist die Definition als vollständiger Artefakttyp abgeschlossen. Fehlt die Zuweisung einer Sprache, handelt es sich lediglich um ein Konzept als Platzhalter für eine vollständige Beschreibung des Artefakttyps. Das ist in früheren Entwicklungsphasen der Fall. Die Definitionen in Abschnitt 7.3.7 aus [Obj11c] gelten soweit in diesem Abschnitt keine Einschränkungen definiert werden. Die Eigenschaften der UML-Klasse bezüglich der Modellierung von Attributen werden explizit beibehalten. Jedoch ist die Spezifikation der Sichtbarkeit der Attribute zu vernachlässigen. Die Definition von Operationen beim «Artefakttyp» wird nicht unterstützt. Die Zustände für ein «Artefakttyp» können durch ein «Lebenszyklusmodell» definiert werden.

Attribute:

`Konzepttyp : Konzepttyp`

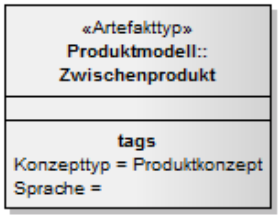
Der Konzepttyp kann die folgenden Werte enthalten: „Produktkonzept“ oder „Entwicklungskonzept“. Das Produktkonzept kann weiter unterteilt werden in „Strukturbeschreibung“, „Verhaltensbeschreibung“ oder „Struktur- und Verhaltensbeschreibung“.

`Sprache : String`

Dem «Artefakttyp» kann über das Attribut Sprache eine Sprache zugeordnet werden. Die Sprache wird über eine frei wählbare Zeichenfolge definiert. Die Identifizierung einer konkreten Sprache muss vom Methodenentwickler erfolgen.

Konkrete Syntax:

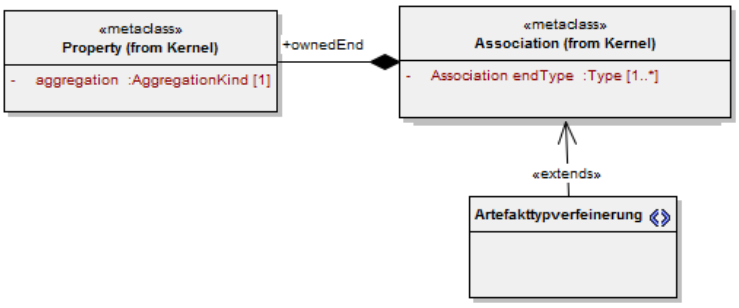
Die konkrete Syntax richtet sich nach der Definition aus Abschnitt 7.3.7 in [Obj11c]. Jedoch wird die konkrete Syntax dahingehend eingeschränkt, dass ein «Artefakttyp» keine Operationen darstellen darf. Dem Namen soll die Zeichenfolge «Artefakttyp» vorangestellt werden.

Einschränkungen:	Die Definition von Operationen ist beim «Artefakttyp» unterbunden. `context Artefakttyp inv: self.ownedOperation->isEmpty()`
Beispiele:	

A.3.6 MMMB::Produktmodell::ProduktmodellArtefakt

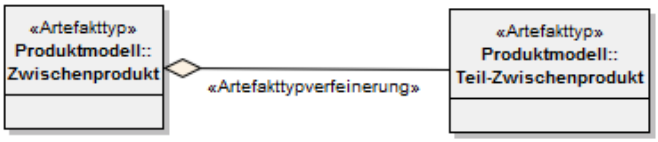
Profil Definition:	Die Enthalten-Beziehung zwischen «Artefakttyp» und «Produktmodell» wird implizit über das «Produktmodell».packagedElement Attribut definiert. In der Vererbungshierarchie ist ein «Artefakttyp» vom Typ PackageableElement.
Beschreibung und Semantik:	Die Assoziation «ProduktmodellArtefakttyp» definiert die Gruppierung von «Artefakttyp»-Elementen in einem «Produktmodell».
Attribute:	keine
Konkrete Syntax:	Die Gruppierung von Elementen des Typs «Artefakttyp» im «Produktmodell» wird implizit über die konkrete Syntax der Pakete definiert (siehe «Produktmodell»). Die Darstellung ist konform zur UML-Paket-Definition (siehe Abschnitt 7.3.38 aus [Obj11c]).
Einschränkungen:	keine
Beispiele:	Siehe Beispiele für «Produktmodell».

A.3.7 MMMB::Produktmodell::Artefakttypverfeinerung

Profil Definition:	
	Das Stereotyp «Artefakttypverfeinerung» wird von der Meta-Klasse `Association` abgeleitet. Die Aggregationsbeziehung ist möglich, indem die `Association` das UML-Element `Property` über das `ownedEnd`-Attribut referenziert. Ein `Property` wird ebenfalls von den assoziierenden Klassen (UML-Element: Class) referenziert. Das aggregierende `Property` nutzt das Attribut `aggregation` vom Typ `aggregationKind`.
Beschreibung und Semantik:	Es besteht die Möglichkeit ein «Artefakttyp» hierarchisch, durch weitere «Artefakttyp»-Elemente, zu verfeinern. Durch verschiedene «Artefakttyp»-Elemente kann also ein komplexer «Artefakttyp» entstehen. Die Verfeinerung geschieht, indem verschiedene «Artefakttyp»-Elemente durch die Assoziation «Artefakttypverfeinerung» miteinander in Beziehung gesetzt werden. Hierbei gilt die Konstruktionsregel Teil-Ganzes. Verschiedene Teile erweitern das Ganze. Die «Artefakttypverfeinerung» definiert mit einem Assoziationsende das Teil, mit dem anderen Ende das Ganze. Diese Eigenschaft wird durch die Typisierung der UML-Assoziation als Aggregation erreicht. Weiterhin gilt die Semantikdefinition aus Abschnitt 11.3.1 in [Obj11b] und Abschnitt 7.3.3 in [Obj11c] mit den in dieser Tabelle definierten Einschränkungen.
Attribute:	keine
Konkrete Syntax:	Siehe konkrete Syntax für UML-Assoziationen, die als Aggregation definiert sind (vgl. 7.3.3 in [Obj11c]). Zusätzlich soll dem Assoziationsnamen der Präfix «Artefakttypverfeinerung» hinzugefügt werden.

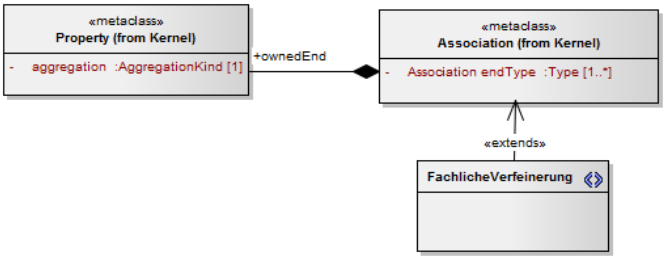
Einschränkungen:	Assoziationen vom Typ «Artefakttypverfeinerung» sind Aggregationen.
	`context Artefakttypverfeinerung inv: self.ownedEnd -> Exist(p: Property.aggregationKind = shared)`
	Assoziationen vom Typ «Artefakttypverfeinerung» sind nur zwischen «Artefakttyp»-Elementen zulässig.
	`context Artefakttypverfeinerung inv: self.endType->forAllt | t.oclIsTypeOf(Artefakttyp)`

Beispiele:



A.3.8 MMMB::Produktmodell::FachlicheVerfeinerung

Profil Definition:



Beschreibung und Semantik:	Das Stereotyp «FachlicheVerfeinerung» wird von der Metaklasse `Association` abgeleitet. Während die «Artefakttypverfeinerung» bereits die strukturelle Verfeinerung eines «Artefakttyp»-Elements beschreibt, werden gerichtete Assoziationen verwendet. Das UML-Element `Property`, welches über das `ownedEnd` Attribut referenziert wird, wird ebenfalls von den assoziierenden Klassen (UML-Element: Class) referenziert. Die Funktion `isNavigable` des Property muss `TRUE` ergeben. Es besteht die Möglichkeit ein «Artefakttyp» fachlich, durch weitere «Artefakttyp»-Elemente, zu verfeinern. Das bedeutet, der Grad an Abstraktion wird verringert. Das fachlich detaillierte Element referenziert das fachlich abstraktere Element. Diese Eigenschaft wird durch die Typisierung als UML-Element `Association` als navigierbare Assoziation erreicht. Weiterhin gilt die Semantikdefinition aus Abschnitt 11.3.1 in [Obj11b] und Abschnitt 7.3.3 in [Obj11c] mit den definierten Einschränkungen.
Attribute:	keine
Konkrete Syntax:	Siehe konkrete Syntax für `Assoziationen`, die als navigierbare Assoziation definiert sind (vgl. 7.3.3 in [Obj11c]). Zusätzlich soll dem Assoziationsnamen der Präfix „«FachlicheVerfeinerung»“ hinzugefügt werden.

Einschränkungen:

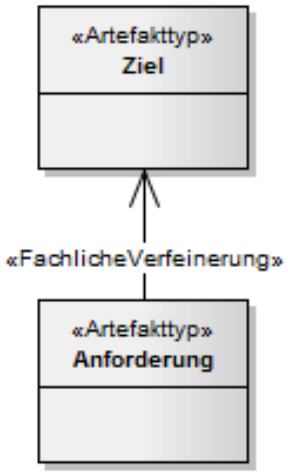
Assoziationen vom Typ «FachlicheVerfeinerung» sind navigierbare Assoziationen, wobei das fachlich abstraktere Element referenziert wird.

```
context FachlicheVerfeinerung inv: self.ownedEnd -
> Exist(p | p.Property.isNavigable() = FALSE) and
self.memberEnd -> Exist (q | q.Property.isNavigable()
= TRUE)
```

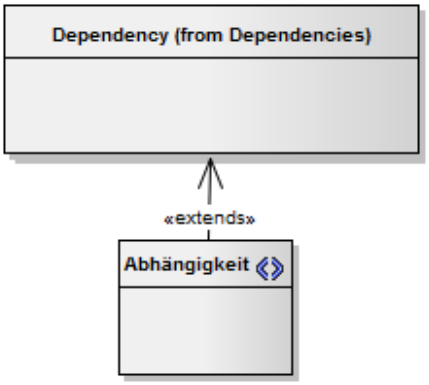
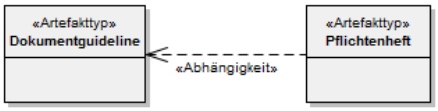
Assoziationen vom Typ «FachlicheVerfeinerung» sind nur zwischen «Artefakttyp»-Elementen zulässig.

```
context FachlicheVerfeinerung inv: self.endType->forAllt
| t.oclIsTypeOf(Artefakttyp)
```

Beispiele:

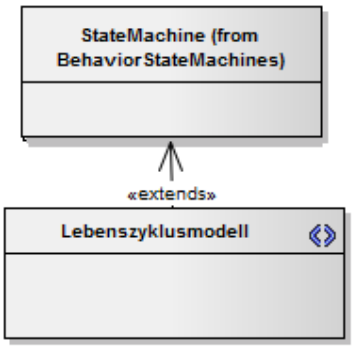


A.3.9 MMMB::Produktmodell::Abhängigkeit

Profil Definition:	
	Das Stereotyp «Abhängigkeit» wird von der Metaklasse `Dependency` abgeleitet.
Beschreibung und Semantik:	Es besteht die Möglichkeit zwei «Artefakttyp»-Elemente miteinander in Beziehung zu setzen. Das bedeutet, ein Klient-Element ist von einem Bediener-Element in irgendeiner direkten oder indirekten Art und Weise abhängig. Diese Eigenschaft wird durch Typisierung als UML-Element `Dependency` erreicht. Es gilt die Semantikdefinition aus Abschnitt 7.3.12 in [Obj11c] mit den definierten Einschränkungen.
Attribute:	keine
Konkrete Syntax:	Siehe konkrete Syntax für UML-Dependency (vgl. 7.3.12 in [Obj11c]). Zusätzlich soll dem Namen der Assoziation der Präfix „«Abhängigkeit»“ hinzugefügt werden.
Einschränkungen:	Assoziationen vom Typ «Abhängigkeit» dürfen nur zwischen «Artefakttyp»-Elementen definiert werden. `context Abhängigkeit inv: self.supplier -> Exist(s | s.oclIsTypeOf(Artefakttyp)) and self.client -> Exist(c| c.oclIsTypeOf(Artefakttyp))`
Beispiele:	

A.3.10 MMMB::Lebenszyklusmodell

Profil Definition:



Beschreibung und Semantik: Das «Lebenszyklusmodell» wird von dem UML-Element `StateMachine` abgeleitet. Ein «Lebenszyklusmodell» ermöglicht die Definition verschiedener Zustände eines Elements vom Typ «Artefakttyp».

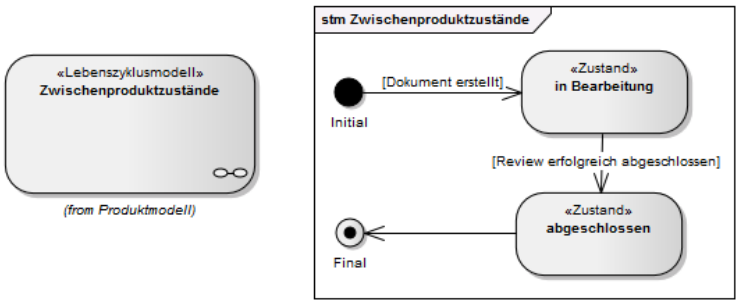
Attribute: keine

Konkrete Syntax: Die konkrete Syntax eines «Lebenszyklusmodell»-Elements entspricht der konkreten Syntax des UML-Elements `StateMachine` (siehe Abschnitt 15.3.12 in [Obj11c]). Dem Namen der Zustandsmaschine soll der Präfix „«Lebenszyklusmodell»“ hinzugefügt werden. Die Zustandsmaschine darf durch UML-Zustandsdiagramme weiter verfeinert werden.

Einschränkungen: Ein «Lebenszyklusmodell» hat den Kontext «Artefakttyp».

```
context Lebenszyklusmodell inv: self -> exist(m : Artefakttyp | m.nestedClassifier = self)
```

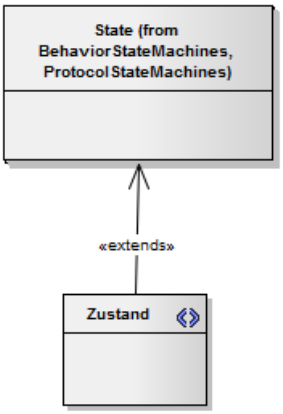
Beispiele:



Auf der linken Seite der Abbildung wird das Symbol für ein «Lebenszyklusmodell» angegeben. Auf der rechten Seite der Abbildung wird ein Beispiel für eine Zustandsmaschine in Form eines UML-Zustandsdiagramms angegeben. Beide Beispiele definieren ein «Lebenszyklusmodell» für das „«Artefakttyp» Zwischenprodukt“.

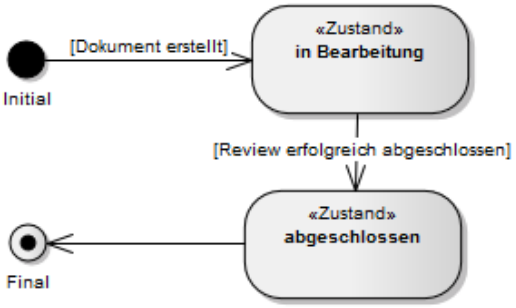
A.3.11 MMMB::Lebenszyklusmodell::Zustand

Profil Definition:



Beschreibung und Semantik:	Der «Zustand» wird von der Metaklasse `State` abgeleitet. Ein «Zustand» beschreibt einen möglichen Zustand für ein «Artefakttyp». Die Zustände werden im «Lebenszyklusmodell» beschrieben.
Attribute:	keine
Konkrete Syntax:	Die konkrete Syntax eines «Zustand»-Elements entspricht der konkreten Syntax des UML-Elements `State` (siehe Abschnitt 15.3.11 in [Obj11c]). Dem Namen des «Zustand»-Elements soll der Präfix „«Zustand»“ hinzugefügt werden.
Einschränkungen:	Ein «Zustand» wird innerhalb eines «Lebenszyklusmodell»-Elements definiert. `context Lebenszyklusmodell inv: self.region -> exist(r | r.vertex -> forAll (v | v.oclIsTypeOf(Zustand))`

Beispiele:



A.3.12 MMMB::Lebenszyklusmodell::ArtefaktZustand

Profil Definition:	Die Assoziation «ArtefaktZustand» wird implizit über die Assoziationskette «Lebenszyklusmodell».region.state beschrieben.

