Benutzerinteraktion in dienstorientierten Architekturen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Stefan Link

aus Karlsruhe

Tag der mündlichen Prüfung:	10. Juli 2009
Erster Gutachter:	Prof. Dr. Sebastian Abeck
Zweiter Gutachter:	Prof. Dr. Wilfried Juling

Für meine Familie

Die Kunst ist, einmal mehr aufzustehen,
als man umgeworfen wird.

Winston Churchill

Vorwort

Diese Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter in der Forschungsgruppe Cooperation & Management am Institut für Telematik der Universität Karlsruhe (TH). In dieser Zeit haben mich viele Personen unterstützt und auf meinem Weg begleitet, denen ich an dieser Stelle herzlich danken möchte.

Meinem Doktorvater Herrn Prof. Dr. Sebastian Abeck gilt mein besonderer Dank, da er mir stets den notwendigen Freiraum für meine Forschung gelassen und mir ein selbständiges wissenschaftliches Arbeiten ermöglicht hat. Seine kontinuierliche Bereitschaft, meine Fragestellungen in einer Vielzahl von Diskussionen zu erörtern und voranzutreiben, trug maßgeblich zum erfolgreichen Abschluss dieser Arbeit bei. Herrn Prof. Dr. Wilfried Juling danke ich vielmals für die Übernahme des Korreferats meiner Arbeit.

Der von mir nie als selbstverständlich erachtete Zusammenhalt unter den Kollegen der Forschungsgruppe bildete die Grundlage für ein wissenschaftliches Arbeiten in einer harmonischen und konstruktiven Atmosphäre. Dementsprechend danke ich meinen Kollegen Dr. Christian Emig, Michael Gebhart, Philip Hoyer, Dr. Karsten Krutz, Dr. Christian Mayerl, Dr. Oliver Mehl, Dr. Christof Momm und Ingo Pansa für ihren Teamgeist, ihre Hilfsbereitschaft und ihre tatkräftige Unterstützung.

Danken möchte ich ebenfalls allen Studierenden, deren Arbeiten ich während meiner Zeit in der Forschungsgruppe betreut habe. Gemeinsam haben wir viele wertvolle Ideen diskutiert, vertieft und umgesetzt. Stellvertretend für alle geht mein Dank an meine Diplomanden Pawel Gerr, Philip Hoyer, Fabian Jakobs, Christian Janz, Tilmann Kopp und Thomas Schuster.

Meiner Familie danke ich für das felsenfeste Vertrauen, die stete Förderung meiner Ausbildung und die uneingeschränkte Unterstützung bei der Verfolgung meiner persönlichen Ziele. Nicht zuletzt gilt mein ganz besonderer Dank meiner Frau Anja. Ihre Liebe und ihre Nachsicht gaben mir die notwendige Kraft und innere Ruhe, die zur Erstellung dieser Arbeit notwendig waren.

Karlsruhe, im August 2009 Stefan Link

Inhaltsverzeichnis

1 Einleitung 1

1.1 Motivation und Gegenstand 1

1.2 Betrachtetes Szenario 2

1.3 Problemstellungen 4

1.4 Zielsetzung und Beiträge dieser Arbeit 6

1.5 Prämissen der Arbeit 8

1.6 Aufbau der Arbeit 9

2 GRUNDLAGEN 11

2.1 Modellierung und modellgetriebene Softwareentwicklung 11

2.1.1 Modelle und Metamodelle 11

2.1.2 Modellgetriebene Softwareentwicklung 14

2.1.3 Modellgetriebene Architektur 16

2.1.4 Unified Modeling Language 21

2.2 Modellgetriebene Entwicklung von Geschäftsprozessen 25

2.2.1 Geschäftsprozesse und Workflows 25

2.2.2 Workflow-Management-Systeme 28

2.2.3 Workflow-Perspektiven und Workflow-Muster 33

2.2.4 Business-Driven Development 35

2.3 Grafische Benutzerschnittstellen 38

2.3.1 Einordnung der Benutzerschnittstelle 38

2.3.2 Modellierung grafischer Benutzerschnittstellen 39

2.4 Dienste und dienstorientierte Architekturen 41

2.4.1 Der Dienstbegriff 41

2.4.2 Dienstorientierte Architekturen 42

2.4.3 Benutzerschnittstellen für dienstorientierte Architekturen 45

2.5 Zusammenfassung 49

3 STAND DER FORSCHUNG 51

3.1 Anforderungen 51

3.1.1 Spezifikation der Benutzerinteraktion 51

3.1.2 Dienstorientierte Architektur zur Unterstützung der Benutzerinteraktion 53

3.1.3 Anforderungskatalog 55

3.2 Spezifikation von Benutzerinteraktion 55

3.2.1 Forschungsansätze aus dem Bereich der Softwaretechnik 55

3.2.2 Forschungsansätze aus dem Bereich des Web Engineering 65

3.3 Dienstorientierte Architektur zur Unterstützung der Benutzerinteraktion 73

3.3.1 Thomas et al.: User Tasks and Access Control over Web Services 73

3.3.2 Rodriguez et al.: Exploring Human Workflow Architectures 75

3.4 Zusammenfassung und Handlungsbedarf 76

4 METAMODELLE ZUR SPEZIFIKATION DER ASPEKTE DER BENUTZERINTERAKTION 79

4.1 Beiträge dieses Kapitels im Überblick 79

4.2 Integration der Benutzerinteraktion in Geschäftsprozesse 82

4.2.1 Konzeptionelles Metamodell für Geschäftsprozesse 82

4.2.2 Das Benutzeraktionsprofil 86

4.2.3 Anwendung des Benutzeraktionsprofils 88

4.3 Spezifikation der Benutzeraufgaben 89

4.3.1 Entwurf des Benutzeraufgabenmetamodells 90

4.3.2 Das Benutzeraufgabenprofil 100

4.3.3 Anwendung des Benutzeraufgabenprofils 104

4.4 Spezifikation der Benutzerschnittstellen 105

4.4.1 Entwurf der Metamodelle 106

4.4.2 Domänenprofil und Strukturprofil 115

4.4.3 Anwendung der Profile 118

4.5 Spezifikation der Dienste 121

4.5.1 Konzeption des Metamodells für Dienste 122

4.5.2 Das Dienstprofil 124

4.5.3 Anwendung des Dienstprofils 125

4.6 Resümee 126

5 AUTOMATISIERTE ERZEUGUNG DER SOFTWAREARTEFAKTE DER BENUTZERINTERAKTION 129

5.1 Beiträge des Kapitels im Überblick 129

5.2 Automatisierte Erzeugung der plattformunabhängigen Modelle 130

5.2.1 Einführung in das verfolgte Transformationsvorgehen 130

5.2.2 Erzeugung des Benutzeraufgabenmodells 133

5.2.3 Erzeugung des benutzerzentrischen Domänenmodells 137

5.2.4 Erzeugung des Strukturmodells 141

5.3 Dienstorientierte Architektur zur Unterstützung von Benutzerinteraktion 146

5.3.1 Erweiterung der dienstorientierten Architektur 147

5.3.2 Spezifikation von Benutzeraufgaben 150

5.3.3 Kommunikationsbeziehungen in der erweiterten dienstorientierten Architektur 152

5.4 Automatisierte Erzeugung der Benutzeraufgabenspezifikation 153

5.4.1 Ableitung des WS-HumanTask-Metamodells 154

5.4.2 Erzeugung des plattformspezifischen Benutzeraufgabenmodells 155

5.4.3 Erzeugung des WS-HumanTask-XML-Quellcode 158

5.5 Resümee 159

6 DEMONSTRATION DER TRAGFÄHIGKEIT 161

6.1 Beschreibung des Szenarios 161

6.1.1 Das Projekt „Karlsruher Integriertes InformationsManagement“ 161

6.1.2 Prüfungsverwaltung einer Hochschule 162

6.2 Modellgetriebene Entwicklung eines Geschäftsprozesses 162
6.2.1 Überblick über das Entwicklungsvorgehen 162
6.2.2 Vorbereitende Maßnahmen 164
6.2.3 Spezifikation des Geschäftsprozessmodells 165
6.2.4 Spezifikation des Workflow-Modells 167
6.2.5 Erzeugung der Benutzeraufgabenspezifikation 168
6.2.6 Erzeugung der Benutzerschnittstellenspezifikation 172
6.3 Umsetzung der Konzepte in der Industrie 180
6.4 Resümee 183

7 BEWERTUNG UND AUSBLICK 185
7.1 Beiträge der Arbeit 185
7.1.1 Metamodelle zur Spezifikation der Benutzerinteraktion 185
7.1.2 Transformation der Modelle der Benutzerinteraktion auf Softwareartefakte einer dienstorientierten Architektur 186
7.1.3 Tragfähigkeitsnachweise 186
7.2 Diskussion der Ergebnisse 187
7.2.1 Spezifikation der Benutzerinteraktion 187
7.2.2 Dienstorientierte Architektur zur Unterstützung von Benutzerinteraktion 190
7.2.3 Resümee 191
7.3 Ausblick 192
7.3.1 Navigationsmodelle für Benutzerschnittstellen 192
7.3.2 Überwachbarkeit der Benutzerinteraktion 193
7.3.3 Entwicklung der benötigten Dienste 195
7.3.4 Einsatz der Konzepte in unterschiedlichen Entwicklungsprozessen 196
7.3.5 Einsatz der entwickelten Konzepte zur Softwareindustrialisierung 196

ANHANG 199
A. Vollständige Transformationsspezifikationen 201
1. Transformation T3: Benutzerzentrisches Domänenmodell – Strukturmodell 201
2. Transformation T6: Strukturmodell – WebPart-Modell 202
3. Transformation T7: WebPart-Modell – WebPart-Spezifikation 207
B. Implementierung der Transformationen 209
1. Benötigte Anpassungen der Transformationen für MediniQVT 209
2. Transformation T1: Workflow-Modell – Benutzeraufgabenmodell 209
3. Transformation T3: Workflow-Modell – Benutzerzentrisches Domänenmodell 210
4. Transformation T6: Strukturmodell – WebPart-Modell 211
C. Abkürzungsverzeichnis 213
D. Abbildungsverzeichnis 217
E. Tabellenverzeichnis 221
F. Literaturverzeichnis 223

1 Einleitung

1.1 Motivation und Gegenstand

Eine Vielzahl unterschiedlicher Geschäftsziele lenkt und steuert heutige Unternehmen. Die zur Realisierung der Geschäftsziele notwendigen Schritte werden abstrakt als Geschäftsprozess bezeichnet. In einem Geschäftsprozess müssen unterschiedliche Einzelaktivitäten in koordinierter Reihenfolge durchgeführt werden, um eine Wertschöpfung für das den Geschäftsprozess ausführende Unternehmen zu erzielen. Die voranschreitende Globalisierung und die dadurch zunehmende Konkurrenzsituation drängt Unternehmen aller Branchen dazu, die eigenen Geschäftsprozesse kontinuierlich zu überdenken und an die aktuelle Marktsituation anzupassen [CB+04, PT+07]. Eine flexible und anpassbare Unterstützung der Geschäftsprozesse durch Informationstechnologie (IT) wird für Unternehmen zum kritischen Erfolgsfaktor [CB+06, WM06, Ri07].