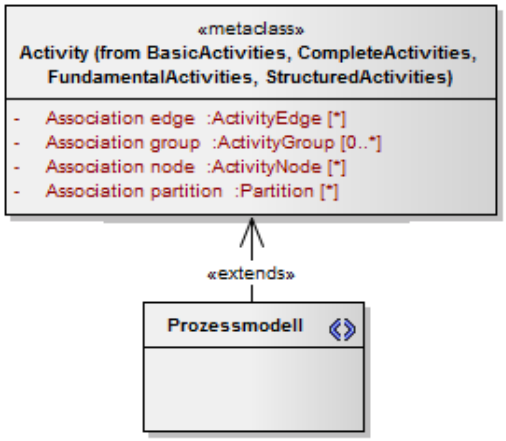
Beschreibung und Semantik:	Die implizite Assoziation «ArtefaktZustand» definiert den Zusammenhang zwischen einem «Lebenszyklusmodell» und einem «Zustand».
Attribute:	keine
Konkrete Syntax:	keine
Einschränkungen:	Siehe Einschränkungen vom «Zustand».
Beispiele:	Siehe Beispiele von «Zustand».

A.3.13 MMMB::Artefaktlebenszyklus

Profil Definition:	Der «Artefakttyplebenszyklus» wird implizit über das Attribut «Artefakttyp».nestedClassifier beschrieben.
Beschreibung und Semantik:	Die implizite Assoziation «Artefakttyplebenszyklus» definiert den Zusammenhang zwischen einem «Lebenszyklusmodell» und einem «Artefakttyp».
Attribute:	keine
Konkrete Syntax:	Aufgrund der impliziten Assoziation gibt es keine graphische Repräsentation der Assoziation «Artefakttyplebenszyklusmodell».
Einschränkungen:	Es darf maximal ein «Lebenszyklusmodell» für ein «Artefakttyp» spezifiziert werden. `context Artefakttyp inv: self.nestedQualifier->size()<=1 and self.nestedQualifier->forAll(i | i.oclIsTypeOf(Lebenszyklusmodell))`
Beispiele:	keine

A.3.14 MMMB::Prozessmodell

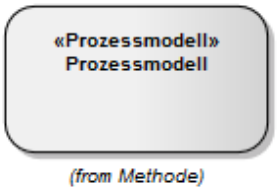
Profil Definition:



Beschreibung und Semantik:	Das «Prozessmodell» wird von der Metaklasse `Activity` abgeleitet. Die konkreten Elemente vom Typ «Aktivität» werden vom UML-Element `ActivityNode` definiert. Deren Ausführungsreihenfolgen werden im «Prozessmodell» festgelegt. Die Definitionen in Abschnitt 12.3.4 aus [Obj11c] gelten soweit in diesem Abschnitt keine Einschränkungen definiert werden.
Attribute:	keine
Konkrete Syntax:	Die Notation der UML-Aktivität gilt (siehe 12.3.4 aus [Obj11c]). Dem Namen soll die Zeichenfolge „«Prozessmodell»“ vorangestellt werden.
Einschränkungen:	Das «Prozessmodell» enthält lediglich Elemente vom Typ «Aktivität», «ZusammengesetzteAktivität», «Artefakt», «Hilfsmittel», «Rolle», «Meilenstein», «Phase», «Eingabe», «Ausgabe», «Ergebnis», «AktivitätHilfsmittel», «Teilnehmer» und «Durchführer».

```
context Prozessmodell inv: self.node -> forAll(i |
i.oclIsTypeOf(Aktivität) or i.oclIsTypeOf(Prozess)) or
i.oclIsTypeOf(Artefakt)) or i.oclIsTypeOf(Hilfsmittel))
or i.oclIsTypeOf(Rolle)) or i.oclIsTypeOf(Meilenstein))
context Prozessmodell inv: self.partition -> forAll(p |
p.oclIsTypeOf(Phase) )
context Prozessmodell inv: self.edge -
> forAll(e | e.oclIsTypeOf(Eingabe) or
e.oclIsTypeOf(Ausgabe)) or e.oclIsTypeOf(Ergebnis))
or e.oclIsTypeOf(AktivitätHilfsmittel)) or
e.oclIsTypeOf(Teilnehmer)) or e.oclIsTypeOf(Durchführer))
```

Beispiele:



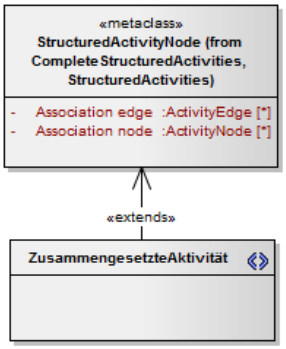
A.3.15 MMMB::MethodenProzessmodell

Profil Definition:	Diese Assoziation wird implizit über die Attribute «Methode».packagedElement definiert.
Beschreibung und Semantik:	Die Assoziation «MethodenProzessmodell» definiert die Gruppierung von «Prozessmodell» und «Methode».
Attribute:	keine
Konkrete Syntax:	Aufgrund der impliziten Definition existiert für diese Assoziation keine grafische Repräsentation.
Einschränkungen:	Siehe Einschränkungen von «Methode».

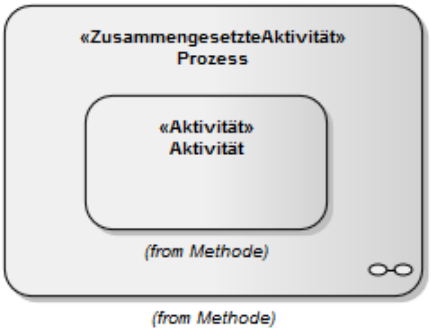
Beispiele:

Auf der linken Seite wird die Assoziation «MethodenProzessmodell» implizit durch das enthaltene «Prozessmodell» Prozessmodell innerhalb der «Methode» dargestellt. Auf der rechten Seite wird der Namensraum vom «Prozessmodell» Prozessmodell unterhalb des Pakets angezeigt. Beide Beispiele repräsentieren den Zusammenhang der Pakete, die durch «MethodenProzessmodell» definiert werden.

A.3.16 MMMB::Prozessmodell::ZusammengesetzteAktivität

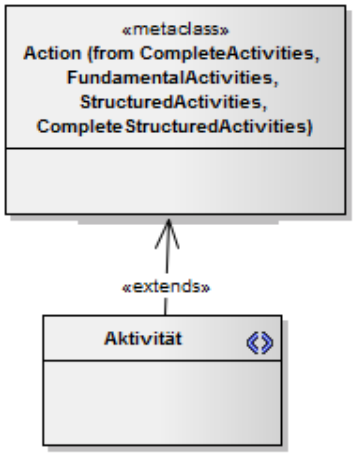
Profil Definition:	
Beschreibung und Semantik:	Die «ZusammengesetzteAktivität» wird von der Metaklasse `StructuredActivityNode` abgeleitet. Eine «ZusammengesetzteAktivität» ist vom Typ «Aktivität». Die «ZusammengesetzteAktivität» kann als Top-Level-Element im «Prozessmodell» definiert werden. Weiterhin können jedoch weitere «ZusammengesetzteAktivität»-Elemente und «Aktivitäten» enthalten sein. Die «ZusammengesetzteAktivität» dient zur Strukturierung von weiteren Elementen vom Typ«Aktivität». Die Semantik des UML-Elements `StructuredActivityNode` (Abschnitt 12.3.48 aus [Obj11c]) wird übernommen.
Attribute:	keine
Konkrete Syntax:	Die konkrete Syntax wird von der Definition des UML-Elements `StructuredActivityNode` aus Abschnitt 12.3.48 aus [Obj11c] übernommen. Dem Namen soll die Zeichenfolge „«ZusammengesetzteAktivität»“ vorangestellt werden.
Einschränkungen:	Eine «ZusammengesetzteAktivität» darf lediglich Elemente vom Typ «ZusammengesetzteAktivität», «Aktivität», «Eingabe» und «Ausgabe» enthalten. `context ZusammengesetzteAktivität inv: self.node -> forAll n|n.oclIsTypeOf(Aktivität) or n.oclIsTypeOf(ZusammengesetzteAktivität) context ZusammengesetzteAktivität inv: self.edge -> forAll e|e.oclIsTypeOf(Eingabe) or e.oclIsTypeOf(Ausgabe)`

Beispiele:



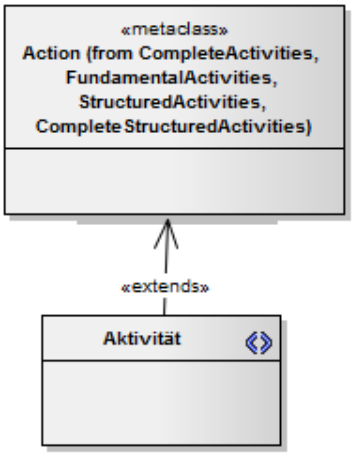
A.3.17 MMMB::Prozessmodell::Aktivität

Profil Definition:



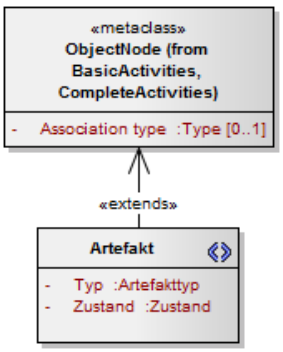
Beschreibung und Semantik:	Die «Aktivität» wird von der Metaklasse `Action` abgeleitet. Das Element «Aktivität» beschreibt eine nicht weiter verfeinerte Aktivität. Die Semantik des UML-Elements „Action“ wird übernommen (Abschnitt 12.3.2 aus [Obj11c]).
Attribute:	keine
Konkrete Syntax:	Die konkrete Syntax des UML-Elements `Action` gilt nach Abschnitt 12.3.2 aus [Obj11c]. Dem Namen soll die Zeichenfolge „«Aktivität»“ vorangestellt werden.
Einschränkungen:	keine

Beispiele:



A.3.18 MMMB::Prozessmodell::Artefakt

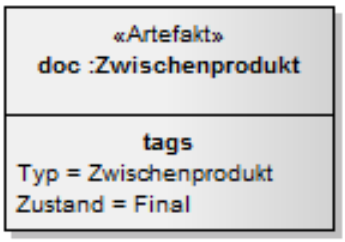
Profil Definition:



Beschreibung und Semantik:

Das «Artefakt» wird von der Metaklasse `ObjectNode` abgeleitet. Ein «Artefakt» repräsentiert einen konkreten Artefakttyp im «Prozessmodell». Diese Repräsentation wird über die Typ-Eigenschaft des Attributs Typ des Elements «Artefakt» definiert. Demnach muss ein «Artefakt» immer über einen «Artefakttyp» getypt sein. Im «Prozessmodell» können «Artefakt»-Elemente als Eingabe- oder Ausgabe-Artefakte einer «Aktivität» oder eines «ZusammengesetzteAktivität»-Elements definiert werden. Dadurch wird es möglich einen Objektfluss im «Prozessmodell» zu definieren. Die Semantik des UML-Elements `ObjectNode` gilt (siehe Abschnitt 12.3.38 aus [Obj11c]).

Attribute:	`Typ : Artefakttyp`
	Der Typ referenziert auf einen «Artefakttyp» aus einem «Produktmodell».
	`Zustand : Zustand`
	Dem «Artefakt» kann über das Attribut `Zustand` ein möglicher Zustand aus dem «Lebenszyklusmodell» des zugrunde liegenden «Artefakttyp»-Elements zugewiesen werden.

Konkrete Syntax:	Die konkrete Syntax wird definiert über den UML-Objektknoten (siehe Abschnitt 12.3.38 in [Obj11c]). Dem Namen soll die Zeichenfolge „«Artefakt»“ vorangestellt werden.
Einschränkungen:	Ein Element vom Typ «Artefakt» hat immer einen definierten Typ, dessen Typ einem «Artefakttyp» entspricht. Hat ein Element vom Typ «Artefakt» einen Zustand, muss dieser Zustand in dem «Lebenszyklusmodell» des «Artefakttyp»-Elements, über den das «Artefakt»-Element getypt ist, definiert sein. `context Artefakt inv: self.type -> notEmpty() and self.type.type == oclIsTypeOfArtefakttyp context Artefakt inv: self.Zustand = z and self.type.type.nestedQualifier->exist (l | l.oclIsTypeOf (Lebenszyklusmodell)`
Beispiele:	 *(from Methode)*

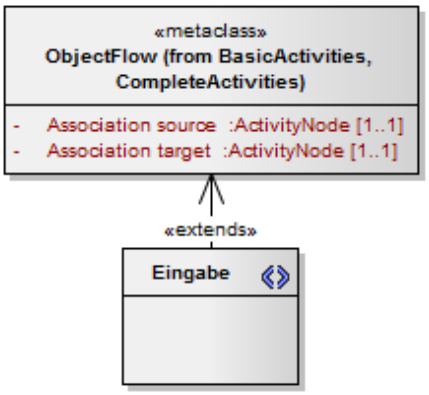
A.3.19 MMMB::Prozessmodell::Typ

Profil Definition:	Der «Typ» wird implizit über das Attribut «Artefakt».type (UML-Attribut) definiert. Das Attribut «Artefakt».Typ (MMMB-Attribut) wird zusätzlich angegeben, in diesem UML-Profil aber nur aus deskriptiven Gründen benutzt.
Beschreibung und Semantik:	«Typ» definiert die Typisierung eines «Artefakt»-Elements über ein «Artefakttyp».
Attribute:	keine
Konkrete Syntax:	Aufgrund der impliziten Definition gibt es keine grafische Darstellung.
Einschränkungen:	Siehe Einschränkungen von «Artefakt».

Beispiele: keine

A.3.20 MMMB::Prozessmodell::Eingabe

Profil Definition:



Beschreibung und Semantik: Die Assoziation «Eingabe» wird von der Metaklasse `ObjectFlow` abgeleitet. Die Assoziation «Eingabe» definiert ein «Artefakt» als Eingabeobjekt für eine «Aktivität» oder «Zusammengesetzte-Aktivität». Die Semantik ist an die Objektflussmodellierung der UML angelehnt. Die Semantik der UML-Objektfluss-Assoziation, wie in Abschnitt 12.3.47 aus [Obj11c], soll beibehalten werden.

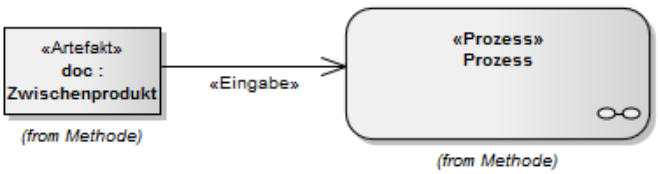
Attribute: keine

Konkrete Syntax: Die konkrete Syntax ist angelehnt an das UML-Element `Objekt-fluss` (siehe Abschnitt 12.3.47 aus [Obj11c]). Dem Namen soll die Zeichenfolge «Eingabe» vorangestellt werden.

Einschränkungen: Die «Eingabe» definiert ein «Artefakt»-Element als Eingabe für eine «Aktivität». Die Quelle der Assoziation soll das «Artefakt» und die Senke die «Aktivität» referenzieren.