Dienstorientierte Architekturen (engl. *Service-Oriented Architecture*, SOA) haben sich in jüngster Vergangenheit zunehmend als Paradigma für den Entwurf einer flexiblen IT-Unterstützung durchsetzen können [JB06]. Als Weiterentwicklung bestehender Softwarearchitekturen für verteilte Anwendungen verfolgen dienstorientierte Architekturen eine vereinfachte Integration und Wiederverwendung bestehender Anwendungen. Deren Fachfunktionalität wird durch standardisierte Schnittstellen gekapselt und in Form lose gekoppelter Dienste zur Verfügung gestellt [DJ+05]. Die als Dienstkomposition bezeichnete Verschaltung mehrerer Dienste gestattet ferner eine Implementierung von Geschäftsprozessen, die hinsichtlich Anforderungsänderungen flexibel ist [LR+02, Er05]. Gleichzeitig rücken dienstorientierte Architekturen durch die Dienstkomposition die zu unterstützenden Geschäftsprozesse der Unternehmen in den Mittelpunkt und ermöglichen eine enge Verzahnung zwischen den Geschäftsprozessen und der IT-Unterstützung [AL+07, KB+05].

Die Abbildung von Geschäftsprozessen auf eine dienstorientierte Architektur wird für vollautomatisierbare Geschäftsprozesse, die ohne die Interaktion eines Menschen ablaufen können, durch verschiedene Ansätze bereits betrachtet [EW+06, Ri07, BM+04]. Viele der tagtäglich in Unternehmen ausgeführten Geschäftsprozesse können jedoch nicht vollautomatisiert ausgeführt werden, sondern bedürfen der menschlichen Interaktion [DJ+05, AA+07a]. Die Interaktion des Menschen, der innerhalb eines Geschäftsprozesses als Nutzer der IT auftritt, kann dabei vielfältiger Natur sein und von einer einfachen Bestätigung einer Anfrage bis hin zur Erfassung komplexer Daten reichen. Um solche Geschäftsprozesse ebenfalls durch IT unterstützen zu können, muss folglich eine Brücke zwischen dem Menschen auf der einen Seite und der IT-Unterstützung auf der anderen Seite geschlagen werden. Dieses Ziel wurde bereits vor dem Aufkommen der dienstorientierten Architekturen durch die von der *Workflow Management Coalition* (WfMC) in Form einer Referenzarchitektur spezifizierten Workflow-Management-Systeme verfolgt [WfMC-WRM1.1]. Durch ein Workflow-Management-System kann der als Workflow bezeichnete automatisierbare Anteil eines Geschäftsprozesses definiert und zur Ausführung gebracht werden [RS04, CB+04, AH+00]. Ein besonderes Merkmal der Workflow-Management-Systeme liegt dabei in einer expliziten Unterstützung der menschlichen Interaktion. Historisch gesehen führte das Ziel, den Menschen bei der Ausführung eines komplexen Arbeitsvorganges zu unterstützen, überhaupt erst zur Entstehung der Workflow-Management-Systeme [DJ+05]. Das Referenzmodell der WfMC fasst deren gemeinsame Konzepte in einer Referenzarchitektur zusammen und spezifiziert unterschiedliche Komponenten, wie beispielsweise eine Benutzerschnittstelle (engl. *User Interface*) oder eine Aufgabenliste (engl. *Worklist*), die zur Unterstützung der Interaktion eines Menschen in einem Workflow notwendig sind.

Im Vergleich zu Workflow-Management-Systemen, die aufgrund der benötigten *Middleware* als schwergewichtige und wartungsintensive Lösung zur Unterstützung von Workflows anzusehen sind [AC+04], gestatten dienstorientierte Architekturen durch die Integration bestehender Fachfunktionalität über standardisierte Schnittstellen und durch die darauf aufbauende Komposition lose gekoppelter Dienste eine flexiblere Unterstützung von Geschäftsprozessen. Um diese flexible Unterstützung auch für Geschäftsprozesse, die der Interaktion eines Menschen bedürfen erreichen zu können, muss deren Abbildung auf eine dienstorientierte Architektur ermöglicht werden. Auf diese Weise kann ein zusätzlicher Teil der in Unternehmen ablaufenden Geschäftsprozesse flexibel durch IT unterstützt werden. Die Berücksichtigung der Interaktion eines Menschen stellt jedoch neue Anforderungen sowohl an das zur Abbildung der Geschäftsprozesse notwendige Entwicklungsvorgehen, als auch an die dienstorientierte Architektur als IT-Unterstützung der Geschäftsprozesse [SK+05]. Diese Anforderungen führen zu zusätzlichen Aktivitäten im Entwicklungsvorgehen, welche die Implementierung der benötigten Softwareartefakte zur Unterstützung von Benutzerinteraktion zum Ziel haben müssen. Ferner muss die dienstorientierte Architektur als Zielplattform des Entwicklungsvorgehens die Ausführung von Geschäftsprozessen mit Benutzerinteraktion ermöglichen. Die Unterstützung der Interaktion eines Menschen im Geschäftsprozess kann daher der Anwendung traditioneller Konzepte des Workflow-Managements auf eine dienstorientierte Architektur gleichgesetzt werden [TP+07].

Zusammenfassend sei festgehalten, dass die Abbildung von Geschäftsprozessen auf eine dienstorientierte Architektur unter Berücksichtigung der Anforderungen der Benutzerinteraktion untersucht werden muss. Diese Untersuchung ist Gegenstand der vorliegenden Arbeit.

1.2 Betrachtetes Szenario

Geschäftsprozesse werden in unterschiedlichsten Formen in Unternehmen ausgeführt und tragen zu deren Wertschöpfung bei. Oftmals sind diese Geschäftsprozesse jedoch nur implizit bekannt und werden auf Basis des Prozesswissens aller Beteiligten ausgeführt. Die Abbildung solcher Geschäftsprozesse auf eine vorhandene oder anzuschaffende IT-Unterstützung gestaltet sich aufgrund des nur schwer überschaubaren Gesamtkontextes als schwierig. Daher muss ein Geschäftsprozess zunächst in einer geeigneten Art und Weise formalisiert werden, um dann als Ausgangspunkt der Abbildung dienen zu können [SS+07a]. Der Geschäftsprozess wird somit zum zentralen Ausgangspunkt eines zur Abbildung benötigten methodischen Vorgehensmodells zur Softwareentwicklung. Diese Herangehensweise wird insbesondere durch den Ansatz der geschäftsgetriebenen Entwicklung (engl. *Business-Driven Development*, BDD) [KH+08] forciert. Die Motivation dieses Ansatzes liegt in der Tatsache begründet, dass eine hohe Flexibilität in der Anpassung von Geschäftsprozessen und eine rasche Umsetzung dieser Anpassungen auf die vorhandene IT-Unterstützung kritische Faktoren für den wirtschaftlichen Erfolg eines Unternehmens sind [BI+06]. Um diese Flexibilität auch auf der Ebene der Geschäftsprozesse zu ermöglichen, baut die geschäftsgetriebene Entwicklung im Kern auf den Konzepten der modellgetriebenen Softwareentwicklung (engl. *Model-Driven Software Development*, MDSD) auf und nutzt Modelle als Kernartefakte. Ein zentrales Ziel der modellgetriebenen Softwareentwicklung liegt in der Bewältigung der Komplexität heutiger IT-Systeme durch Abstraktion von komplexen Sachverhalten mittels Modellen [SV05]. Eine konkrete Ausprägung der modellgetriebenen Softwareentwicklung stellt die durch die *Object Management Group* (OMG) spezifizierte modellgetriebene Architektur (engl. *Model-Driven Architecture*, MDA) [OMG-MDA] dar, deren Konzepte in dieser Arbeit zum Einsatz kommen.

Wie eingangs erörtert, bietet eine dienstorientierte Architektur die notwendigen Voraussetzungen, um eine flexible IT-Unterstützung von Geschäftsprozessen zu ermöglichen. Dementsprechend soll im

betrachteten Szenario eine dienstorientierte Architektur als Ziel der Abbildung der Geschäftsprozesse dienen. Dienstorientierte Architekturen existieren dabei in unterschiedlichen, häufig auch als Reifegrade bezeichneten Entwicklungsstufen [RG08, AH06]. In der einfachsten Entwicklungsstufe einer dienstorientierten Architektur erfolgt lediglich die Anhebung der Kommunikationsbeziehungen zwischen den vorhandenen Anwendungen auf gemeinsame Schnittstellen und Kommunikationsprotokolle. Diese Anhebung entspricht im Wesentlichen den Herausforderungen der klassischen unternehmensweiten Anwendungsintegration (engl. *Enterprise Application Integration*, EAI) [CH+06]. In einer zweiten Entwicklungsstufe wird durch die Bereitstellung der Fachfunktionalität unterschiedlicher Anwendungen in Form von Diensten eine explizite Komposition dieser Fachfunktionalität über die Grenzen verschiedener Anwendungen hinweg ermöglicht. Auf dieser technisch motivierten Integration aufbauend kann durch die Berücksichtigung zusätzlicher Anforderungen, wie langlaufende Prozesse oder Benutzerinteraktion, in einer dritten Entwicklungsstufe eine unmittelbare Abbildung von Geschäftsprozessen auf Dienstkompositionen erreicht werden.