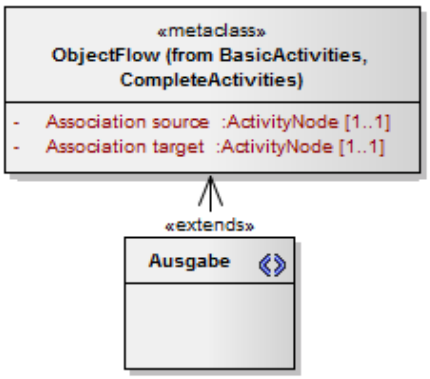
```
context Eingabe inv: self.source -> oclIsTypeOfArtefakt
and self.target -> oclIsTypeOfAktivität
```

Beispiele:



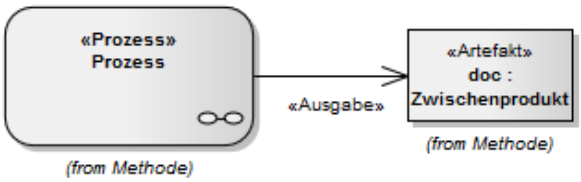
A.3.21 MMMB::Prozessmodell::Ausgabe

Profil Definition:

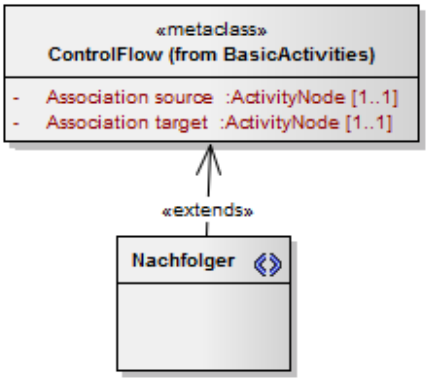
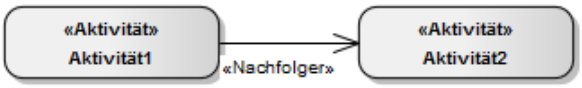


Beschreibung und Semantik:	Die Assoziation «Ausgabe» wird von der Metaklasse `ObjectFlow` abgeleitet. Die Assoziation «Ausgabe» definiert ein Element vom Typ «Artefakt» als Ausgabe-Objekt für eine «Aktivität». Die Semantik ist an die Objektflussmodellierung der UML angelehnt. Die Semantik des UML-Elements „ObjectNode“, wie in Abschnitt 12.3.37 aus [Obj11c], soll beibehalten werden.
Attribute:	keine
Konkrete Syntax:	Die konkrete Syntax ist angelehnt an das UML-Element `ObjectFlow` wie in Abschnitt 12.3.37 aus [Obj11c]. Dem Namen soll die Zeichenfolge „«Ausgabe»“ vorangestellt werden.
Einschränkungen:	Die «Ausgabe» definiert ein «Artefakt» als Ausgabe für eine «Aktivität». Die Quelle der Assoziation soll die «Aktivität» und die Senke das «Artefakt» referenzieren. `context Ausgabe inv: self.source -> oclIsTypeOfAktivität and self.target -> oclIsTypeOfArtefakt`

Beispiele:



A.3.22 MMMB::Prozessmodell::Nachfolger

Profil Definition:	
Beschreibung und Semantik:	Die Assoziation «Nachfolger» wird von der Metaklasse `ControlFlow` abgeleitet. Die Assoziation «Nachfolger» definiert einen Kontrollfluss zwischen zwei «Aktivität»-Elementen. Die Semantik des UML-Elements `Control Flow`, wie in Abschnitt 12.3.19 aus [Obj11c], soll beibehalten werden.
Attribute:	keine
Konkrete Syntax:	Die konkrete Syntax ist angelehnt an das UML-Element `Control Flow` wie in Abschnitt 12.3.19 aus [Obj11c]. Dem Namen soll die Zeichenfolge „«Nachfolger»“ vorangestellt werden.
Einschränkungen:	«Nachfolger» definiert eine «Aktivität» als Nachfolger für eine andere «Aktivität». Die Quelle der Assoziation muss ein «Aktivität»- oder «ZusammengesetzteAktivität»-Element und die Senke ein anderes «Artefakt»- oder «Prozess»-Element referenzieren. `context Nachfolger inv: self.source -> oclIsTypeOfAktivität or oclIsTypeOfZusammengesetzteAktivität and self.target -> oclIsTypeOfAktivität or oclIsTypeOfZusammengesetzteAktivität and self.source <> self.target`
Beispiele:	

A.3.23 MMMB::Prozessmodell::ProzessmodellArtefakt

Profil Definition:	Diese Assoziation wird implizit über das Attribut «Prozessmodell».node definiert. Das Attribut enthält alle Elemente vom Typ ActivityNode, nach Definition auch «Artefakt»-Elemente.
Beschreibung und Semantik:	keine

Attribute:	keine
Konkrete Syntax:	Durch die implizite Definition gibt es keine grafische Repräsentation.
Einschränkungen:	keine
Beispiele:	keine

A.3.24 MMMB::Prozessmodell::ProzessmodellAktivität

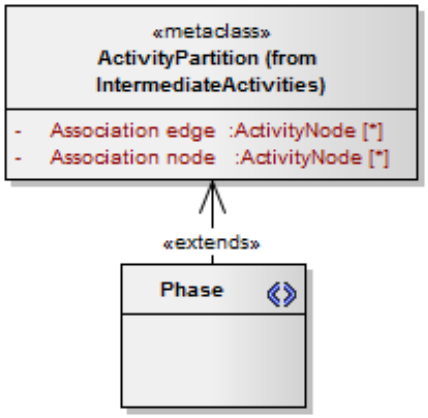
Profil Definition:	Diese Assoziation wird implizit über das Attribut «Prozessmodell».node definiert. Das Attribut enthält alle Elemente vom Typ ActivityNode, nach Definition auch «Aktivität»- und «ZusammengesetzteAktivität»-Elemente.
Beschreibung und Semantik:	Die Assoziation «ProzessmodellAktitvität» beschreibt die Aggregation der Top-Level-Elemente «Aktivität» und «ZusammengesetzteAktivität» im «Prozessmodell».
Attribute:	keine
Konkrete Syntax:	Durch die implizite Definition gibt es keine grafische Repräsentation.
Einschränkungen:	keine
Beispiele:	keine

A.3.25 MMMB::Prozessmodell::ProzessAktivität

Profil Definition:	Diese Assoziation wird implizit durch «ZusammengesetzteAktivität» modelliert, welche wiederum Elemente vom Typ «Aktivität» enthalten kann. Das Attribut «ZusammengesetzteAktivität».node referenziert Elemente vom Typ ActivityNode. Das sind nach Definition auch Elemente vom Typ «Aktivität».
Beschreibung und Semantik:	Die Assoziation «ProzessAktivität» verbindet «ZusammengesetzteAktivität» und konkrete Elemente vom Typ «Aktivität» miteinander.
Attribute:	keine
Konkrete Syntax:	keine
Einschränkungen:	keine
Beispiele:	keine

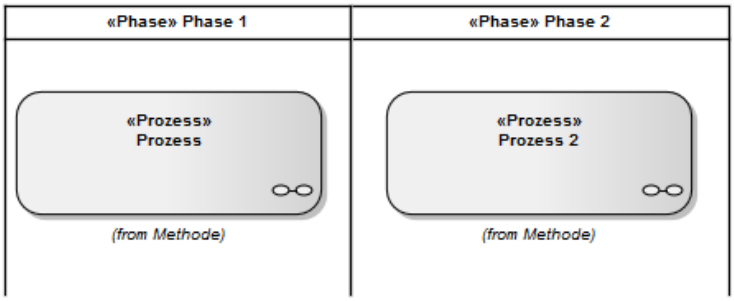
A.3.26 MMMB::Prozessmodell::Phase

Profil Definition:



Beschreibung und Semantik:	Die «Phase» wird von der Metaklasse `ActivityPartition` abgeleitet. Eine «Phase» ist ein reines Strukturelement. Es unterteilt das Prozessmodell in Phasen. Beispielsweise die Darstellung von RUP-Disziplinen, V-Modell-Phasen, SCRUM-Iterationen oder weitere Organisations-spezifische Entwicklungsphasen. Elemente vom Typ «Aktivität», «ZusammengesetzteAktivität» und «Artefakt» können einer «Phase» zugeordnet werden.
Attribute:	keine
Konkrete Syntax:	Die «Phase» übernimmt die konkrete Syntax des UML-Elements `ActivityPartition` (siehe Abschnitt 12.3.10 aus [Obj11c]). Dem Namen soll der Präfix „«Phase»“ vorangestellt werden.
Einschränkungen:	Eine «Phase» kann nur innerhalb vom «Prozessmodell» definiert werden. Siehe Einschränkung «Prozessmodell». Eine «Phase» darf lediglich Elemente vom Typ «Aktivität», «ZusammengesetzteAktivität», «Artefakt», «Eingabe» und «Ausgabe» enthalten. `context Phase inv: self.node -> forAll n|n.oclIsTypeOf(Aktivität)or n.oclIsTypeOf(ZusammengesetzteAktivität) or n.oclIsTypeOf(Artefakt) context Phase inv: self.edge -> forAll e|e.oclIsTypeOf(Eingabe) or e.oclIsTypeOf(Ausgabe)`

Beispiele:



A.3.27 MMMB::Prozessmodell::ProzessmodellPhase

Profil Definition:	Diese Assoziation wird implizit über das Attribut «Prozessmodell».group definiert. Das Attribut enthält alle Elemente vom Typ ActivityGroup und deren Untertypen ActivityPartition, nach Definition auch Elemente vom Typ «Phase».
Beschreibung und Semantik:	Die Assoziation «ProzessmodellPhase» beschreibt die Aggregation der Elemente vom Typ «Phase» im «Prozessmodell».
Attribute:	keine
Konkrete Syntax:	Aufgrund der impliziten Definition gibt es keine grafische Repräsentation.
Einschränkungen:	keine
Beispiele:	keine

A.3.28 MMMB::Prozessmodell::PhasenAktivität

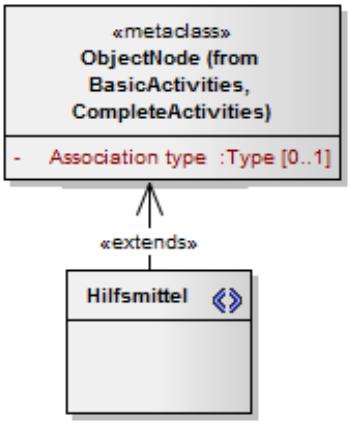
Profil Definition:	Diese Assoziation wird implizit über das Attribut «Phase».node definiert. Das Attribut enthält alle Elemente vom Typ ActivityNode, die der «Phase» zugewiesen werden, und deren Untertypen ActivityNode, nach Definition auch Elemente vom Typ «Aktivität».
Beschreibung und Semantik:	Die Assoziation PhaseAktivität aggregiert eine Menge von Elementen des Typs «Aktivität», die der «Phase» zugewiesen werden.
Attribute:	keine
Konkrete Syntax:	keine
Einschränkungen:	Siehe Einschränkungen von «Phase».
Beispiele:	Siehe Beispiele von «Phase».

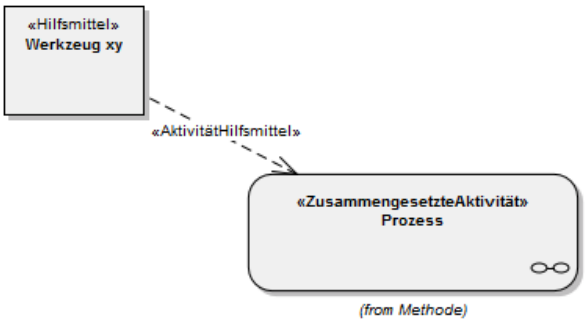
A.3.29 MMMB::Prozessmodell::PhasenArtefakt

Profil Definition:	Diese Assoziation wird implizit über das Attribut «Phase».node definiert. Das Attribut enthält alle Elemente vom Typ ActivityNode, die der «Phase» zugewiesen werden, und deren Untertypen ActivityNode, nach Definition auch «Artefakt»-Elemente.
Beschreibung und Semantik:	Die Assoziation «PhaseArtefakt» aggregiert eine Menge von «Artefakt»-Elementen, die der «Phase» zugewiesen werden.
Attribute:	keine
Konkrete Syntax:	keine
Einschränkungen:	Siehe Einschränkungen von «Phase».
Beispiele:	Siehe Beispiele von «Phase».

A.3.30 MMMB::Prozessmodell::Hilfsmittel

Profil Definition:



Beschreibung und Semantik:	Das «Hilfsmittel» wird von der Metaklasse `ObjectNode` abgeleitet. «Hilfsmittel» definieren informell einer «Aktivität» zugewiesene Hilfsmittel. Das «Hilfsmittel» wird über einen frei wählbaren String definiert. Die Identifizierung eines konkreten Hilfsmittels über den String muss vom Methodenentwickler erfolgen. Die Annotation eines Elements vom Typ «Hilfsmittel» an einer «Aktivität» überträgt keine semantischen Informationen, lediglich den Hinweis der Existenz eines Hilfsmittels. Wird das «Hilfsmittel» konkret von der «Aktivität» genutzt, wie beispielsweise eine Dokumentvorlage, muss das «Hilfsmittel» konkret als «Artefakttyp» definiert werden und einer «Aktivität» in Form eines «Artefakt»-Elements als Eingabeartefakt übergeben werden. Die Entscheidung «Hilfsmittel» oder «Artefakttyp» obliegt dem Methodenentwickler. Die Semantik des UML-Elements „ObjectNode“, wie in Abschnitt 12.3.37 aus [Obj11c], soll beibehalten
Attribute:	keine
Konkrete Syntax:	Die konkrete Syntax ist angelehnt an das UML-Element `ObjectNode` wie in Abschnitt 12.3.37 aus [Obj11c]. Das «Hilfsmittel» soll die Zeichenfolge „«Hilfsmittel»“ enthalten.
Einschränkungen:	Siehe Einschränkungen «AkivitätHilfsmittel».
Beispiele:	

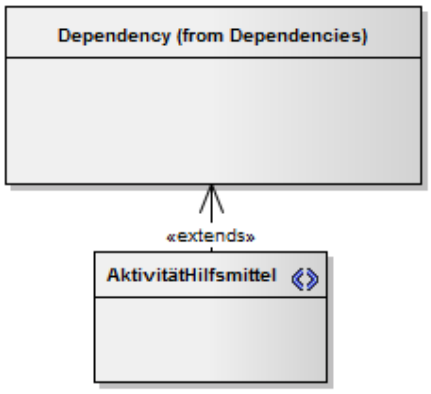
A.3.31 MMMB::Prozessmodell::ProzessmodellHilfsmittel

Profil Definition:	Diese Assoziation wird implizit über das Attribut «Prozessmodell».node definiert. Das Attribut enthält alle Elemente vom Typ ActivityNode, nach Definition auch «Hilfsmittel»-Elemente.
Beschreibung und Semantik:	Die Assoziation «ProzessmodellHilfsmittel» beschreibt die Aggregation der Top-Level-Elemente «Hilfsmittel» im «Prozessmodell».
Attribute:	keine

Konkrete Syntax:	Durch die implizite Definition gibt es keine grafische Repräsentation.
Einschränkungen:	keine
Beispiele:	keine

A.3.32 MMMB::Prozessmodell::AktivitätHilfsmittel

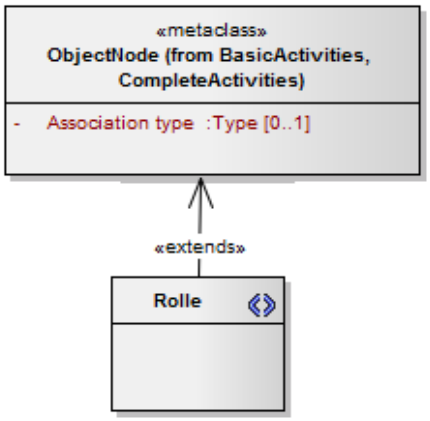
Profil Definition:



	Das Stereotyp «AktivitätHilfsmittel» wird von der Metaklasse Dependency abgeleitet.
Beschreibung und Semantik:	«AktivitätHilfsmittel» aggregiert eine Menge von Elementen des Typs «Aktivität», die dem «Hilfsmittel» zugewiesen werden. Diese Eigenschaft wird durch Typisierung als UML-Element `Dependency` erreicht. Es gilt die Semantikdefinition aus Abschnitt 7.3.12 in [Obj11c] mit den definierten Einschränkungen.
Attribute:	keine
Konkrete Syntax:	Siehe konkrete Syntax für das UML-Element `Dependency` (vgl. 7.3.12 in [Obj11c]). Zusätzlich soll dem Assoziationsnamen der Präfix „«AktivitätHilfsmittel»“ hinzugefügt werden.
Einschränkungen:	Assoziationen vom Typ «AktivitätHilfsmittel» dürfen nur zwischen Elementen vom Typ «Hilfsmittel» als Supplier und Elementen vom Typ «Aktivität» und «ZusammengesetzteAktivität» definiert werden. `context AktivitätHilfsmittel inv: self.supplier -> Exist(s | s.oclIsTypeOf(Hilfsmittel) and self.client -> Exist(c| c.oclIsTypeOf(Aktivität)) or c.oclIsTypeOf(ZusammengesetzteAktivität)`
Beispiele:	Siehe Beispiele «Hilfsmittel».