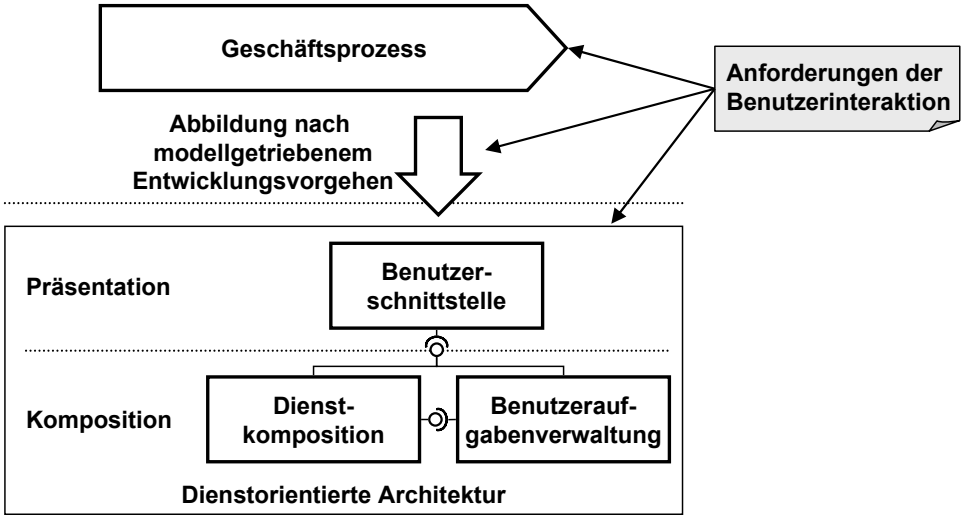

Abbildung 1: Betrachtetes Szenario

Um den Fokus weiter einschränken zu können, wird im betrachteten Szenario ferner davon ausgegangen, dass bestehende Anwendungen bereits in eine unternehmensinterne, dienstorientierte Architektur integriert wurden und die geschäftsrelevante Fachfunktionalität dieser Anwendungen in Form von Diensten angeboten wird. Um diese Dienste im Kontext der Abbildung von Geschäftsprozessen mit Benutzerinteraktion nutzen zu können, sieht die im Szenario betrachtete dienstorientierte Architektur neben der Dienstkomposition zwei weitere zentrale Dienste vor. Die in der logischen Kompositionsschicht der dienstorientierten Architektur angeordnete Benutzeraufgabenverwaltung übernimmt in Anlehnung an die in Abschnitt 1.1 beleuchteten Workflow-Management-Systeme die Steuerung und Überwachung der Interaktion eines Menschen in einem Geschäftsprozess. Auf der Kompositionsschicht aufbauend bildet die Präsentationsschicht die oberste logische Schicht der betrachteten dienstorientierten Architektur. Über die hier angeordnete Benutzerschnittstelle, die in dieser Arbeit ebenfalls als Dienst aufgefasst wird, kann ein Mensch innerhalb eines Geschäftsprozesses mit der IT-Unterstützung in Interaktion treten. Die Benutzerschnittstelle, die Benutzeraufgabenverwaltung und die Dienstkomposition bilden somit im Szenario der Arbeit die drei zentralen Komponenten der dienstorientierten Architektur zur Unterstützung von Geschäftsprozessen mit Benutzerinteraktion.

Im Fokus dieses Szenarios soll der Gegenstand der Arbeit detailliert betrachtet werden. Es ist somit zu untersuchen, wie eine Abbildung von Geschäftsprozessen mit Benutzerinteraktion auf eine dienstori-

entierte Architektur unter der Verwendung der Konzepte der modellgetrieben Architektur erreicht werden kann.

1.3 Problemstellungen

Bei der Abbildung von Geschäftsprozessen auf die IT treten unter Berücksichtigung der Benutzerinteraktion die folgenden Problemstellungen auf.

Beachtung der spezifischen Anforderungen der Benutzerinteraktion

Im Kontext der Spezifikation von Geschäftsprozessen und deren Abbildung auf die IT kommen unterschiedlichste Sprachen der Informatik zum Einsatz. Die in formalisierten Geschäftsprozessen zu verankernden Geschäftsanforderungen können in Bezug auf die Benutzerinteraktion durch bestehende Spezifikationssprachen jedoch nur ungenügend erfasst und durch vorhandene Ausführungssprachen für Geschäftsprozesse nicht oder nur sehr unzureichend umgesetzt werden [WA+06, KK+05, RA+06a]. Obwohl bei den an einem Entwicklungsvorgehen beteiligten Entwicklungsrollen ein umfassendes Wissen über die spezifischen Anforderungen der Benutzerinteraktion von der Integration des Menschen in einen Geschäftsprozess bis hin zur benötigten Benutzerschnittstelle vorhanden ist, kann dieses Wissen durch bestehende Spezifikationssprachen nicht formalisiert werden [LV+05, PP02, KV06, SK05]. Die dennoch notwendige Umsetzung dieser Anforderungen resultiert in einem hohen manuellen Entwicklungs- und Konfigurationsaufwand und verhindert eine flexible IT-Unterstützung. Daher bedarf es formaler, ausdrucksstarker Spezifikations- und Ausführungssprachen, um die **spezifischen Anforderungen der Benutzerinteraktion** in einem modellgetriebenen Entwicklungsvorgehen berücksichtigen zu können (vgl. auch [HV06]).

Nachvollziehbarkeit im Rahmen der Entwicklung

Die Berücksichtigung der spezifischen Anforderungen der Benutzerinteraktion erfordert zusätzliche Aktivitäten zur Entwicklung der benötigten Softwareartefakte. Diese Aktivitäten werden von bestehenden Ansätzen vom eigentlichen Entwicklungsvorgehen entkoppelt und isoliert für jeden einzelnen Aspekt der Benutzerinteraktion betrachtet [SP03, BJ04, Bi06, SK+05]. Ergibt sich nun eine Änderung der Spezifikation des übergeordneten Geschäftsprozesses, so muss diese Änderung durch das Entwicklungsvorgehen auch auf die verschiedenen Softwareartefakte der Benutzerinteraktion abgebildet werden. Um hinsichtlich solcher Änderungen flexibel reagieren zu können, müssen die an der Entwicklung beteiligten Entwicklungsrollen wissen, welche Auswirkungen die geänderten Anforderungen mit sich bringen und welche Softwareartefakte von einer Änderung betroffen sind [SS+07b, LV05]. Das verwendete Entwicklungsvorgehen muss demnach stets eine **Nachvollziehbarkeit** (engl. *Traceability*) [HT06] zwischen Anforderungen und Implementierung gewährleisten [KH+08, LC+03]. Durch die entkoppelten Entwicklungsaktivitäten im Kontext der Benutzerinteraktion kann diese Nachvollziehbarkeit jedoch nur erschwert durch einen hohen manuellen Entwicklungsaufwand erreicht werden, welcher einer flexiblen IT-Unterstützung für Geschäftsprozesse mit Benutzerinteraktion im Wege steht [Bi06, WF+05, Ko01].

Verifikation der Anforderungen der Benutzerinteraktion

Die kontinuierliche Evolution der Anforderungen der Unternehmen bedingt eine fortwährende Anpassung der Geschäftsprozessspezifikationen [CB+04]. Wird vor der Abbildung eines Geschäftsprozesses auf die IT-Unterstützung die Korrektheit der geänderten Spezifikation hinsichtlich der neuen Anforderungen nicht verifiziert, so kann eine fehlerhafte Abbildung des Geschäftsprozesses die Folge sein

[So06, Ba00]. Im Kontext der Benutzerinteraktion stellt eine nachgelagerte Verifikation ein besonders kritisches Problem dar [SS+07a, SC+06]. Da die Benutzerschnittstelle den direkten Zugriff des Menschen auf die IT-Unterstützung ermöglicht, kommt ein gemeinsames Verständnis über die Anforderungen zwischen Auftraggeber und Entwickler häufig erst über die Benutzerschnittstelle oder die zugeordneten Benutzeraufgaben zustande [RF+04, NF00]. Das Entwicklungsvorgehen muss daher eine **frühzeitige Verifikation** der Umsetzung der Anforderungen der Benutzerinteraktion berücksichtigen, da eine nachgelagerte Verifikation dieser Anforderungen im schlimmsten Fall zu einem Neubeginn des gesamten Entwicklungsvorgehens führen kann [CC03].

Umgang mit Heterogenität

Mit vorhandenen herstellerspezifischen Entwicklungsumgebungen kann die Abbildung von Geschäftsprozessen mit Benutzerinteraktion bereits erreicht werden. Neben einem häufig nicht transparent nachvollziehbaren Entwicklungsvorgehen (vgl. vorangegangene Problemstellung) sind die mit herstellerspezifischen Entwicklungsumgebungen realisierten Lösungen jedoch hochgradig plattformspezifisch und können nur unter erheblichem Aufwand für weitere Plattformen adaptiert oder in diese integriert werden [Bi06, FR07]. Derartige Lösungen können demnach nicht zur Bewältigung der heute vorhandenen Heterogenität der IT-Systeme eingesetzt werden, die im Kontext der Benutzerinteraktion durch eine Vielzahl an bestehenden Plattformen und Endgeräten besonders ausgeprägt ist [PS+08, AH+03, Co03, LC+03]. Folglich muss zur Erreichung einer flexiblen IT-Unterstützung der Plattformunabhängigkeit und der daraus resultierenden **Portabilität** der entwickelten Lösungen ein zentraler Stellenwert beigemessen werden [PM06].

Komplexitätssteigerung durch Berücksichtigung der Benutzerinteraktion

Die Berücksichtigung der Aspekte der Benutzerinteraktion führt zu neuen Anforderungen an das Entwicklungsvorgehen und die beteiligten Entwicklungsrollen und hat damit eine erhöhte Komplexität des gesamten Entwicklungsprojektes zur Folge [PS+08]. Neben der initialen Spezifikation der Anforderungen der Benutzerinteraktion muss deren Abbildung auf die IT-Unterstützung in Form unterschiedlicher Softwareartefakte bewerkstelligt werden. Hierbei muss die Konsistenz und Nachhaltigkeit aller Softwareartefakte über das gesamte Entwicklungsvorgehen hinweg sichergestellt werden. Dieser Notwendigkeit wirken die ständige Evolution der Anforderungen und die Verwendung unterschiedlicher Entwicklungsumgebungen, die gegebenenfalls zusätzlich von verschiedenen Herstellern stammen können, entgegen. Das verwendete Entwicklungsvorgehen muss daher dieser Komplexitätssteigerung durch geeignete **Mechanismen zur Reduktion der Komplexität** begegnen.