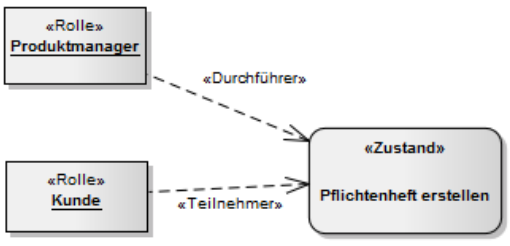
A.3.33 MMMB::Prozessmodell::Rolle

Profil Definition:



Beschreibung und Semantik:	Die «Rolle» wird von der Metaklasse `ObjectNode` abgeleitet. Elemente vom Typ «Rolle» definieren informell einer «Aktivität» zugewiesene Rollen. Das Element «Rolle» wird über einen frei wählbaren String definiert. Die Identifizierung einer konkreten Rolle über den String muss vom Methodenentwickler erfolgen. Eine Rolle beschreibt den verantwortlichen Durchführer oder Teilnehmer einer Aktivität. Die Semantik des UML-Elements `ObjectNode` wie in Abschnitt 12.3.37 aus [Obj11c] soll beibehalten werden.
Attribute:	keine
Konkrete Syntax:	Die konkrete Syntax ist angelehnt an das UML-Element `ObjectNode` wie in Abschnitt 12.3.37 aus [Obj11c]. Dem Namen soll die Zeichenfolge „«Rolle»“ vorangestellt werden.
Einschränkungen:	Eine «Rolle» darf lediglich über «Teilnehmer» oder «Durchführer» «Aktivität»-Elemente referenzieren (siehe Einschränkungen «Teilnehmer» und «Durchführer»).

Beispiele:

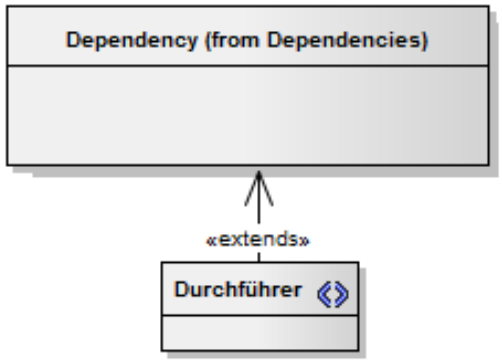


A.3.34 MMMB::Prozessmodell::ProzessmodellRolle

Profil Definition:	Diese Assoziation wird implizit über das Attribut «Prozessmodell».node definiert. Das Attribut enthält alle Elemente vom Typ ActivityNode, nach Definition auch Elemente vom Typ «Rolle».
Beschreibung und Semantik:	Die Assoziation «ProzessmodellRolle» beschreibt die Aggregation der Top-Level-Elemente «Rolle» im «Prozessmodell».
Attribute:	keine
Konkrete Syntax:	Durch die implizite Definition gibt es keine grafische Repräsentation.
Einschränkungen:	keine
Beispiele:	keine

A.3.35 MMMB::Prozessmodell::Durchführer

Profil Definition:



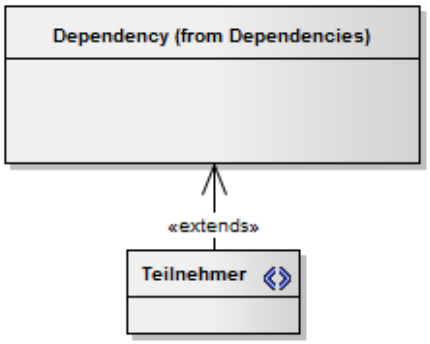
Das Stereotyp «Durchführer» wird von der Metaklasse `Dependency` abgeleitet.

Beschreibung und Semantik:	«Durchführer» referenziert Elemente des Typs «Aktivität», die der «Rolle» zugewiesen werden. Diese Eigenschaft wird durch Typisierung als UML-Element `Dependency` erreicht. Es gilt die Semantikdefinition aus Abschnitt 7.3.12 in [Obj11c] mit den definierten Einschränkungen. Die Bedeutung ist, dass eine «Rolle», die eine «Aktivität» als «Durchführer» referenziert, die verantwortliche Instanz zur Durchführung der Aktivität ist.
Attribute:	keine
Konkrete Syntax:	Siehe konkrete Syntax für UML-Dependency (vgl. 7.3.12 in [Obj11c]). Der konkreten Syntax soll das Stereotyp «Durchführer» hinzugefügt werden.

Einschränkungen:	Assoziationen vom Typ «Durchführer» dürfen nur zwischen Elementen vom Typ «Rolle» als Supplier und Elementen vom Typ «Aktivität» und «ZusammengesetzteAktivität» definiert werden.
	`context Durchführer inv: self.supplier ->`
	`Exist(s | s.oclIsTypeOf(Rolle) and self.client`
	`-> Exist(c| c.oclIsTypeOf(Aktivität)) or`
	`c.oclIsTypeOf(ZusammengesetzteAktivität)`

Beispiele:	Siehe Beispiele «Rolle».

A.3.36 MMMB::Prozessmodell::Teilnehmer

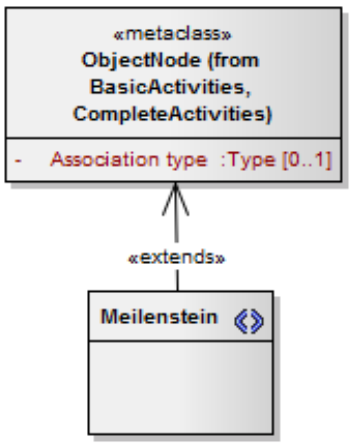
Profil Definition:	
	Das Stereotyp «Teilnehmer» wird von der Metaklasse `Dependency` abgeleitet.
Beschreibung und Semantik:	«Teilnehmer» referenziert Elemente des Typs «Aktivität», die der «Rolle» zugewiesen werden. Diese Eigenschaft wird durch Typisierung als UML-Element `Dependency` erreicht. Es gilt die Semantikdefinition aus Abschnitt 7.3.12 in [Obj11c] mit den in dieser Tabelle definierten Einschränkungen. Die Bedeutung ist, dass eine «Rolle», die eine «Aktivität» als «Teilnehmer» referenziert, lediglich an der Durchführung der «Aktivität» beteiligt sein soll.
Attribute:	keine
Konkrete Syntax:	Siehe konkrete Syntax für UML-Dependency (vgl. 7.3.12 in [Obj11c]). Der konkreten Syntax soll das Stereotyp „«Teilnehmer»“ hinzugefügt werden.

Einschränkungen:	Assoziationen vom Typ «Teilnehmer» dürfen nur zwischen Elementen vom Typ «Rolle» als Supplier und Elementen vom Typ «Aktivität» und «ZusammengesetzteAktivität» definiert werden.
	`context Teilnehmer inv: self.supplier ->`
	`Exist(s | s.oclIsTypeOf(Rolle) and self.client`
	`-> Exist(c| c.oclIsTypeOf(Aktivität)) or`
	`c.oclIsTypeOf(ZusammengesetzteAktivität)`

Beispiele:	Siehe Beispiele «Rolle».

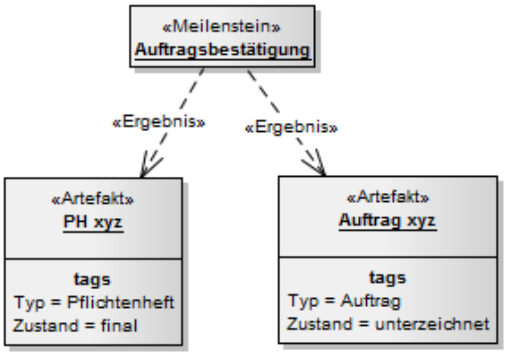
A.3.37 MMMB::Prozessmodell::Meilenstein

Profil Definition:



Beschreibung und Semantik:	Der «Meilenstein» wird von der Metaklasse `ObjectNode` abgeleitet. Elemente vom Typ «Meilenstein» aggregieren eine Menge von «Artefakt»-Elementen. Die Summe der «Artefakt»-Elemente mit ihren Zuständen definieren einen Gesamtzustand an erstellten «Artefakt»-Elementen, die einen Meilenstein definieren. Die Semantik des UML-Elements „ObjectNode“, wie in Abschnitt 12.3.37 aus [Obj11c], soll beibehalten werden.
Attribute:	keine
Konkrete Syntax:	Die konkrete Syntax ist angelehnt an das UML-Element `Object-Node`, wie in Abschnitt 12.3.37 aus [Obj11c]. Ein Element «Meilenstein» soll die Zeichenfolge „«Meilenstein»“ enthalten.
Einschränkungen:	Siehe Einschränkungen «Ergebnis».

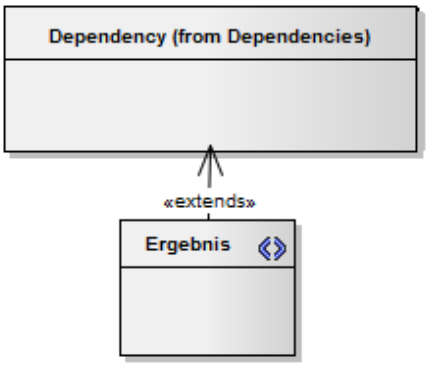
Beispiele:



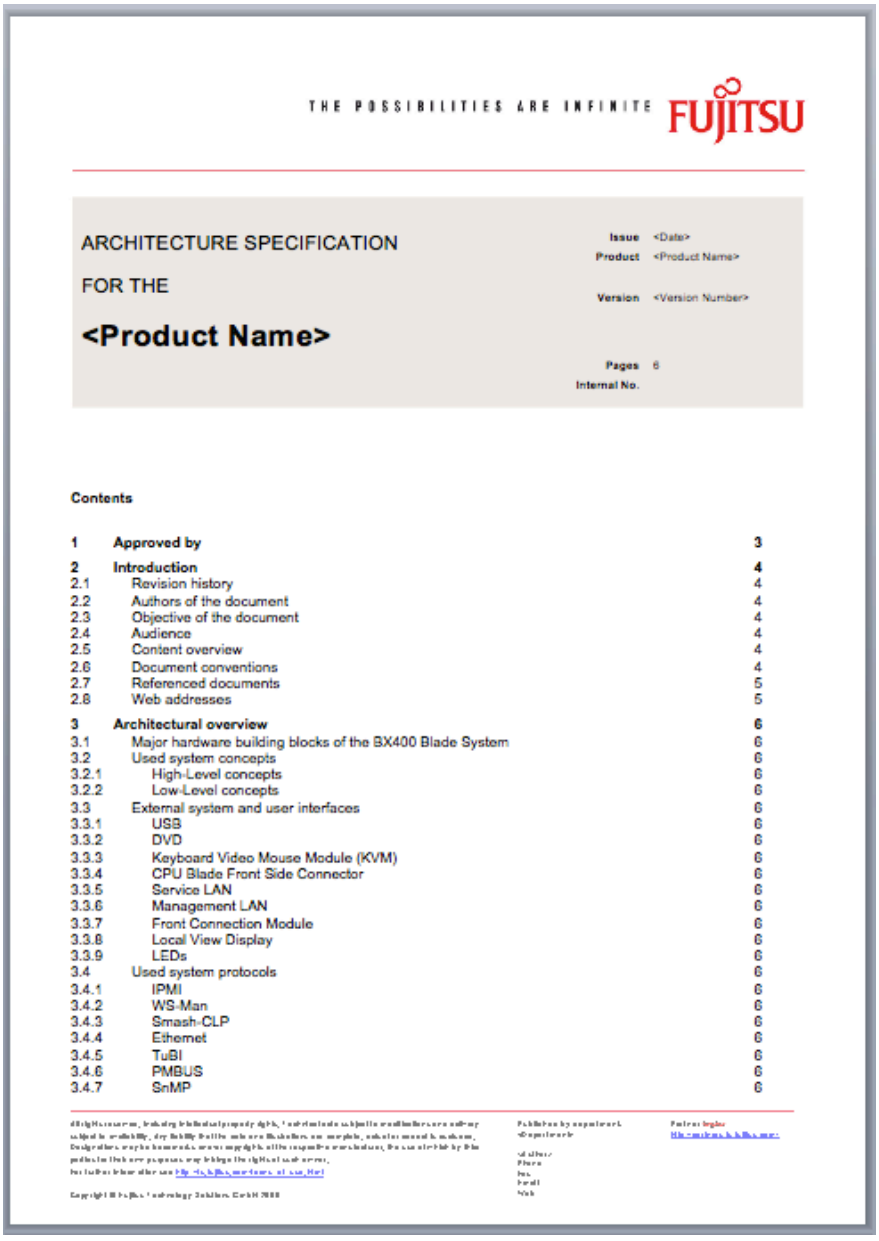
A.3.38 MMMB::Prozessmodell::ProzessmodellMeilenstein

Profil Definition:	Diese Assoziation wird implizit über das Attribut «Prozessmodell».node definiert. Das Attribut enthält alle Elemente vom Typ ActivityNode, nach Definition auch Elemente vom Typ «Meilenstein».
Beschreibung und Semantik:	Die Assoziation «ProzessmodellMeilenstein» beschreibt die Aggregation der Top-Level-Elemente «Meilenstein» im «Prozessmodell».
Attribute:	keine
Konkrete Syntax:	Durch die implizite Definition gibt es keine grafische Repräsentation.
Einschränkungen:	keine
Beispiele:	keine

A.3.39 MMMB::Prozessmodell::Ergebnis

Profil Definition:	
	Das Stereotyp «Ergebnis» wird von der Metaklasse Dependency abgeleitet.
Beschreibung und Semantik:	«Ergebnis» referenziert Elemente des Typs «Artefakt», die einem «Meilenstein» zugewiesen werden. Diese Eigenschaft wird durch Typisierung als UML-Dependency erreicht. Es gilt die Semantikdefinition aus Abschnitt 7.3.12 in [Obj11c] mit den definierten Einschränkungen. Die Bedeutung ist, dass ein «Meilenstein» ein «Artefakt» als «Ergebnis» referenziert. Die Menge an Ergebnissen definiert einen Meilenstein.
Attribute:	keine
Konkrete Syntax:	Siehe konkrete Syntax für das UML-Element `Dependency` (vgl. 7.3.12 in [Obj11c]). Zusätzlich soll der konkreten Syntax das Stereotyp «Ergebnis» hinzugefügt werden.
Einschränkungen:	Die Verbindung vom Typ «Ergebnis» darf nur zwischen «Meilenstein»-Elementen und «Artefakt»-Elementen definiert werden. `context Ergebnis inv: self.supplier -> Exist(s | s.oclIsTypeOf(Meilenstein)) and self.client -> Exist(c| c.oclIsTypeOf(Artefakt))`
Beispiele:	Siehe Beispiele «Meilenstein».