Sicherstellung der Interoperabilität

Eine dienstorientierte Architektur bietet die Fachfunktionalität bestehender IT-Systeme in Form von Diensten über standardisierte Schnittstellen an und stellt auf diese Weise die Interoperabilität zwischen den integrierten IT-Systemen her [SV05]. Zur Berücksichtigung der Benutzerinteraktion muss eine dienstorientierte Architektur zusätzliche Fachfunktionalität wie etwa eine Benutzerschnittstelle oder eine Benutzeraufgabenverwaltung vorsehen. Diese Fachfunktionalität wird von bestehenden Ansätzen jedoch nicht interoperabel in Form lose gekoppelter Dienste, sondern als jeweils herstellerspezifische Implementierung bereitgestellt [AC+05]. Herstellerspezifische Lösungen, die nicht konform zu bestehenden Standards implementiert werden, verhindern eine durchgängige **Interoperabilität** und lose Kopplung zwischen den integrierten IT-Systemen [TP+07].

1.4 Zielsetzung und Beiträge dieser Arbeit

Eine flexible IT-Unterstützung der Geschäftsprozesse ist ein kritischer Erfolgsfaktor für Unternehmen. Geänderte Spezifikationen der Geschäftsprozesse müssen sich daher unmittelbar auf die IT-Unterstützung abbilden lassen. Eine derartige Abbildung muss auch für diejenigen Geschäftsprozesse erreicht werden, die der Interaktion des Menschen bedürfen. Zur Überwindung der zuvor identifizierten Problemstellungen wird ein integrierter Ansatz benötigt.

Ziel der vorliegenden Arbeit ist es daher, bestehende modellgetriebene Verfahren dahin gehend zu ergänzen, dass die spezifischen Aspekte der Benutzerinteraktion zielgerichtet und nachvollziehbar im Rahmen der Entwicklung beachtet und die entstehenden Softwareartefakte durch eine geeignete dienstorientierte Architektur zur Ausführung gebracht werden können. Hierbei müssen die verwendeten Spezifikationen von bestehenden hersteller- und plattformspezifischen Lösungen abstrahieren, um die Portabilität der entwickelten Lösungen sicherstellen zu können. Ferner müssen diese Lösungen **interoperabel** in Bezug zu der weiteren Fachfunktionalität der dienstorientierten Architektur bereitgestellt werden.

Um dieses Ziel zu erreichen, basieren die entwickelten Beiträge dieser Arbeit auf den Prinzipien der modellgetriebenen Softwareentwicklung, wie sie in Abschnitt 2.1 dieser Arbeit detailliert eingeführt werden. Hierbei orientiert sich diese Arbeit an dem durch die *Object Management Group* (OMG) bereitgestellten und als modellgetriebene Architektur (*Model-Driven Architecture*, MDA) [OMG-MDA] bezeichneten Rahmenwerk zur modellgetriebenen Entwicklung. Durch das MDA-Prinzip der **plattformunabhängigen Abstraktion** [KW+03] kann einerseits die Portabilität der entstehenden Lösungen sichergestellt werden und andererseits der durch die zusätzliche Berücksichtigung der Benutzerinteraktion entstehenden Komplexitätssteigerung begegnet werden. Zur weiteren Reduzierung der Komplexität sieht diese Arbeit eine **Automatisierung** bei der Erzeugung der benötigten Softwareartefakte vor. Die automatisierte Erzeugung ermöglicht ebenso eine frühzeitige Verifikation der Anforderungen der Benutzerinteraktion. Dieser grundsätzlichen Herangehensweise folgend wird das Ziel dieser Arbeit durch die in Abbildung 2 dargestellten Beiträge erreicht. Diese werden im Folgenden in zusammengefasster Form eingeführt und in Kapitel 4 und 5 im Detail vorgestellt.

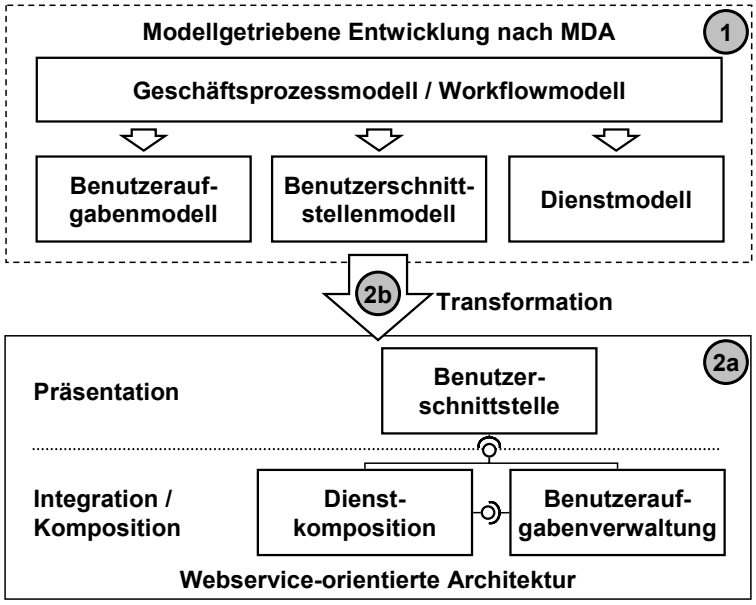

Abbildung 2: Beiträge der Arbeit im Überblick

Beitrag B1: Metamodelle zur Spezifikation der Benutzerinteraktion

Im ersten Beitrag dieser Arbeit erfolgt die Konzeption von plattformunabhängigen Metamodellen, die eine präzise Spezifikation der Anforderungen der Benutzerinteraktion ermöglichen. Die Beachtung bereits existierender Metamodelle ermöglicht die Integration der neu bereitgestellten Modellelemente in bestehende modellgetriebene Verfahren. Dementsprechend folgt das präsentierte Entwicklungsvorgehen der Modellhierarchie der modellgetriebenen Architektur (engl. *Model-Driven Architecture*, MDA) und den Konzepten der geschäftsgetriebenen Entwicklung (engl. *Business-Driven Development*, BDD) und nutzt ein plattformunabhängiges Modell eines Geschäftsprozesses als Ausgangspunkt der Entwicklung, in dem erste Anforderungen der Benutzerinteraktion verankert werden. Aus dem plattformunabhängigen Modell eines Geschäftsprozesses werden anschließend weitere plattformunabhängige Modelle, wie beispielsweise das Benutzeraufgabenmodell, das im Zusammenhang mit dem Workflow-Modell eine präzise Spezifikation des Einsatzes menschlicher Ressourcen gestattet, oder das Benutzerschnittstellenmodell, das zu einer detaillierten Spezifikation der Benutzerschnittstelle dient, abgeleitet. Diese plattformunabhängigen Modelle erlauben eine von technischen Rahmenbedingungen abstrahierende Spezifikation der Anforderungen der Benutzerinteraktion. Die daraus resultierende Portabilität und Wiederverwendbarkeit der Modelle für beliebige Zielplattformen wird der heute vorhandenen Vielfalt der technischen Systeme gerecht. Durch den Bezug zum Geschäftsprozess ist eine Nachvollziehbarkeit und Konsistenz der entwickelten Modelle von der Ausgangsspezifikation an gegeben. Als zentraler Mehrwert dieses Beitrags wird eine integrierte Betrachtung der spezifischen Anforderungen der Benutzerinteraktion in einem modellgetriebenen Entwicklungsvorgehen ermöglicht.

Beitrag B2: Transformation der Modelle der Benutzerinteraktion auf Softwareartefakte einer dienstorientierten Architektur

Der zweite Beitrag dieser Arbeit befasst sich mit der vollständig automatisierten Abbildung der im ersten Beitrag vorgestellten Modelle. Ein plattformunabhängiges Modell eines abzubildenden Geschäftsprozesses mit Benutzerinteraktion dient als Ausgangspunkt; lauffähige Softwareartefakte für eine dienstorientierte Architektur sind das Ziel des modellgetriebenen Entwicklungsvorgehens. Hierzu wird im zweiten Beitrag zunächst eine dienstorientierte Architektur konzipiert, die den betrachteten Anforderungen der Benutzerinteraktion genügt und die zusätzlich benötigte Fachfunktionalität interoperabel zur Verfügung stellt. Zu dieser Konzeption greift die Arbeit auf bestehende Standards zurück, wodurch eine Allgemeingültigkeit und Wiederverwendbarkeit der im Kern des zweiten Beitrags spezifizierten Transformationen gewährleistet ist. Diese ermöglichen eine vollständig automatisierte Abbildung der entwickelten Modelle auf lauffähige Softwareartefakte für die konzipierte dienstorientierte Architektur, wodurch einerseits eine frühzeitige Verifikation der Anforderungen der Benutzerinteraktion und andererseits eine Reduktion der Komplexität des Entwicklungsvorgehens erreicht wird. Die Transformationen bilden somit den wesentlichen Mehrwert dieses zweiten Beitrags.

Demonstration der Tragfähigkeit

Die Tragfähigkeit der in der Arbeit entwickelten Konzepte wurde anhand der Entwicklung der IT-Unterstützung eines Geschäftsprozesses aus dem Szenario der Aus- und Weiterbildung demonstriert (siehe dazu Kapitel 6). Ausgehend von einem plattformunabhängigen Modell dieses Geschäftsprozesses konnten die Anforderungen der Benutzerinteraktion durch die zur Verfügung gestellten Modelle erfasst und durch vollständig automatisierte Transformationen auf ausführbare Implementierungen für die konzipierte dienstorientierte Architektur überführt werden. Die Demonstration der Anwendbarkeit der entwickelten Konzepte in der Praxis erfolgte durch Implementierungen im Kontext von For-

schungs- und Industrieprojekten, unter anderem im Projekt "Karlsruher Integriertes InformationsManagement" (KIM).

1.5 Prämissen der Arbeit

Sowohl die modellgetriebene Softwareentwicklung, die Modellierung von Geschäftsprozessen und Anforderungen der Benutzerinteraktion als auch die dienstorientierte Architektur stellen separiert betrachtet umfangreiche Forschungsgebiete dar. Daher sollen für diese Arbeit einige Einschränkungen vorgenommen und Annahmen getroffen werden.

Prämisse P1: Grafische Benutzerschnittstelle

Dem menschlichen Benutzer steht heute eine Vielzahl an Benutzerschnittstellen zur Verfügung. Diese können grafischer, akustischer oder haptischer Natur sein. Als einfache Beispiele seien ein Bildschirm, eine Sprachsteuerung oder das Lenkrad eines Autos genannt. Eine aus diesen Klassen kombinierte Benutzerschnittstelle ist bei einem modernen Mobiltelefon (grafisch, akustisch, haptisch) oder einem Touchscreen (grafisch, haptisch) vorhanden. Die Betrachtung aller Klassen von Benutzerschnittstellen sprengt den Rahmen dieser Arbeit. Aus diesem Grund stehen rein grafische Benutzerschnittstellen im Mittelpunkt, die im Folgenden der Einfachheit halber kurz als Benutzerschnittstelle bezeichnet werden. Zusätzlich soll der Fokus dieser Arbeit auf formularbasierten Benutzerschnittstellen liegen, wie sie beispielsweise bei Webanwendungen, Datenbanksystemen, ERP- oder CRM-Systemen, nicht aber bei Spielen oder Simulatoren zum Einsatz kommen.

Prämisse P2: Funktionale Betrachtung der Benutzerschnittstelle

Eine Benutzerschnittstelle erlaubt einem menschlichen Benutzer, mit einem IT-System zu interagieren. Um diese Interaktion für den Menschen optimal unterstützen zu können, beschäftigt sich der Forschungsbereich der Softwareergonomie (engl. *Usability Engineering*) [RF07, St07] unter anderem mit Fragen der Anthropotechnik und Ergonomie im Kontext von Benutzerschnittstellen. Zur Einschränkung der Komplexität werden diese Aspekte durch diese Arbeit nicht erörtert, sondern eine funktionale Betrachtung der Benutzerschnittstelle fokussiert. Da diese jedoch die Grundlage für weitere Betrachtungen hinsichtlich der Ergonomie und Benutzerfreundlichkeit legt, werden diese Aspekte im Ausblick dieser Arbeit wieder aufgegriffen.

Prämisse P3: Unternehmensinterne Architektur

Geschäftsprozesse können sich in unternehmensübergreifenden Formen der Zusammenarbeit (engl. *Business to Business*, B2B) über mehrere Unternehmensgrenzen hinweg erstrecken. Die durch Geschäftsprozesse gesteuerte, föderative Kooperation mehrerer Unternehmen führt zu technischen, organisatorischen sowie rechtlichen Randbedingungen, die in dieser Arbeit nicht betrachtet werden. So stellt beispielsweise die unternehmensinterne Manipulation von personenbezogenen Daten im Gegensatz zur unternehmensübergreifenden Nutzung einen rechtlich differenziert zu betrachtenden Sachverhalt dar.

Prämisse P4: Webservice-basierte dienstorientierte Architektur

In Theorie und Praxis bestehen sowohl unterschiedliche Ausprägungen als auch Auffassungen über dienstorientierte Architekturen. Diese Arbeit betrachtet eine dienstorientierte Architektur als eine flexible und adaptive Facharchitektur, die Fachfunktionalität hinter standardisierten Schnittstellen kapselt. Webservices haben sich als vielversprechende und daher häufig genutzte Implementierungs-

form dieser Schnittstellen etabliert. Sowohl die angestrebte Plattform- und damit Herstellerunabhängigkeit als auch die Einfachheit der technischen Realisierung haben hierzu beigetragen [DJ+05]. Daher kommt in dieser Arbeit eine Webservice-basierte dienstorientierte Architektur als konkrete Ausprägung zum Einsatz. Dennoch können viele Ergebnisse dieser Arbeit auch auf andere Ausprägungsformen (CORBA, DCOM, Jini etc.) übertragen werden.

1.6 Aufbau der Arbeit

Die Arbeit ist wie in Abbildung 3 dargestellt in die folgenden Kapitel gegliedert. **Kapitel 1** motiviert über die Notwendigkeit einer flexiblen IT-Unterstützung für Geschäftsprozesse mit Benutzerinteraktion den Gegenstand der Arbeit. Unter Betrachtung eines konkreten Szenarios und der hier bestehenden Problemstellungen leiten sich das Ziel und die Beiträge dieser Arbeit ab. Zur präzisen Einordnung der Ziele werden die Prämissen dieser Arbeit definiert. Im Anschluss führt **Kapitel 2** alle zum Verständnis der Arbeit notwendigen Begriffe und Grundlagen ein. **Kapitel 3** diskutiert den Stand der Forschung im aufgezeigten Themenbereich. Diese Diskussion spiegelt sich an einem Anforderungskatalog, anhand dessen Defizite bestehender Ansätze herausgearbeitet werden. Der Handlungsbedarf dieser Arbeit wird durch diese Defizite motiviert und entlang der folgenden Kapitel bearbeitet.

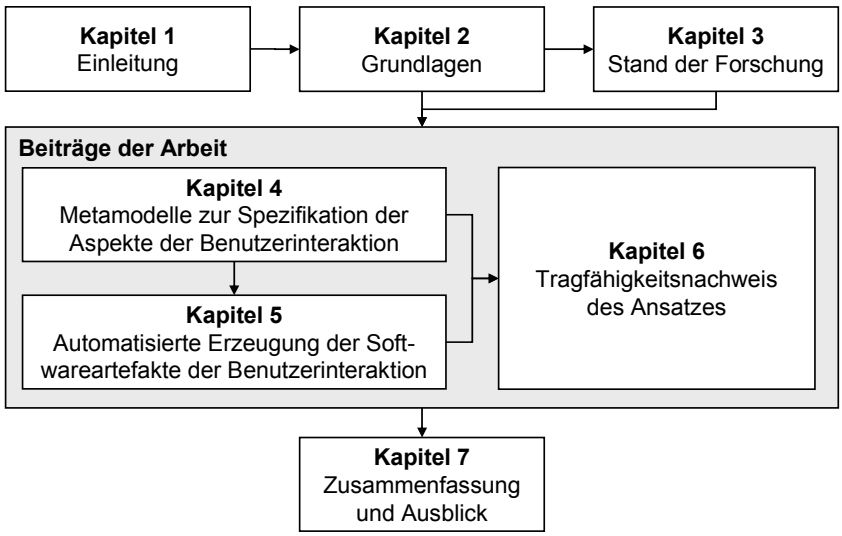

Abbildung 3: Gliederung der Arbeit im Überblick

In der durch die beiden Beiträge der Arbeit vorgegebenen Reihenfolge präsentiert **Kapitel 4** die entwickelten Metamodelle zur Spezifikation der Anforderungen der Benutzerinteraktion. **Kapitel 5** konzipiert eine dienstorientierte Architektur zur Unterstützung von Benutzerinteraktion als Zielplattform des modellgetriebenen Entwicklungsvorgehens und stellt Transformationen für eine automatisierte Abbildung von Geschäftsprozessen auf diese Zielplattform bereit. Um die Machbarkeit und den Mehrwert der Beiträge dieser Arbeit zu demonstrieren, liefert **Kapitel 6** einen Tragfähigkeitsnachweis anhand konkreter Implementierungsszenarien. **Kapitel 7** schließt die Arbeit mit einer Bewertung des eigenen Ansatzes sowie einem Ausblick auf weiterführende wissenschaftliche Fragestellungen ab.

2 Grundlagen

In diesem Kapitel werden die für das weitere Verständnis dieser Arbeit notwendigen Grundlagen, Konzepte und Begriffe eingeführt. Hierbei stehen zunächst die Modellierung, die Metamodellierung und die modellgetriebene Softwareentwicklung im Mittelpunkt. Diese Konzepte werden anschließend für die Modellierung von Geschäftsprozessen und Workflows ausgeprägt, wobei ein besonderer Schwerpunkt auf die Interaktion des Menschen in einem Geschäftsprozess gelegt wird. Mit dem gleichen Schwerpunkt erfolgt im Anschluss die Betrachtung dienstorientierter Architekturen als flexible IT-Unterstützung für Geschäftsprozesse.

2.1 Modellierung und modellgetriebene Softwareentwicklung

Das übergeordnete Ziel dieser Arbeit ist es, Geschäftsprozesse unter Berücksichtigung der Benutzerinteraktion auf eine dienstorientierte Architektur abzubilden. Um die dabei entstehenden Konzepte und Artefakte im Sinne der Informatik korrekt erfassen zu können, bedarf es deren Formalisierung. Modelle spielen hierbei eine wesentliche Rolle, weshalb im folgenden Abschnitt der Begriff des Modells sowie das Konzept der Metamodellierung eingeführt werden.

2.1.1 Modelle und Metamodelle

Modelle gestatten es, abstrakte oder reale Sachverhalte und Objekte aus unterschiedlichen Blickwinkeln darzustellen und gleichzeitig von gewissen Eigenschaften der Ausgangsobjekte zu abstrahieren [Go95]. Die einem Ausgangsobjekt zugrunde liegende Komplexität kann durch ein Modell beherrschbar gemacht werden, da ein Modell nicht alle Eigenschaften des Ausgangsobjektes wiedergeben muss [PM06]. Das Modell kann diese auf gewisse Perspektiven reduzieren. Nach [SV05] ist ein **Modell**

> *„eine abstrakte Repräsentation von Struktur, Funktion oder Verhalten eines Systems.“*

Im Handbuch der Softwarearchitektur [RH06] führt [Uh06] als ein typisches Beispiel für ein Modell eine Landkarte an. Landkarten zeigen eine abstrahierte Sicht der Oberfläche unseres Planeten. Der Fokus kann dabei auf politischen Grenzen, physischen Aspekten wie der Topologie einer Region oder dem Verlauf von Verkehrswegen liegen. Jede Karte abstrahiert somit von der Realität unter einer bestimmten Perspektive. Folglich weist ein Modell nach [St73] und [Uh06] drei wesentliche Eigenschaften auf:

- **Abbildungstreue.** Die Beziehung zwischen realem oder abstraktem Objekt und dessen Repräsentation im Modell wird durch eine Abbildung (im mathematischen Sinne) hergestellt. Zu einem Objekt können mehrere Abbildungen für unterschiedliche Perspektiven existieren.