A.4 Beispiel Dokumentvorlage für die System-Architektur-Spezifikation



THE POSSIBILITIES ARE INFINITE FUJITSU

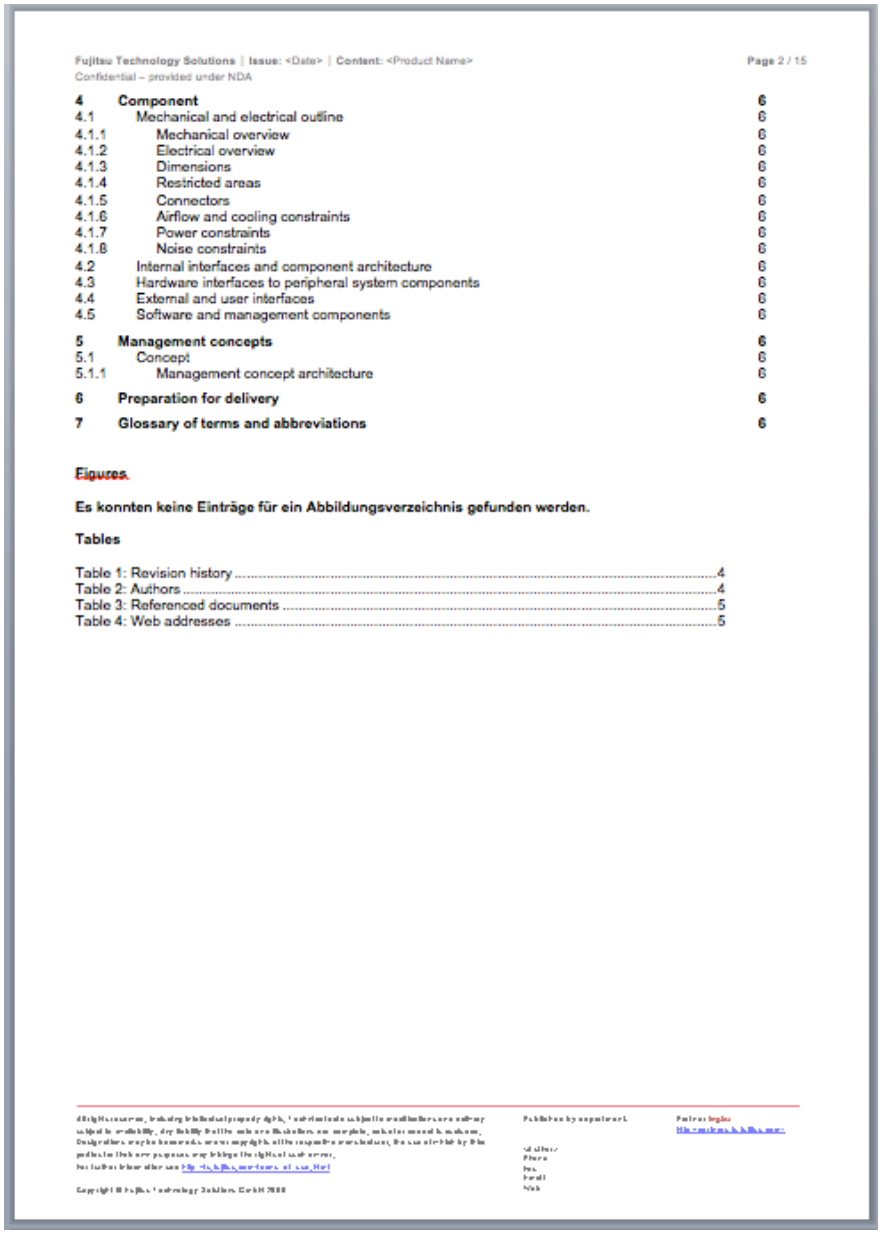
ARCHITECTURE SPECIFICATION

FOR THE

<Product Name>

Issue <Date>
Product <Product Name>
Version <Version Number>
Pages 6
Internal No.

Contents



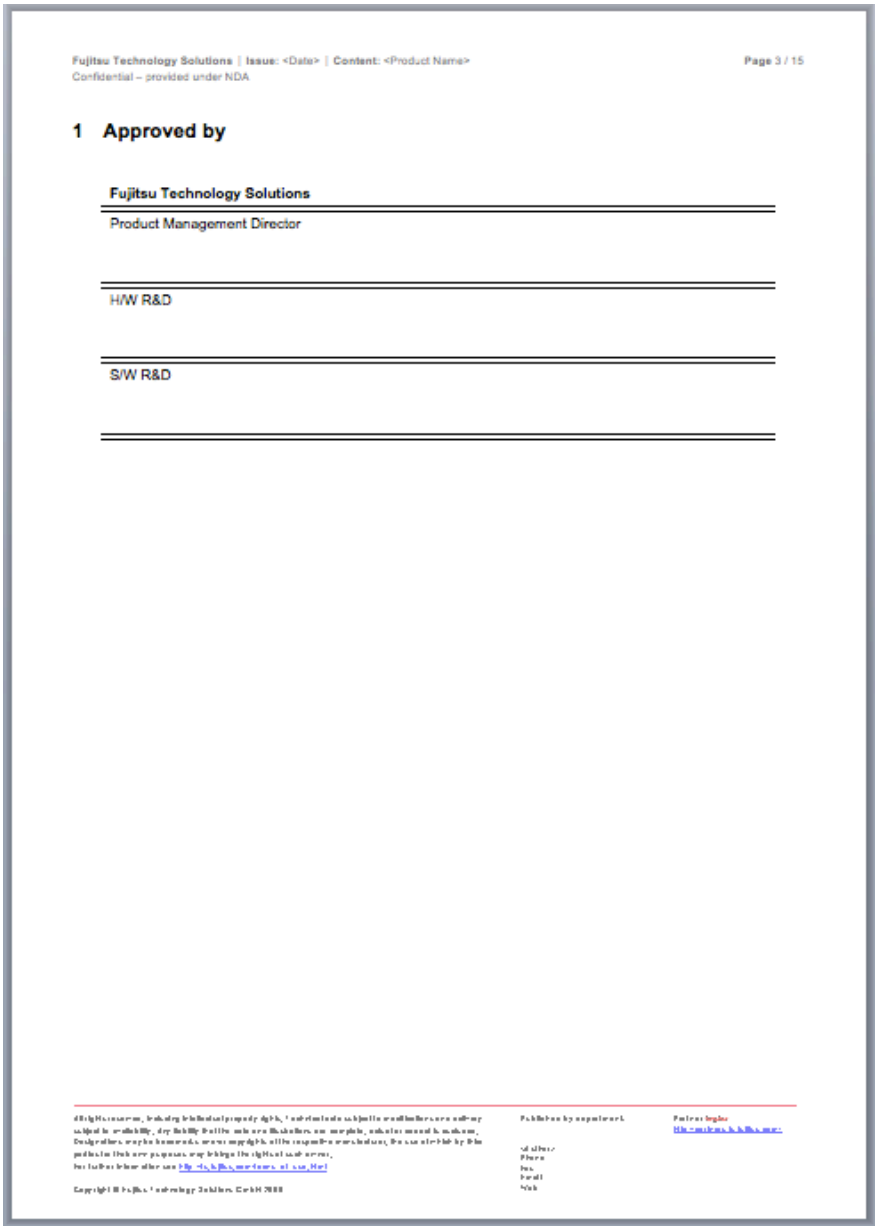
Figures

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

Tables

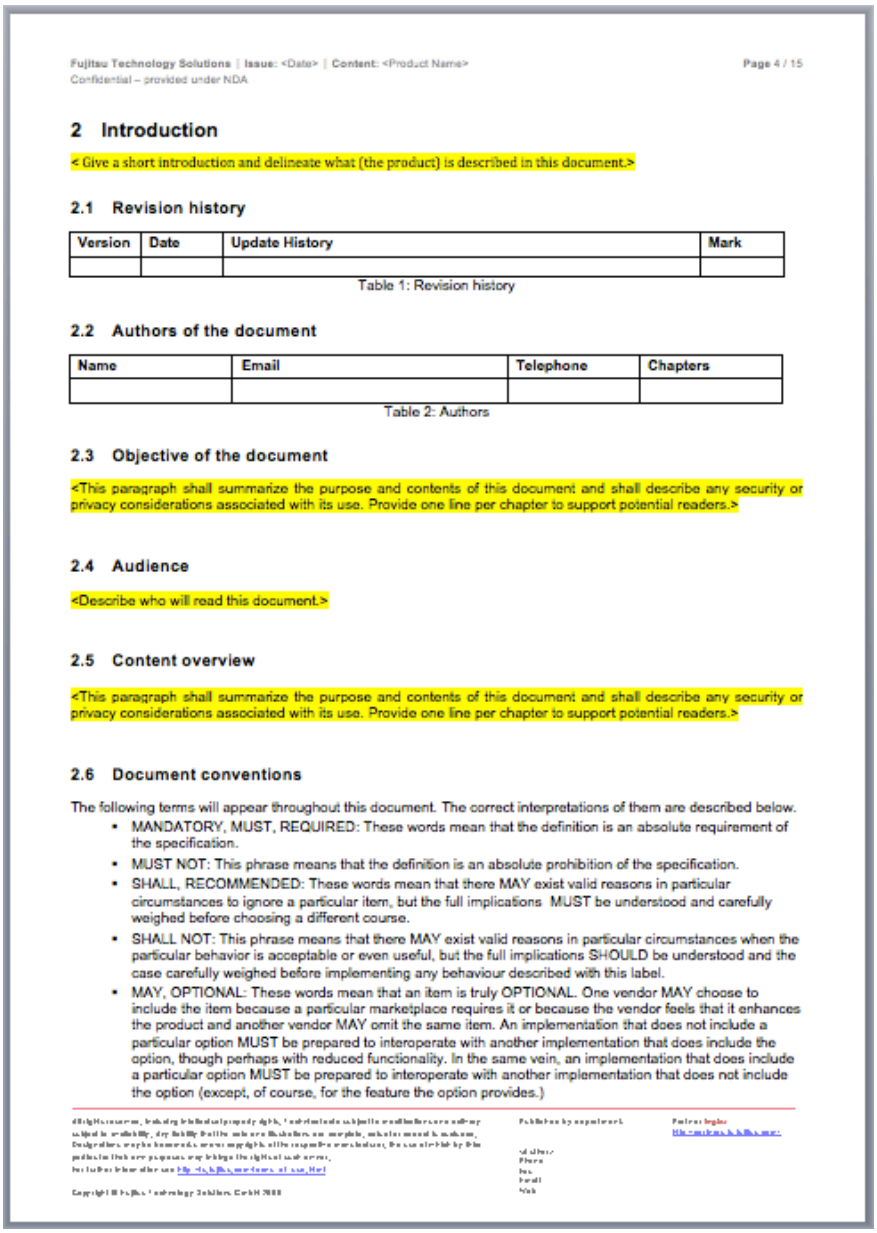
Published by

Author:



1 Approved by

Fujitsu Technology Solutions
Product Management Director
H/W R&D
S/W R&D



2 Introduction

< Give a short introduction and delineate what (the product) is described in this document.>

2.1 Revision history

Version	Date	Update History	Mark

Table 1: Revision history

2.2 Authors of the document

Name	Email	Telephone	Chapters

Table 2: Authors

2.3 Objective of the document

<This paragraph shall summarize the purpose and contents of this document and shall describe any security or privacy considerations associated with its use. Provide one line per chapter to support potential readers.>

2.4 Audience

<Describe who will read this document.>

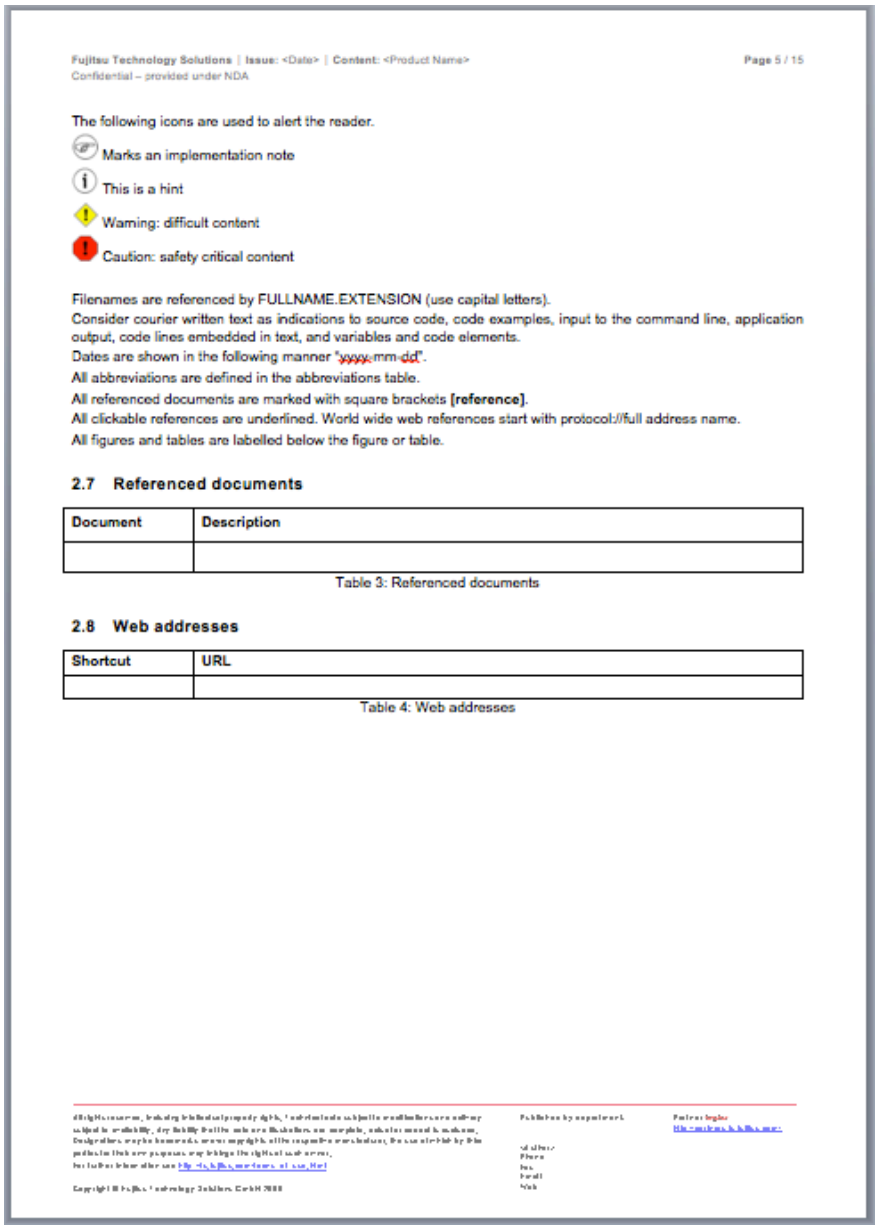
2.5 Content overview

<This paragraph shall summarize the purpose and contents of this document and shall describe any security or privacy considerations associated with its use. Provide one line per chapter to support potential readers.>

2.6 Document conventions

The following terms will appear throughout this document. The correct interpretations of them are described below.

- MANDATORY, MUST, REQUIRED: These words mean that the definition is an absolute requirement of the specification.
- MUST NOT: This phrase means that the definition is an absolute prohibition of the specification.
- SHALL, RECOMMENDED: These words mean that there MAY exist valid reasons in particular circumstances to ignore a particular item, but the full implications MUST be understood and carefully weighed before choosing a different course.
- SHALL NOT: This phrase means that there MAY exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications SHOULD be understood and the case carefully weighed before implementing any behaviour described with this label.
- MAY, OPTIONAL: These words mean that an item is truly OPTIONAL. One vendor MAY choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product and another vendor MAY omit the same item. An implementation that does not include a particular option MUST be prepared to interoperate with another implementation that does include the option, though perhaps with reduced functionality. In the same vein, an implementation that does include a particular option MUST be prepared to interoperate with another implementation that does not include the option (except, of course, for the feature the option provides.)



The following icons are used to alert the reader.

Marks an implementation note

This is a hint

Warning: difficult content

Caution: safety critical content

Filenames are referenced by FULLNAME.EXTENSION (use capital letters).
Consider courier written text as indications to source code, code examples, input to the command line, application output, code lines embedded in text, and variables and code elements.
Dates are shown in the following manner "yyyy-mm-dd".
All abbreviations are defined in the abbreviations table.
All referenced documents are marked with square brackets **[reference]**.
All clickable references are underlined. World wide web references start with protocol://full address name.
All figures and tables are labelled below the figure or table.

2.7 Referenced documents

Document	Description

Table 3: Referenced documents

2.8 Web addresses

Shortcut	URL

Table 4: Web addresses

Published by
Fujitsu Technology Solutions GmbH
Mies-van-der-Rohe-Straße 8
80807 München
Germany

Web
http://www.fujitsu.com/fts/

Fujitsu Technology Solutions | Issue: <Date> | Content: <Product Name> Page 6 / 15
Confidential – provided under NDA

3 Architectural overview

<This paragraph shall briefly state the hardware and management concept of the blade system. Additionally a short introduction about the content of this chapter should be provided. >

3.1 Major hardware building blocks of the <Product Name>

<Present an overview of the system. A SysML block definition diagram is recommended to define number and dependencies of all blocks. Additionally a picture of a possible solution can help to get an idea of the location of blocks and general overview.>

3.2 Used system concepts

<State here an overview, details in separate chapter in this document>

3.2.1 High-Level concepts

<Provide short description of each concept.>

Example:

3.2.1.1 System management

3.2.1.1.1 Power states

3.2.1.1.2 Power management concept

3.2.1.2 I/O concept

3.2.1.3 Shared storage

3.2.1.4 VIOM

3.2.1.5 Continuous operation

3.2.1.6 Cooling concept

3.2.2 Low-Level concepts

<Provide short description of each concept.>

Example:

3.2.2.1 Slot assignment

3.2.2.2 ACPI

3.2.2.3 SDR

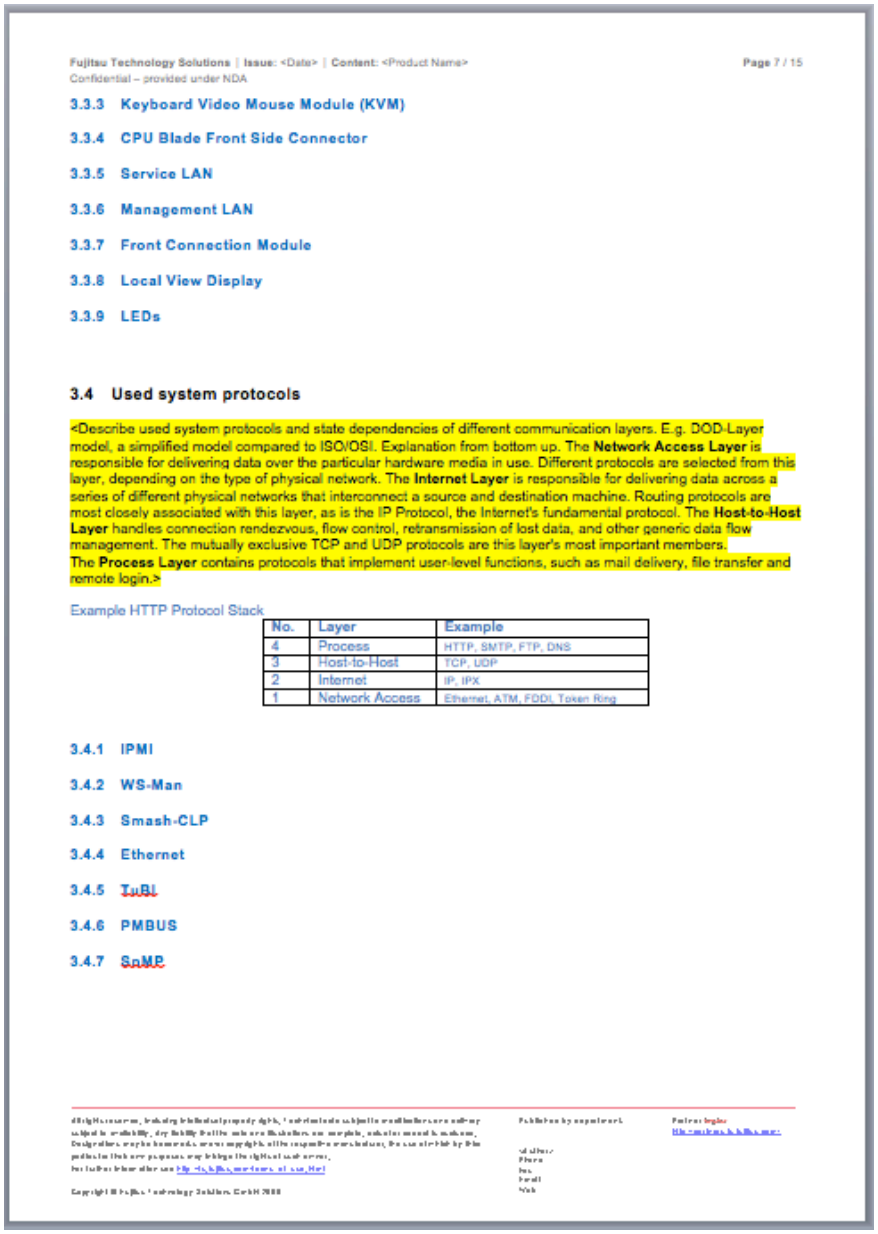
3.3 External system and user interfaces

<A brief description about Interfaces to the outer world and the management capability.>

Example:

3.3.1 USB

3.3.2 DVD



3.3.3 Keyboard Video Mouse Module (KVM)

3.3.4 CPU Blade Front Side Connector

3.3.5 Service LAN

3.3.6 Management LAN

3.3.7 Front Connection Module

3.3.8 Local View Display

3.3.9 LEDs

3.4 Used system protocols

<Describe used system protocols and state dependencies of different communication layers. E.g. DOD-Layer model, a simplified model compared to ISO/OSI. Explanation from bottom up. The **Network Access Layer** is responsible for delivering data over the particular hardware media in use. Different protocols are selected from this layer, depending on the type of physical network. The **Internet Layer** is responsible for delivering data across a series of different physical networks that interconnect a source and destination machine. Routing protocols are most closely associated with this layer, as is the IP Protocol, the Internet's fundamental protocol. The **Host-to-Host Layer** handles connection rendezvous, flow control, retransmission of lost data, and other generic data flow management. The mutually exclusive TCP and UDP protocols are this layer's most important members.
The **Process Layer** contains protocols that implement user-level functions, such as mail delivery, file transfer and remote login.>

Example HTTP Protocol Stack

No.	Layer	Example
4	Process	HTTP, SMTP, FTP, DNS
3	Host-to-Host	TCP, UDP
2	Internet	IP, IPX
1	Network Access	Ethernet, ATM, FDDI, Token Ring

3.4.1 IPMI

3.4.2 WS-Man

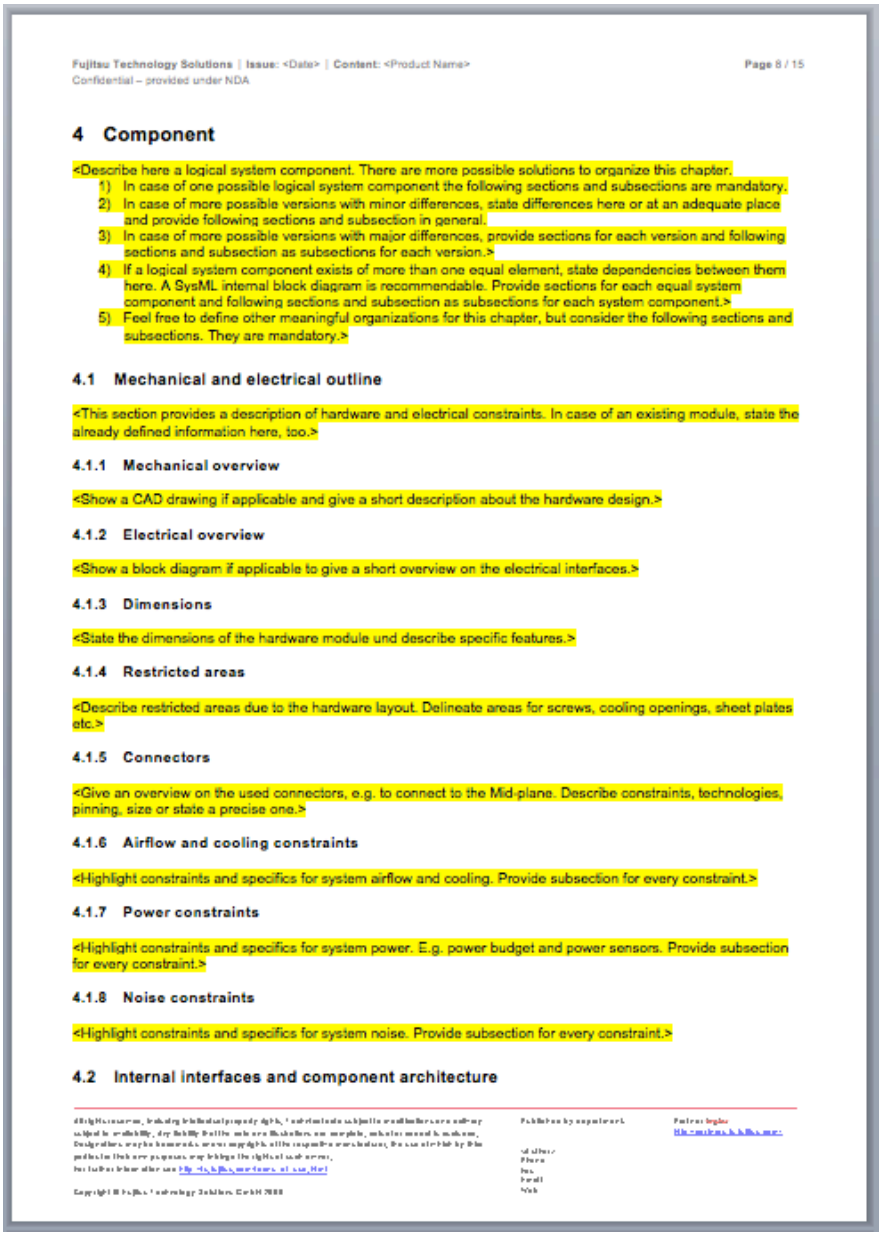
3.4.3 Smash-CLP

3.4.4 Ethernet

3.4.5 TuBI

3.4.6 PMBUS

3.4.7 SnMP



4 Component

<Describe here a logical system component. There are more possible solutions to organize this chapter.
1) In case of one possible logical system component the following sections and subsections are mandatory.
2) In case of more possible versions with minor differences, state differences here or at an adequate place and provide following sections and subsection in general.
3) In case of more possible versions with major differences, provide sections for each version and following sections and subsection as subsections for each version.>
4) If a logical system component exists of more than one equal element, state dependencies between them here. A SysML internal block diagram is recommendable. Provide sections for each equal system component and following sections and subsection as subsections for each system component.>
5) Feel free to define other meaningful organizations for this chapter, but consider the following sections and subsections. They are mandatory.>

4.1 Mechanical and electrical outline

<This section provides a description of hardware and electrical constraints. In case of an existing module, state the already defined information here, too.>

4.1.1 Mechanical overview

<Show a CAD drawing if applicable and give a short description about the hardware design.>

4.1.2 Electrical overview

<Show a block diagram if applicable to give a short overview on the electrical interfaces.>

4.1.3 Dimensions

<State the dimensions of the hardware module und describe specific features.>

4.1.4 Restricted areas

<Describe restricted areas due to the hardware layout. Delineate areas for screws, cooling openings, sheet plates etc.>

4.1.5 Connectors

<Give an overview on the used connectors, e.g. to connect to the Mid-plane. Describe constraints, technologies, pinning, size or state a precise one.>

4.1.6 Airflow and cooling constraints

<Highlight constraints and specifics for system airflow and cooling. Provide subsection for every constraint.>

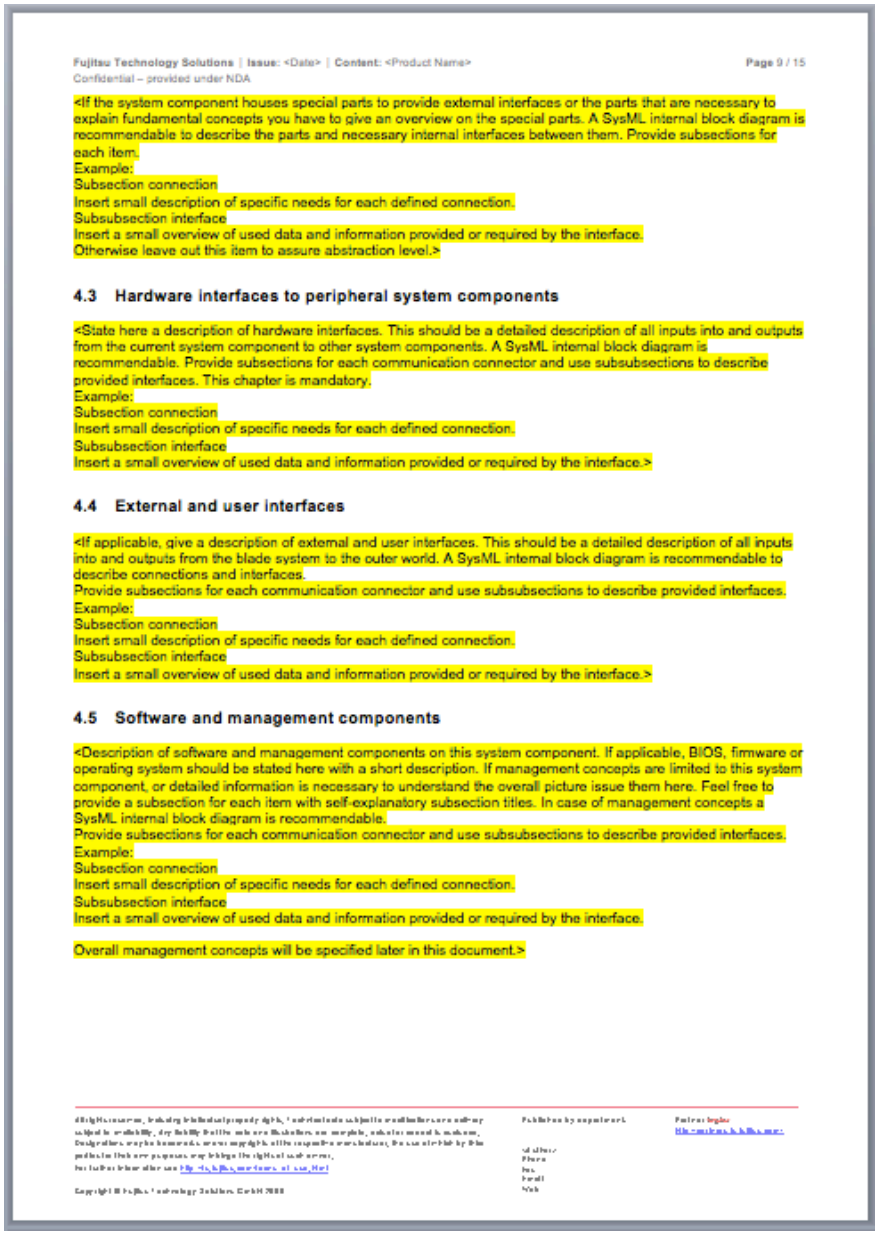
4.1.7 Power constraints

<Highlight constraints and specifics for system power. E.g. power budget and power sensors. Provide subsection for every constraint.>

4.1.8 Noise constraints

<Highlight constraints and specifics for system noise. Provide subsection for every constraint.>

4.2 Internal interfaces and component architecture



<If the system component houses special parts to provide external interfaces or the parts that are necessary to explain fundamental concepts you have to give an overview on the special parts. A SysML internal block diagram is recommendable to describe the parts and necessary internal interfaces between them. Provide subsections for each item.
Example:
Subsection connection
Insert small description of specific needs for each defined connection.
Subsubsection interface
Insert a small overview of used data and information provided or required by the interface.
Otherwise leave out this item to assure abstraction level.>

4.3 Hardware interfaces to peripheral system components

<State here a description of hardware interfaces. This should be a detailed description of all inputs into and outputs from the current system component to other system components. A SysML internal block diagram is recommendable. Provide subsections for each communication connector and use subsubsections to describe provided interfaces. This chapter is mandatory.
Example:
Subsection connection
Insert small description of specific needs for each defined connection.
Subsubsection interface
Insert a small overview of used data and information provided or required by the interface.>

4.4 External and user interfaces

<If applicable, give a description of external and user interfaces. This should be a detailed description of all inputs into and outputs from the blade system to the outer world. A SysML internal block diagram is recommendable to describe connections and interfaces.
Provide subsections for each communication connector and use subsubsections to describe provided interfaces.
Example:
Subsection connection
Insert small description of specific needs for each defined connection.
Subsubsection interface
Insert a small overview of used data and information provided or required by the interface.>

4.5 Software and management components

<Description of software and management components on this system component. If applicable, BIOS, firmware or operating system should be stated here with a short description. If management concepts are limited to this system component, or detailed information is necessary to understand the overall picture issue them here. Feel free to provide a subsection for each item with self-explanatory subsection titles. In case of management concepts a SysML internal block diagram is recommendable.
Provide subsections for each communication connector and use subsubsections to describe provided interfaces.
Example:
Subsection connection
Insert small description of specific needs for each defined connection.
Subsubsection interface
Insert a small overview of used data and information provided or required by the interface.