- **Abstraktion.** Ein Modell repräsentiert ein Objekt und beschränkt sich dabei auf bestimmte Aspekte, die für den gedachten Zweck des Modells relevant sind. Es abstrahiert somit von Aspekten, die für das Modell nicht von Bedeutung sind.

- **Pragmatik.** Ein Modell wird zu einem bestimmten Zweck angefertigt und ist daher pragmatisch. Der Zweck bestimmt neben der Abbildung zwischen Objekt und Modell auch die durch das Modell vorgenommene Abstraktion.

Modelle im oben beschriebenen Sinne sind ein zentraler Bestandteil dieser Arbeit. Die Konzepte der geschäftsgetriebenen Entwicklung, der Benutzerinteraktion und der dienstorientierten Architektur können jeweils durch Modelle formalisiert und damit im Sinne der Informatik nutzbar gemacht werden. Diese entsprechend benötigten Modelle werden im weiteren Verlauf der Arbeit schrittweise eingeführt.

Je nach Pragmatik eines Modells sind nur gewisse Objekte sinnvoll in einem Modell abbildbar. Im angeführten Beispiel der Landkarte wäre es verwunderlich, wenn beispielsweise Moleküle und deren Atome in einer Landkarte abgebildet wären. Ein Modell entspringt daher immer einem als **Domäne** bezeichneten Wissens- oder Fachgebiet, dessen Konzepte durch eine geeignete Sprache formalisiert werden sollen. Die Struktur einer Domäne, also das Wissen, wie ein Modell aufzubauen ist, wird durch ein Metamodell vorgegeben. Ein derartiges Metamodell ist für viele Fach- und Wissensgebiete bei einem Menschen implizit vorhanden. Ab einem gewissen Alter kennt ein Mensch in der Regel das Metamodell einer Landkarte und weiß daher, dass ein Atom oder Molekül nicht zur Domäne der Landkarte gehört. Um dieses Wissen über eine Domäne und deren Struktur für ein IT-System verfügbar zu machen, muss das entsprechende Metamodell formalisiert werden. Mit dessen Hilfe können dann Modelle erstellt und auf ihre Korrektheit hinsichtlich der Verwendung der Modellelemente überprüft werden. [Uh06] definiert ein **Metamodell** als

> *„ein Modell, das eine Menge anderer Modelle definiert, die als Instanzen des Metamodells bezeichnet werden.“*

Somit stellt jedes Modell die Instanz eines Metamodells dar, das wiederum selbst als Modell zu verstehen ist. Diese Hierarchiebildung kann theoretisch beliebig fortgesetzt werden, da auch ein Metamodell wiederum ein Modell und damit eine Instanz eines weiteren Metamodells ist. Wie Abschnitt 2.1.4 näher ausführt, hat sich in der Praxis jedoch gezeigt, dass ab einer bestimmten Metaebene Modelle als selbstdefinierend spezifiziert werden sollten und eine Beschränkung auf eine feste Anzahl an Metaebenen sinnvoll ist.

Ein Metamodell als Bauplan für Modelle legt die Menge der verfügbaren Modellelemente mittels einer **abstrakten Syntax** fest. Die abstrakte Syntax trifft jedoch noch keine Aussagen hinsichtlich einer Repräsentation der spezifizierten Konzepte. Hierzu bedarf es einer **konkreten Syntax**. Eine abstrakte Syntax kann dabei durch mehrere konkrete Syntaxen repräsentiert werden (vgl. dazu Abbildung 4). Zusätzlich verfügt ein Metamodell über eine **statische Semantik**, die auf Basis der abstrakten Syntax gebildet wird und die Wohlgeformtheitskriterien einer Sprache bestimmt [SV05]. Das Attribut „statisch“ drückt dabei aus, dass die spezifizierten Eigenschaften bereits zur Entwurfszeit überprüft werden können, ohne das Modell zur Laufzeit betrachten zu müssen [BB+08]. Die abstrakte Syntax des Landkartenbeispiels kann unter anderem die Elemente `Straße` und `Stadt` beinhalten. Die Repräsentation dieser Elemente kann je nach konkreter Syntax unterschiedlich sein. Manche Landkarten stellen eine Stadt als Punkt, andere wiederum als Quadrat dar. Die statische Semantik des Metamodells der Landkarte legt beispielsweise fest, dass eine Stadt eine Mindesteinwohnerzahl haben muss, um als Großstadt und nicht als Stadt oder Kleinstadt im Modell erfasst zu werden. Alle modellierbaren Elemente erhalten durch die **dynamische Semantik** letztendlich ihre eigentliche Bedeutung. Letztere kann im Gegensatz zur statischen Semantik nicht zur Entwurfszeit ausgewertet werden, da sich die spezifizierten Regeln auf die „Ausführung“ des Modells beziehen. Wie in [BB+08] angeführt, würde dadurch bei höheren Programmiersprachen beispielsweise definiert, was ein Programm zur Laufzeit tut, um somit die eigentliche Absicht der Modelle überprüfen zu können. Die dynamische Semantik kann einerseits durch eine gut dokumentierte, textuelle Beschreibung spezifiziert werden. Andererseits können Sprachen mit einer formalen Semantik hinzugezogen werden, um die Bedeutung

der Modellelemente durch die Spezifikation einer Abbildung auf diese Sprache zu definieren. Die Funktionsweise eines Programms kann so beispielsweise als eine schrittweise Zustandsänderung einer abstrakten Maschine aufgefasst und die dynamische Semantik dieses Programms durch eine Abbildung auf endliche Automaten oder Petri-Netze formal festgelegt werden.

Um den Bezug zwischen einem Metamodell, den Modellen als Instanzen des Metamodells und den durch die Metamodelle repräsentierten Domänen herzustellen, findet sich in der Literatur häufig der Begriff der **domänenspezifischen Sprache** (engl. *Domain Specific Language,* DSL) [DK+00]. Domänenspezifische Sprachen werden in der Informatik in unterschiedlichsten Teildisziplinen verwendet, um die Schlüsselaspekte einer Domäne formalisierbar zu machen [SV05]. Als bekannte Beispiele für domänenspezifische Sprachen seien an dieser Stelle in Anlehnung an [PT+07] die in der Informatik häufig zum Einsatz kommenden regulären Ausdrücke zur Beschreibung von Zeichenketten oder Datenbanksprachen, wie etwa SQL (*Structured Query Language*), erwähnt. Im Kontext der Modellierungssprachen umfasst eine domänenspezifische Sprache ein Metamodell inklusive der statischen Semantik und eine konkrete Syntax zur Entwicklung von Modellen, durch welche die Konzepte der Domäne ausgedrückt werden können. Abbildung 4 zeigt in Form eines konzeptionellen Modells zusammenfassend die Zusammenhänge zwischen den eingeführten Begriffen auf.

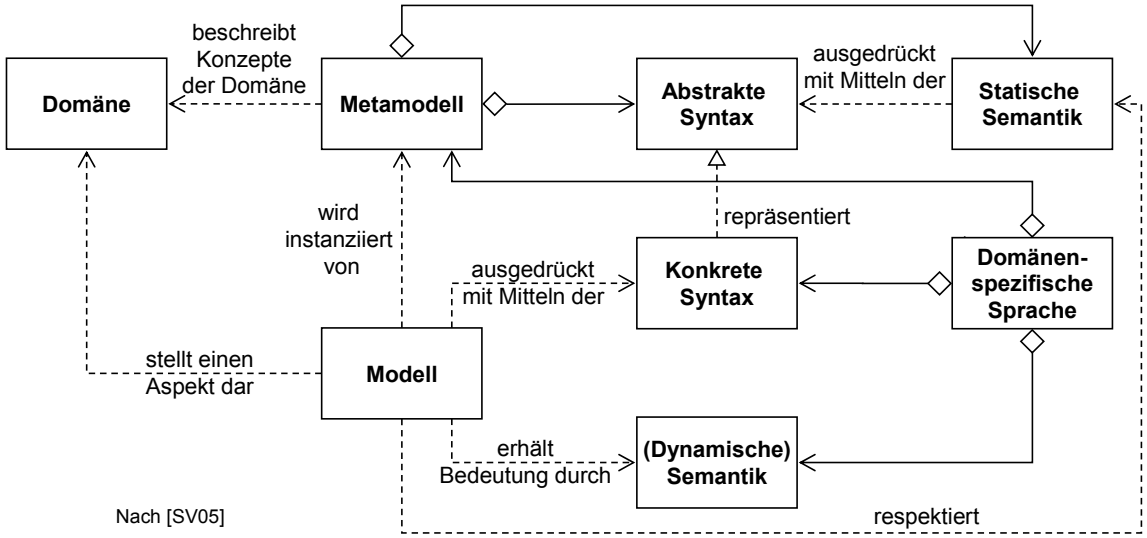

Abbildung 4: Begriffsbildung Metamodelle und domänenspezifische Sprachen

Entsprechend den in Abbildung 4 aufgezeigten Begriffen gibt [PM06] eine ausführliche Definition für ein Metamodell:

> *„Ein Modell ist – bezogen auf die Metaebene eines anderen Modells – dann ein Metamodell, wenn es in einer Abstraktionsbeziehung im Sinne eines höheren Abstraktionsgrades zu diesem steht, Aussagen über das andere Modell (und nicht den Gegenstandsbereich des Modells) macht und die Modellelemente des anderen Modells als Instanzen der Metamodellelemente des Metamodells verstanden werden können. Das Metamodell befindet sich auf der höheren (Meta-)Ebene als das Modell. Modelle und Metamodelle können dieselbe konkrete Syntax verwenden, besitzen jedoch eine unterschiedliche abstrakte Syntax bzw. Semantik.“*

Durch die Formalisierung einer Domäne durch eine domänenspezifische Sprache sind die Mengen und Relationen dieser Domäne hinreichend genau erfasst, um sie mithilfe eines IT-Systems verarbeiten zu

können. Diese Formalisierung bildet auch die Voraussetzung und die Grundlage der modellgetriebenen Softwareentwicklung, deren Konzepte im folgenden Abschnitt eingeführt werden.

2.1.2 Modellgetriebene Softwareentwicklung

Modelle werden bereits seit vielen Jahren in der Softwareentwicklung angewendet. Spätestens mit der Definition allgemeiner Modellierungssprachen wie der *Unified Modeling Language* (UML) [OMG-UML2-Super], die in Abschnitt 2.1.4 detailliert betrachtet wird, gewinnen Modelle zunehmend an Bedeutung. Modelle helfen, die oftmals komplexen Aspekte eines Softwaresystems so zu abstrahieren, dass deren Komplexität für den Menschen handhabbar wird. Modelle schaffen damit eine wesentliche Basis sowohl zur Kommunikation über das Softwaresystem als auch zu dessen Dokumentation [HT06]. Die Verbindung zwischen Software und Modell ist dabei jedoch rein gedanklich. Ändert sich ein Modell, so wird diese Änderung durch die Software nicht reflektiert. Gleichermaßen wird eine Änderung in der Software nicht automatisch im Modell widergespiegelt. Diese Art der Verwendung von Modellen in der Softwareentwicklung wird daher als modellbasiert bezeichnet [SV05]. Obwohl die modellbasierte Softwareentwicklung durch die Verwendung von Modellen einen wesentlichen Fortschritt im Vergleich zu früheren Entwicklungsvorgehen ohne jegliche Modelle darstellt, wird sie den heutigen Anforderungen nach flexiblen und adaptierbaren Softwaresystemen nur schwer gerecht [BI+06]. Um Änderungen an der Software auch im Modell zu manifestieren, ist immer die manuelle Anpassung des entsprechenden Modells notwendig. Folglich betrachten viele Entwickler die sorgfältige Anpassung der Modelle als unnötigen Mehraufwand. Inkonsistente Modelle führen damit in letzter Konsequenz zu einer ebenso inkonsistenten oder gar falschen Dokumentation des Softwaresystems.

Die modellgetriebene Softwareentwicklung (engl. *Model-Driven Software Development*, MDSD) ordnet Modellen daher einen anderen Stellenwert zu. Hier dienen Modelle nicht nur der Dokumentation eines Softwaresystems und damit der Dokumentation von Quellcode; die Modelle sind vielmehr dem Quellcode gleichzusetzen. Somit befasst sich die modellgetriebene Softwareentwicklung mit der Automatisierung in der Softwareherstellung durch die generative Ableitung von Softwareartefakten aus formalen Modellen [PT+07]. Die automatisierte Abbildung der Modelle auf den Quellcode der Softwareartefakte erfolgt mittels sogenannter Transformationen. Die eigentliche Softwareentwicklung wird auf diese Weise vom Quellcode auf die Ebene der Modelle angehoben. Änderungen am Softwaresystem müssen demnach nur im Modell durchgeführt werden, die Umsetzung in den entsprechenden Quellcode erfolgt anschließend automatisch. Durch die Formalisierung der Konzepte einer Domäne durch Modelle kann eine direkte Verbindung zwischen Modellen und daraus abgeleiteten Softwareartefakten hergestellt werden, wodurch im Vergleich zu heutigen Entwicklungsmethoden eine Vielzahl an Vorteilen resultiert. Einige wesentliche Vorteile seien im Folgenden erwähnt [PT+07, SV05]:

- Die Formalisierung einer Domäne ermöglicht die Generierung von Softwareartefakten aus Modellen durch Transformationen. Diese Automation führt zu einer Effizienzsteigerung und Aufwandsreduzierung in der Softwareentwicklung.

- Die manuelle Übertragung eines Modells in Quellcode durch den Menschen basiert auf dessen Interpretation (der Semantik) des Modells und ist daher häufig fehlerhaft. Der Einsatz von Transformationen führt folglich zu einer verbesserten Softwarequalität.

- Die Abstraktion eines Modells gestattet es, den am Entwicklungsprozess beteiligten Experten genau die Aspekte des Softwaresystems zu präsentieren, die für sie von Interesse sind. Diese Trennung der Verantwortlichkeiten (engl. *Separation of Concerns*) gestattet ein fokussiertes

Entwicklungsvorgehen aller Beteiligten. Die Abstraktion durch Modelle führt somit insgesamt zu einer besseren Handhabbarkeit des Softwaresystems.

- Durch die formale Spezifikation einer Domäne können Transformationen, Architekturbausteine und die domänenspezifische Sprache an sich im Sinne einer Softwareproduktionsstraße zur Herstellung unterschiedlicher Softwaresysteme wiederverwendet werden.

Die modellgetriebene Softwareentwicklung ist für sich betrachtet kein neues Entwicklungsvorgehen und auch nicht an ein solches gebunden. Vielmehr stellt sie eine Methode dar, die in unterschiedlichen Entwicklungsvorgehen zum Einsatz kommen kann. Allgemein ist anzumerken, dass sich der Einsatz der modellgetriebenen Softwareentwicklung in iterativen Entwicklungsvorgehen besonders vorteilhaft gestaltet. Softwareentwicklungsprojekte können heute nach Auslieferung der fertigen Anwendung im seltensten Fall als abgeschlossen betrachtet werden. Durch die eingangs angesprochene Konkurrenzsituation der Unternehmen müssen Anwendungen kontinuierlich weiterentwickelt und den aktuellen Anforderungen angepasst werden [SB05]. Neue Anforderungen kommen mit der Zeit hinzu, was eine immer wiederkehrende, iterative Durchführung des Entwicklungsvorgehens bedingt. Durch die mit der modellgetriebenen Softwareentwicklung einhergehende Automation können Iterationszyklen verkürzt und neue Anforderungen rascher umgesetzt werden. Hierbei ist jedoch zu beachten, dass der Einsatz der modellgetriebenen Softwareentwicklung nicht immer zielführend ist. Insbesondere im Bereich der Individualsoftware, die häufig in kleinen oder mittelständischen Softwareunternehmen entwickelt wird, muss erörtert werden, ob sich der initiale Aufwand, der durch die Integration der modellgetriebenen Konzepte in das eigene Entwicklungsvorgehen entsteht, mittel- bis langfristig trägt. Neben der rein technischen Umstellung, die z. B. auch die Anschaffung entsprechender Werkzeuge erfordert, muss vor allem ein Umdenken in der Entwicklungsphilosophie vorangetrieben werden [HT06].

Neben den in Abschnitt 2.1.1 eingeführten Grundlagen der Modelle und Metamodelle führt die modellgetriebene Softwareentwicklung **Modelltransformationen** als weiteres zentrales Konzept ein. Eine Modelltransformation überführt ein Quellmodell in ein Zielmodell und ist daher nach [Uh06] definiert als

„eine berechenbare Abbildung, die einem Quellmodell ein Zielmodell zuordnet."

Ferner unterscheidet die modellgetriebene Softwareentwicklung zwischen **Modell-zu-Modell-** und **Modell-zu-Text-Transformationen**. Erstere überführen ein Quellmodell als Instanz eines Quellmetamodells in ein Zielmodell als Instanz eines Zielmetamodells auf der Grundlage von Transformationsregeln. Zur Spezifikation der Transformationsregeln wird eine Transformationssprache benötigt, die ebenfalls durch ein Metamodell beschrieben ist. Die Ausführung der Transformationsregeln erfolgt durch eine sogenannte **Transformationsausführungsmaschine** (engl. *Transformation Engine*) [CH06]. Diese nimmt Instanzen des Quellmetamodells entgegen, durchsucht diese nach Instanzen von Metaklassen, welche in den Transformationsregeln spezifiziert sind und übersetzt diese Instanzen des Quellmetamodells nach den Transformationsregeln in Instanzen des Zielmetamodells. Quell- und Zielmetamodell müssen dabei nicht zwingend verschieden sein.

Abbildung 5 zeigt die Funktionsweise der Modell-zu-Modell-Transformationen grafisch auf (vgl. auch [KW+03]). Das Resultat einer Modell-zu-Modell-Transformation ist ein Modell, welches das Softwaresystem entweder detaillierter beschreibt und damit ein Modell eines höheren Abstraktionsgrades in ein Modell eines geringeren Abstraktionsgrades überführt (vertikale Transformation), oder ein Modell, das lediglich eine andere Perspektive in Bezug auf die Beschreibung des Softwaresystems einnimmt und damit den Abstraktionsgrad beibehält (horizontale Transformation). Durch horizontale

Transformationen kann damit die Modellierung aspektorientiert erfolgen, indem für jeden Aspekt des Softwaresystems ein eigenständiges Modell erzeugt wird [PT+07].

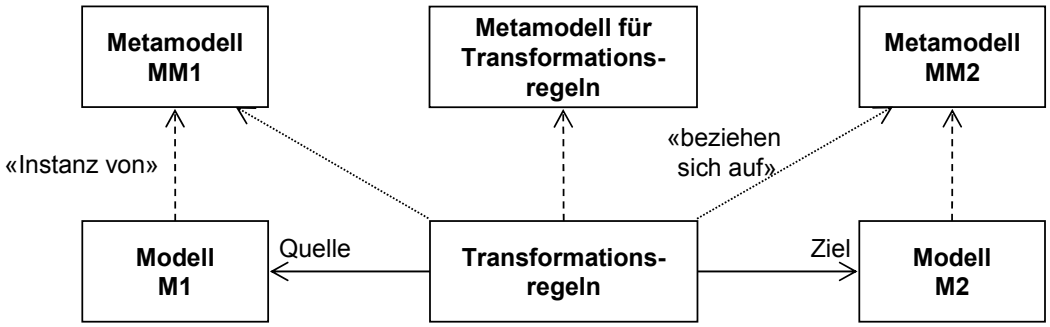

Abbildung 5: Funktionsweise von Modell-zu-Modell-Transformationen

Trotz der Vision der modellgetriebenen Softwareentwicklung, alle Softwareartefakte durch Modelle spezifizieren zu können, ist diese Vision heute noch nicht erreicht. Gerade für höhere Programmiersprachen wie beispielsweise Java oder .NET, die heute im Kontext der eigentlichen Implementierung zum Einsatz kommen, liegen keine Metamodelle im Sinne der modellgetriebenen Softwareentwicklung vor. Vielmehr beruhen diese Sprachen auf einer konkreten textuellen Syntax. Daher werden Modell-zu-Text-Transformationen benötigt, welche die textuelle Syntax einer höheren Programmiersprache mit in der Regel nicht vorhandenem Metamodell als Ziel haben. Somit entsprechen Modell-zu-Text-Transformationen bis auf das unbekannte Zielmetamodell den Modell-zu-Modell-Transformationen und werden daher als ein Spezialfall der Modell-zu-Modell-Transformationen aufgefasst. Zur Spezifikation und Ausführung von Transformation existiert eine Vielfalt an Transformationssprachen und Transformationsausführungsmaschinen, deren ausführliche Behandlung in [CH06] erfolgt und daher nicht Gegenstand dieser Arbeit ist.