Overall management concepts will be specified later in this document.>

5 Management concepts

<Describe here overall management concepts. Provide a separate section for each concept.>

5.1 Concept

<Describe the system concept here. You may use the full spectrum of UML and SysML diagrams. Consider UML activity diagrams to explain sequences of actions and UML Sequence diagram to explain example scenarios. Feel free to add meaningful subsections. Consider overall concept architecture as a separate subsection.>

5.1.1 Concept Use Cases

<A **use case** is a description of a system's behavior# as it responds to a request that originates from outside of that system. In other words, a use case describes "who" can do "what" with the system in question. The use case technique is used to capture a system's behavioral requirements by detailing scenario-driven threads through the functional requirements. A Use Use refines a functional requirement.>

Example

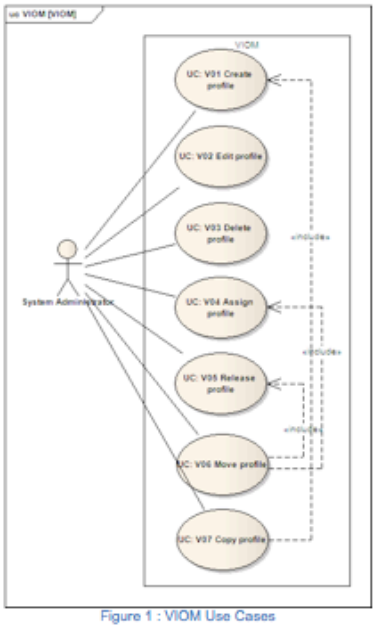


Figure 1 : VIOM Use Cases

Published by
Fujitsu Technology Solutions
Email
Web

Publication date
Template version

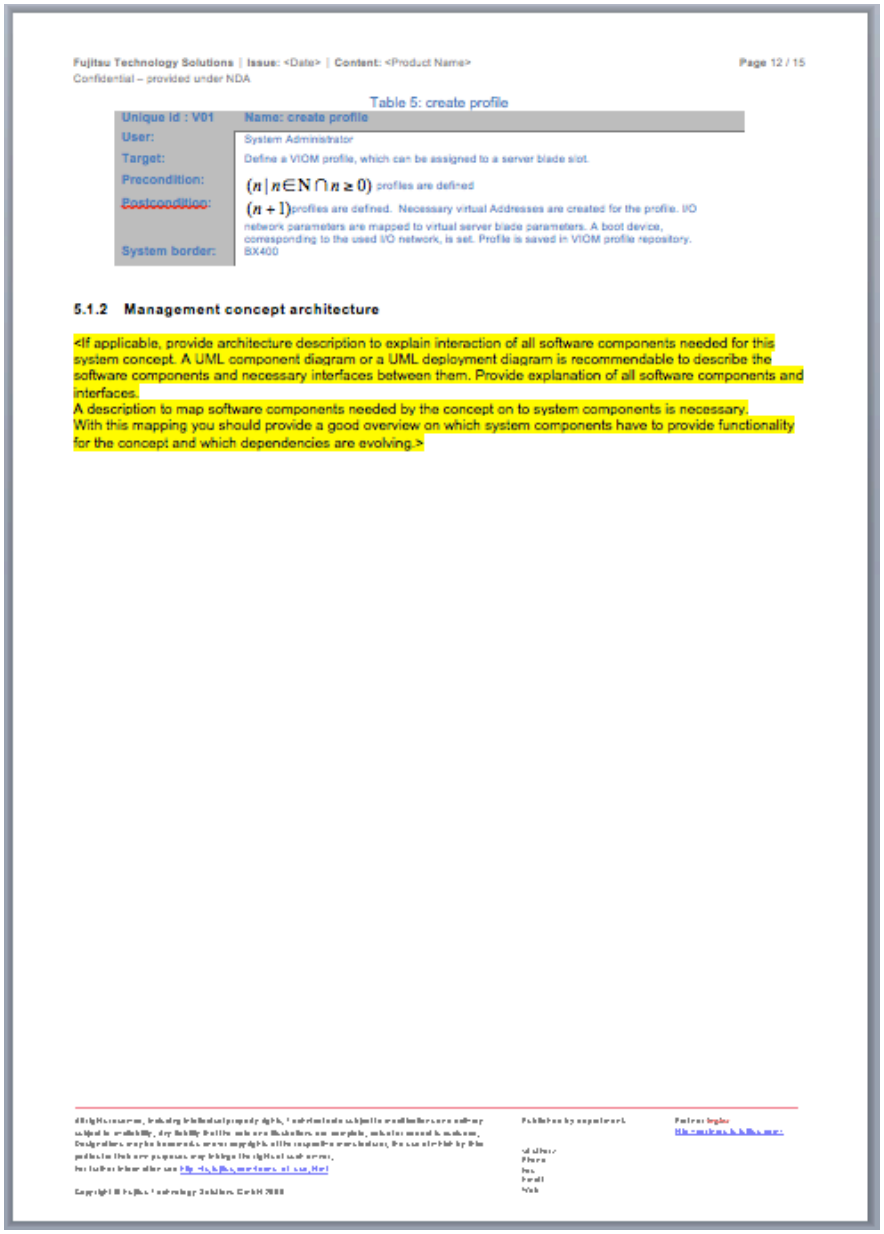
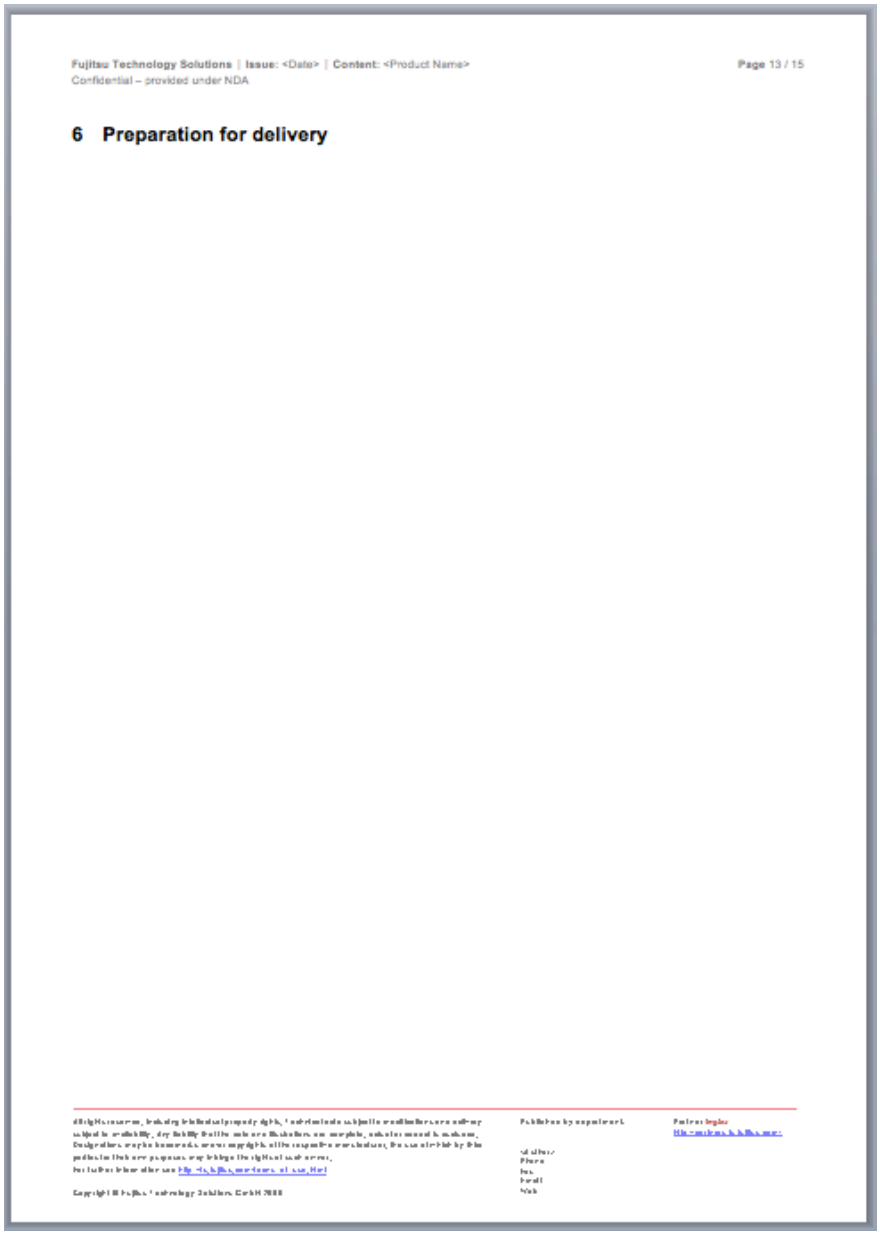


Table 5: create profile

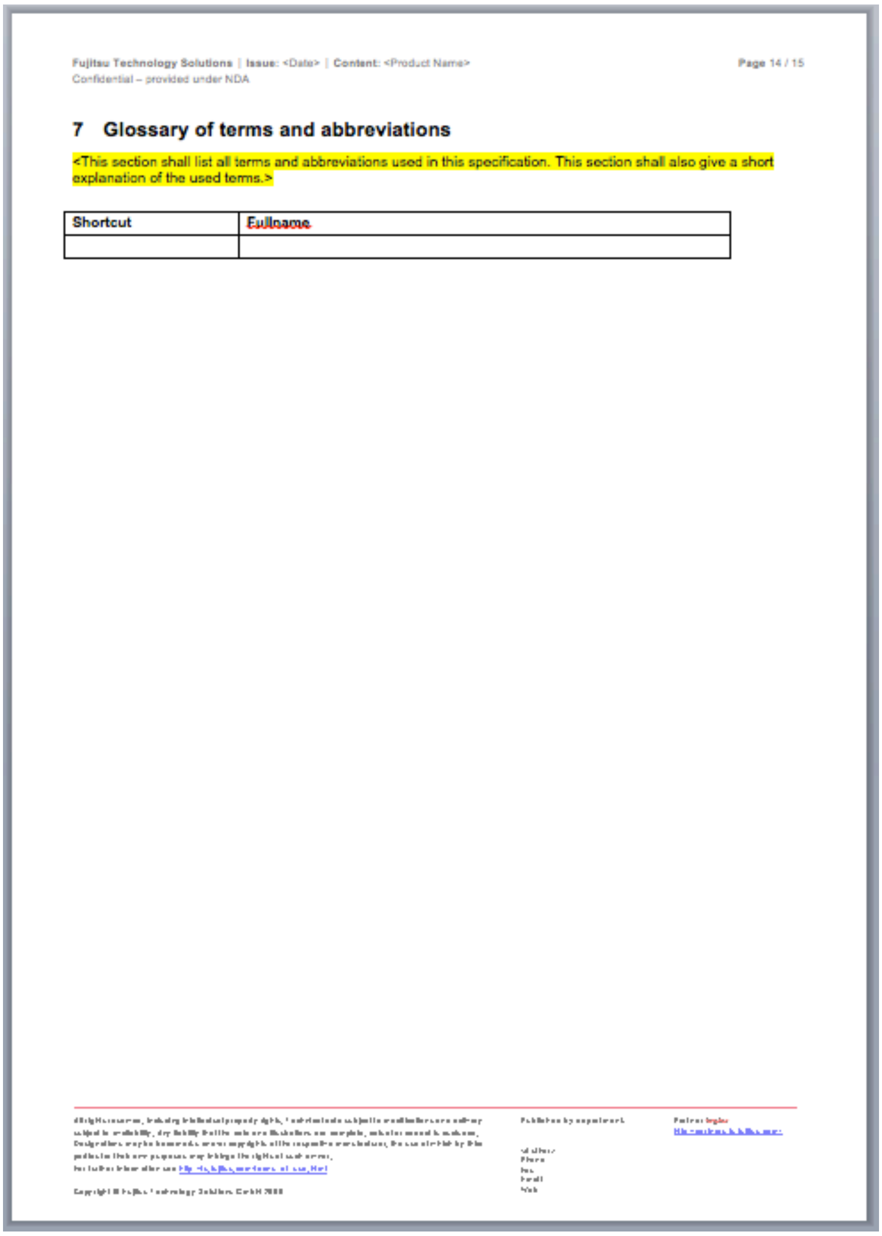
Unique id : V01	Name: create profile
User:	System Administrator
Target:	Define a VIOM profile, which can be assigned to a server blade slot.
Precondition:	$(n \mid n \in \mathbb{N} \cap n \geq 0)$ profiles are defined
Postcondition:	$(n+1)$profiles are defined. Necessary virtual Addresses are created for the profile. I/O network parameters are mapped to virtual server blade parameters. A boot device, corresponding to the used I/O network, is set. Profile is saved in VIOM profile repository.
System border:	BX400

5.1.2 Management concept architecture

<If applicable, provide architecture description to explain interaction of all software components needed for this system concept. A UML component diagram or a UML deployment diagram is recommendable to describe the software components and necessary interfaces between them. Provide explanation of all software components and interfaces.
A description to map software components needed by the concept on to system components is necessary. With this mapping you should provide a good overview on which system components have to provide functionality for the concept and which dependencies are evolving.>



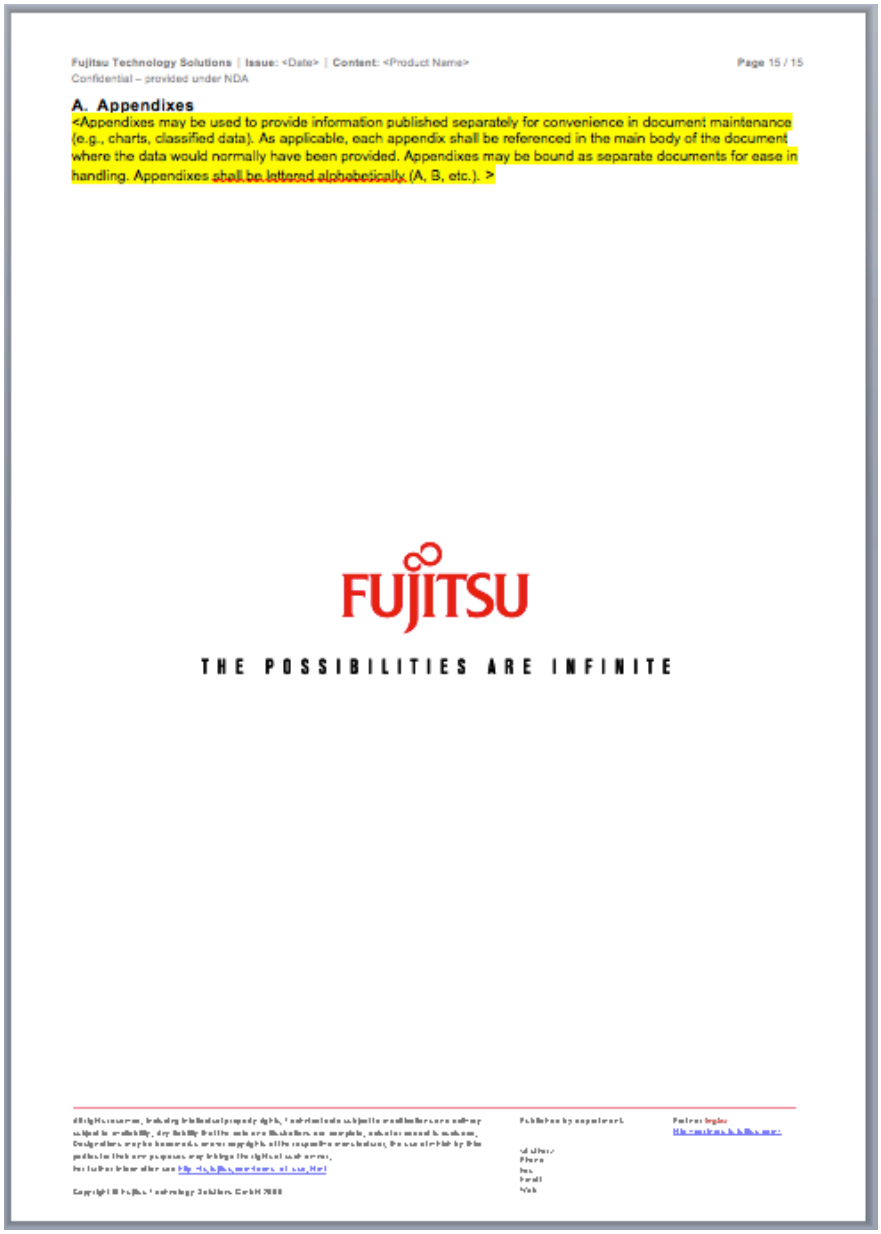
6 Preparation for delivery


Fujitsu Technology Solutions | Issue: <Date> | Content: <Product Name> Page 14 / 15
Confidential – provided under NDA

7 Glossary of terms and abbreviations

<This section shall list all terms and abbreviations used in this specification. This section shall also give a short explanation of the used terms.>

Shortcut	Fullname



A. Appendixes

<Appendixes may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data). As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided. Appendixes may be bound as separate documents for ease in handling. Appendixes shall be lettered alphabetically (A, B, etc.). >

FUJITSU

THE POSSIBILITIES ARE INFINITE

A.5 MOF-Datei-Generierung

Dieser Abschnitt stellt die MOF-Datei-Generierungstemplates für den Enterprise Architect vor. Der Template-basierte Codegenerator des Enterprise Architect stellt Basis-Templates für jede Eigenschaft der UML-Elemente, die er kennt, bzw. für die Profil-Elemente eines UML-Profils bereit. Bei der Auswahl eines Elements im Projektbrowser und beim Starten der Codegenerierung fragt das Werkzeug nach einer Sprachdefinition. Für eine Sprachdefinition können die vorgegebenen Templates ausgewählt und angepasst werden. Die ausgewählten und angepassten Templates für die Erstellung einer MOF-Datei werden im Folgenden dargestellt.

```
// Copyright Fujitsu Technology Solutions 2013
// %fileName%
// Created on: %eaDateTime%
%if classAuthor != ""%
// Original author: %classAuthor%
%endIf%
// ================================================
// %className%
// ================================================
%Class%
```

```
%ClassDeclaration%
{
%ClassBody%
};
```

Listing A.2: *Class Template für die MOF-Datei*

```
%PI=""%
[%classAbstract=="T"?"Abstract, "%
%classTag:"Association"?"Association,\n"%
%classTag:"Indication"?"Indication,\n"%
%classTag:"UMLPackagePath"?"UMLPackagePath " value ",\n"%
Description(\n
%qt%%classNotes%%qt%
)]\n
class %className% : %ClassInherits%
```

Listing A.3: *ClassDeclaration Template für die MOF-Datei*

```
%list="Attribute" @separator="\n" @indent="\t"%\n
%list="Operation" @separator="\n\n" @indent="\t"%
```

Listing A.4: *ClassBody Template für die MOF-Datei*

```
$bases=%list="ClassBase"
% @separator=", "%
$bases
```

Listing A.5: *ClassBody Template für die MOF-Datei*

```
%AttributeDeclaration%
```

Listing A.6: *Attribute Template für die MOF-Datei*

```
%PI=""%
[Description (\n
%qt%%attNotes%%qt%
\n)
```

```
%attTag:"ValueMap"?",\nValueMap " value %
%attTag:"Values"?",\nValues " value %
%attTag:"Required"?",\nRequired " value %
%attTag:"Override"?",\nOverride " value %
%attTag:"ModelCorrespondance"?",\nModelCorrespondance " value %
%attTag:"MinLen"?",\nMinLen " value %
%attTag:"MinValue"?",\nMinValue " value %
%attTag:"MaxLen"?",\nMaxLen " value %
%attTag:"MaxValue"?",\nMaxValue " value %
]\n
%attType% %attName%;
```

Listing A.7: *AttributeDeclaration Template für die MOF-Datei*

```
%linkParentName%
```

Listing A.8: *LinkBaseClass Template für die MOF-Datei*

```
%OperationDeclaration%
```

Listing A.9: *Operation Template für die MOF-Datei*

```
%PI=""%
[Description (\n
%qt%%opNotes%%qt%
\n)
%opTag:"ValueMap"?",\nValueMap " value %
%opTag:"Values"?",\nValues " value %
%opTag:"Override"?",\nOverride " value %
%opTag:"ModelCorrespondance"?",\nModelCorrespondance " value %
]\n
%opReturnType%%opReturnArray=="T" ? "[]" : ""% %opName%(\n
%list="Parameter" @separator=",\n" @indent="\t"%
\n);
```

Listing A.10: *OperationDeclaration Template für die MOF-Datei*

```
%PI=""%
[%paramKind=="in"?"In, "%
%paramKind=="out"?"In (false), Out, "%
%paramKind=="inout"?"In, Out, "%
Description (\n
%qt%%paramNotes%%qt%
\n)
]\n
%paramType% %paramName%
```

Listing A.11: *Parameter Template für die MOF-Datei*

Mit Hilfe der erstellten Textvorlagen lässt sich ein beliebiges im EA modelliertes CIM-Element (CIM-Klasse oder CIM-Assoziation) in die MOF-Datei-Darstellung überführen. Als Beispiel wird in Abbildung A.7 die CIM-Klasse in ihrer Repräsentation im EA-CIM-Modell dargestellt.

In der folgenden Auflistung wird die entsprechend generierte MOF-Datei gezeigt.

```
// Copyright Fujitsu Technology Solutions 2013
// SVS_iRMCServiceProcessor.mof
// Created on: 08-Sep-2014 16:12:18
// Original author: Michael Spijkerman
// ==========================================
// SVS_iRMCServiceProcessor
// ==========================================
```

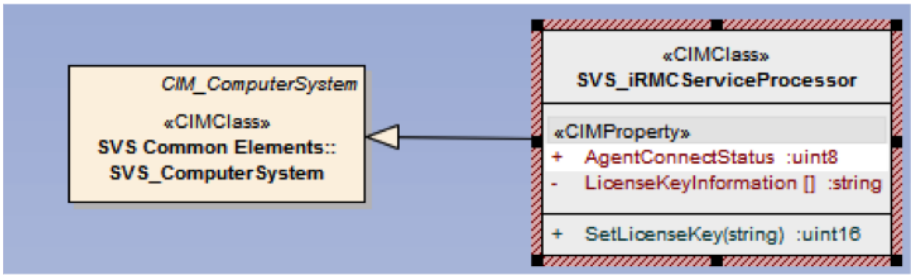


Abbildung A.7: *Beispiel: CIM-Klasse SVS iRMC Service Processor*

```
8  [Version("1.0.0"),
9  UMLPackagePath ( "CIM::SVS::iRMC::SVS_iRMCServiceProcessor" ),
10 Description( "An instance of this class describes the logical iRMC S4 of a base
11 server." )]
12 class SVS_iRMCServiceProcessor : SVS_ComputerSystem
13 {
14         [Description (
15         "This property returns the agent connect status."
16         "0 Disconnected"
17         "1 Management agent connected"
18         "2 Agentless service connected"
19         ),
20         ValueMap { "0","1","2" },
21         Values { "Disconnected","SV Management Agent","Agentless service connected" }]
22         uint8 AgentConnectStatus;
23
24         [Description (
25         "This properties shows license key information. For each license entity one
26         index is available that show the current information in string format." )]
27         string LicenseKeyInformation [];
28
29         [Description (  "This method uploads a license key."    )]
30         uint16 SetLicenseKey(
31                 [IN, Description (
32                 ""
33                 )]
34                 string LicenseKey
35         );
36 };
```

Listing A.12: *Beispiel MOF-Datei für die CIM-Klasse SVS iRMC Service Processor*

A.6 Beispiel der CIM-Entwicklung mit dem EA

Die folgende Abbildung A.8 stellt die für die CIM-Entwicklung angepasste Entwicklungsumgebung auf Basis des Enterprise Architects dar.

Auf der linken Seite findet sich die für das SVS CIM Profile (UML-Profile) angepasste Werkzeugleiste. In der Mitte findet sich exemplarisch ein spezifiziertes Profil-Diagramm des SVS iRMC Base Servers. Auf der rechten Seite ist das entsprechende CIM-Modell zu erkennen. Im unteren Bereich findet sich das TaggedValue-Fenster, in dem CIM-spezifische Metadaten für sämtliche Modellelemente eingegeben werden können.

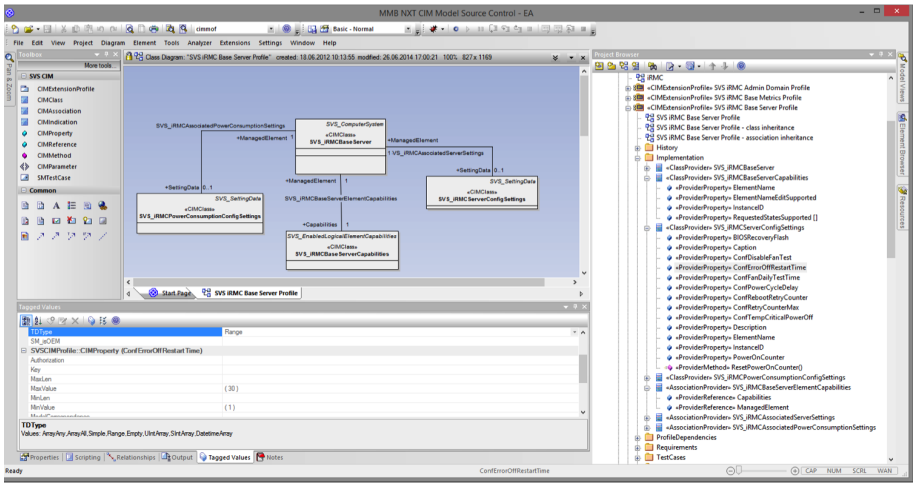


Abbildung A.8: *Beispiel: CIM-Entwicklung mit dem EA und dem SVS CIM-Schema*

A.7 Beispiel der Testausführung mit der JavaCIMTestSuite

Bei der CIM-Testfallausführung werden die vom CIM-Server zurückgelieferten Instanzen mit einer aus dem CIM-Modell erstellten Testinstanz verglichen.

Die JavaCIMTestSuite versucht in einem ersten Schritt die in XMI bereitgestellten Modellinformationen in eine Testinstanz zu überführen. Dafür wurde ein Parser programmiert. Beim Parsen wird bereits überprüft, ob die Modellinformationen vollständig sind. Für jedes CIM-Profil werden die definierten CIM-Provider untersucht. Dabei werden folgende Eigenschaften überprüft.

- Ist der CIM-Provider für eine CIM-Klasse oder CIM-Assoziation definiert.
- Ist die minimale und maximale Anzahl der möglichen Instanzen definiert.

Für CIM-Provider, die CIM-Klassen implementieren, werden die einzelnen definierten CIM-Properties auf folgende Eigenschaften überprüft.

- Ist für jedes CIM-Property der Testdatentyp definiert (uint8, uint 16, uint32, sint 8, sint16, sint32, sint64, string, datetime, boolean und Arrays der aufgeführten Datentypen).
- Sind gültige Testwerte für die CIM-Properties definiert (feste Werte, dynamische Werte, Bereiche, beliebige Werte)

Für CIM-Provider, die CIM-Assoziationen implementieren, können zusätzlich noch eine Reihe an Bedingungen definiert werden, die gültige Assoziationspaare bzw. Paare der referenzierten Instanzen der CIM-Klassen definieren.

Sind die Informationen der CIM-Modell nicht ausreichend oder fehlerhaft, so dass keine Testinstanz gebildet werden kann, wird der Benutzer bereits an dieser Stelle darauf hingewiesen das CIM-Modell anzupassen.

Sind für alle CIM-Elemente ein CIM-Profil erstellt worden, werden alle Instanzen für einen CIM-Provider des CIM-Profils abgefragt und jede Instanz mit der Testinstanz verglichen. Dabei werden die folgenden Tests durchgeführt.

- Liefert der CIM-Provider Instanzen?
- Liegt die Anzahl der Instanzen im definierten Bereich (min, max)?
- Werden Werte für alle definierten CIM-Properties zurückgeliefert?
- Stimmt der zurückgelieferte Datentyp der CIM-Properties?
- Liegt der Wert des CIM-Properties in der Menge der gültigen Werte?
- Bildet eine Instanz einer CIM-Assoziation ein gültiges Assoziationspaar?
- (Wenn Bedingungen für CIM-Assoziationen vorliegen) Liegt für jede Bedingung mindestens ein Assoziationspaar vor?

Eine Freigabe für ein CIM-Profil erfolgt, wenn alle diese Tests gültig ausgewertet werden können.

Mit Hilfe der JavaCIMTestSuite können alle strukturellen Eigenschaften der CIM-Provider überprüft werden. Was die JavaCIMTestSuite (noch) nicht leisten kann ist:

- eine Semantiküberprüfung dahingehend, dass die zurückgelieferten Werte konform zum Laufzeitzustand des Systems passen. Wird beispielsweise ein Zustand eines Sensors auf die Zustände OK, Warning, Error hin überprüft, kann die JavaCIMTest-Suite beim Wert OK nicht validieren, ob der Sensor tatsächlich im Zustand OK ist. Dafür müsste ein Vergleichswert über eine weitere Verwaltungsschnittstelle ermittelt werden.
- dynamische Überprüfung von Szenarien. Beispielsweise die Überprüfung eines CIM-Property-Wertes, danach die Durchführung der Manipulation des Wertes im System und anschließende Überprüfung, ob der Wert sich entsprechend verändert hat.
- Überprüfung der intrinsischen Methoden, wie beispielsweise der Aufruf einer ModifyInstance-Methode auf einer Instanz zum Setzen von Verwaltungsinformationen.
- Ausführung und Überprüfung der korrekten Funktionsweise der extrinsichen Methoden (siehe SetLicenseKey(), der CIM-Klasse SVS iRMC Service Processor aus der Abbildung A.7).

Die Ausführung der möglichen Testfälle, die von der JavaCIMTestSuite durchgeführt werden können, sind für die Entwickler vorgesehen, bevor ein CIM-Profil zum Testen durch die Qualitätsabteilung freigegeben wird. Die weiteren Testfälle, die von der JavaCIMTestSuite nicht durchgeführt werden können, werden von der Qualitätsabteilung teils manuell, teils automatisiert mit weiteren Testwerkzeugen durchgeführt.

In der Abbildung A.9 wird ein Auszug aus der von der JavaCIMTestSuite erstellten Ausgabe dargestellt.

Executing class: SVS_iRMCServiceProcessor

Element	Test	Result
	Checking if provider is working	ResultOK
	Checking number of instances Expected: 1<=x<=1 Found: 1	ResultOK
Instance: (1/1) [Key] string CreationClassName = "SVS_iRMCServiceProcessor"; [Key] string Name = "IPMI Controller 4";		
Property: CreationClassName		
	is available?	ResultOK
	Check DataType Expected: "string" Returned: string	ResultOK
	Testing Value Expected value: "SVS_iRMCServiceProcessor" Returned value: "SVS_iRMCServiceProcessor"	ResultOK
Property: Dedicated		
	is available?	ResultOK
	Check DataType Expected: "uint16[]" Returned: uint16[]	ResultOK
	Testing Value Expected ARRAYALL: [28] Returned values: [28]	ResultOK
Property: ElementName		
	is available?	ResultOK
	Check DataType Expected: "string" Returned: string	ResultOK
	Testing Value Expected value: ".*" Returned value: "iRMC S4"	ResultOK
Property: EnabledState		
	is available?	ResultOK
	Check DataType Expected: "uint16" Returned: uint16	ResultOK
	Testing Value Expected value: "2" Returned value: 2	ResultOK
Property: HealthState		
	is available?	ResultOK
	Check DataType Expected: "uint16" Returned: uint16	ResultOK
	Testing Value Expected value is one of: [0, 5, 25] Returned value: 5	ResultOK
Property: IdentifyingDescriptions		
	is available?	ResultOK
	Check DataType Expected: "string[]" Returned: string	ResultFAIL
Property: LicenseKeyInformation		
	is available?	ResultFAIL

Abbildung A.9: *Auszug aus dem Testergebnis*

Abbildungsverzeichnis

Tabellenverzeichnis