Zur Entwicklung der im weiteren Verlauf benötigten Transformationen setzt diese Arbeit zur Modell-zu-Modell-Transformation die Transformationssprache *Query, View, Transformation* (QVT) ein, die im Kontext der modellgetriebenen Architektur in Abschnitt 2.1.3.2 näher beschrieben wird und nach [CH06] zu den relationalen Ansätzen zu zählen ist. Zur Umsetzung der benötigten Modell-zu-Text-Transformationen hingegen wird ein sogenannter vorlagenbasierter Ansatz bevorzugt. Eine Vorlage (engl. *Template*) kann im einfachsten Fall ein Softwareartefakt sein, das durch die Ausprägung einiger Platzhalter durch eine Transformationsausführungsumgebung zu lauffähigem Code abgebildet werden kann. Dieser Ansatz geht nicht von der Existenz eines Metamodells aus und eignet sich daher insbesondere für Modell-zu-Text-Transformationen. Kapitel 5 geht auf beide Transformationsarten detailliert ein.

2.1.3 Modellgetriebene Architektur

Eine konkrete Ausprägung der modellgetriebenen Softwareentwicklung stellt die modellgetriebene Architektur (*Model-Driven Architecture*, MDA) dar, welche durch die *Object Management Group* (OMG) standardisiert wurde [OMG-MDA]. Als Ausprägung der modellgetriebenen Softwareentwicklung verfolgt die MDA somit die gleichen Ziele, konkretisiert diese aber durch die Vorgabe anzuwendender Standards [MM03]. So gibt die MDA als Meta-Metamodell die *Meta Object Facility* (MOF) [OMG-MOF2] vor und spezifiziert MOF gleichzeitig als die oberste Meta-Ebene, die sich selbst beschreibt. Theoretisch gestattet die MDA die Verwendung jedes auf der MOF aufbauenden Metamodells. Sie empfiehlt aber die Verwendung von UML-Profilen, dem Erweiterungsmechanismus der *Unified Modeling Language* (UML) und damit die Verwendung der UML an sich. Sowohl die UML

als auch deren Erweiterungsmechanismen werden daher in Abschnitt 2.1.4 detailliert betrachtet. Zur Spezifikation der statischen Semantik sieht die MDA den Einsatz der *Object Constraint Language* (OCL) [OMG-OCL2] vor.

2.1.3.1 Object Constraint Language

Die *Object Constraint Language* (OCL) [OMG-OCL2] ist eine formale, deklarative Sprache der OMG und ermöglicht die semantische Anreicherung von objektorientierten Modellen [PM06]. Hierbei kann die OCL auf alle Modelle angewendet werden, die konform zur *Meta Object Facility* (MOF) sind [OMG-MOF2]. Durch Einschränkungen (engl. *Constraints*), die für die Elemente eines Modells gelten, kann sowohl die statische als auch die dynamische Semantik des Modells spezifiziert werden. In [WK03] wird eine Einschränkung definiert als

> *"a restriction on one or more values of (part of) an object-oriented model or system."*

Für das Verständnis der in dieser Arbeit spezifizierten OCL-Ausdrücke seien die drei folgenden syntaktischen Konstrukte der OCL detailliert eingeführt [PM06]:

- **Kontext**: Das Schlüsselwort `context` verweist auf einen Namensraum, für den die nachfolgenden Einschränkungen gelten.
- **Invarianten, Vor-** und **Nachbedingungen** werden durch die entsprechenden Schlüsselwörter `inv, pre, post` eingeleitet.
- **Operationen**: OCL kennt eine Vielzahl vordefinierter Operationen, wie etwa die relationalen Operationen >, <= oder `implies`.

Mit diesen Konstrukten lässt sich am Beispiel der Landkarte die Einschränkung, dass eine Stadt genau dann eine Großstadt ist, sofern sie mehr als 100.000 Einwohner hat, wie folgt formalisieren.

```
context Stadt
inv: (einwohner > 100.000) implies großstadt = true
```

In obigem OCL-Ausdruck wird im Kontext der Stadt eine Invariante definiert, welche die boolesche Eigenschaft `großstadt` auf wahr (`true`) setzt, sofern mehr als 100.000 Einwohner in einer Stadt vorhanden sind. Da OCL-Ausdrücke immer zu den booleschen Werten `wahr` oder `falsch` ausgewertet werden können, kann durch die Auswertung aller Einschränkungen eines Modells festgestellt werden, ob sich ein Modell in konsistentem Zustand befindet und damit wohlgeformt ist oder nicht. Diese Präzisierung des Modellelements `Stadt` ließe sich durch unterschiedlichste Sprachen, inklusive der natürlichen, ausdrücken. Die Verwendung formaler Sprachen, welche wie die OCL auf einer mathematischen Grundlage aufbauen, reduziert jedoch die Gefahr einer Fehlinterpretation der Einschränkungen. Trotz ihrer mathematischen Grundlage achtete die OMG bei der Spezifikation der OCL auf die Anwendbarkeit der Sprache durch ein möglichst breites Spektrum von Anwendern, woraus sich die oben beispielhaft vorgestellte eingängige Syntax der OCL erklärt [WK03].

Neben der wichtigen Funktion der formalen Spezifikation der statischen und dynamischen Semantik von Modellen wird die OCL im Kontext der MDA auch zur formalen Spezifikation von Transformationen eingesetzt. Wie im folgenden Abschnitt verdeutlicht wird, muss der für die modellgetriebene Softwareentwicklung zentrale Aspekt der (automatischen) Modelltransformation ebenfalls durch eine

formale Sprache untermauert werden. Die OMG schlägt hier die Verwendung des eigenen Standards *Query, View, Transformation* (QVT) [OMG-QVT] vor.

2.1.3.2 Query, View, Transformation

Die von der OMG spezifizierte Transformationssprache *Query, View, Transformation* (QVT) [OMG-QVT] ist Teil der Spezifikation der *Meta Object Facility* (MOF) [OMG-MOF2]. Die elementaren Sprachelemente von QVT werden bereits durch die Namensgebung der Sprache umrissen. Ein *Query* (dt. Abfrage) ist ein formaler Ausdruck, der zur Auswahl von Modellelementen benötigt wird. Eine *View* (dt. Sicht) ist eine komplexe *Query*, die ein ganzes Modell oder einen bestimmten Teil davon auswählt. Eine Transformation schließlich umfasst einen Satz von Regeln, der Beziehungen zwischen Modellen spezifiziert und für die Abbildung eines Quellmodells in ein Zielmodell genutzt werden kann [Bo06, He05]. Um Abfragen auf diesen Modellen auszuführen, nutzt QVT die zuvor vorgestellte *Object Constraint Language* und referenziert diese dementsprechend im vorliegenden Standard [OMG-QVT] als Abfragesprache. QVT kann in der Praxis dazu genutzt werden, um Modelle zu transformieren, Veränderungen auf Modellen vorzunehmen und Modellelemente zu löschen, zu verändern oder neue einzufügen sowie um Validierungen auf Modellen auszuführen. Die OMG unterteilt QVT in verschiedene Sprachteile, wie Abbildung 6 zeigt.

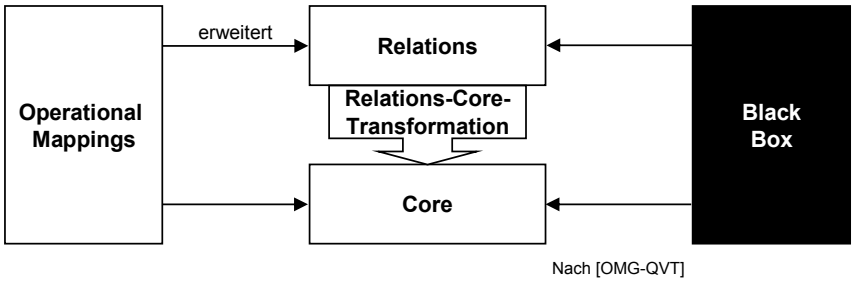

Abbildung 6: Zusammenhang der Sprachteile der QVT-Spezifikation

Mit *Relations* und *Core* definiert die QVT-Spezifikation zwei deklarative Sprachen, die bezogen auf ihren Funktionsumfang identisch sind, sich aber in der Komplexität der verfügbaren Konstrukte unterscheiden. Während *Core* simple und für den Menschen leichter verständliche Konstrukte zur Spezifikation von Transformationen zur Verfügung stellt, verfügt *Relations* nur über komplexere Konstrukte, die sich dafür aber in kürzeren Transformationsspezifikationen niederschlagen. Die Gleichmächtigkeit beider Sprachen wird im Schaubild durch die *Relations*-*Core*-Transformation angedeutet. Da diese Arbeit Transformationen in der Sprache *Relations* entwickelt, werden dieser Sprachteil und insbesondere die durch die OMG bereitgestellte grafische Syntax für *Relations* in Kapitel 5.2.1 detailliert betrachtet. Einen weiteren Sprachteil bilden die sogenannten *Operational Mappings*, welche QVT neben den beiden deklarativen Sprachen *Relations* und *Core* um eine imperative Sprache erweitern. Syntaktisch enthalten die *Operational Mappings* viele Konstrukte wie beispielsweise Schleifen oder bedingte Befehlsausführungen, die aus anderen imperativen Programmiersprachen bekannt sind. Mit der *Black Box* (vgl. Abbildung 6) führt die OMG schließlich einen Mechanismus ein, der eine Erweiterung von QVT um zusätzliche Transformationsregeln, die in anderen Sprachen wie beispielsweise XSLT, SQL oder Java spezifiziert wurden, ermöglicht.

2.1.3.3 Abstraktionsebenen der modellgetriebenen Architektur

Neben der Konkretisierung der modellgetriebenen Softwareentwicklung durch deren Untermauerung durch Standards geht die MDA noch einen Schritt weiter, indem sie die heute vorhandene Vielzahl der

Technologien, Plattformen und Konzepte adressiert. Diese Vielfalt kann durch die Entwicklung eines einzigen Softwaresystems nicht berücksichtigt werden. Daher führt die MDA eine klare Trennung zwischen der eigentlichen Fachfunktionalität des zu entwickelnden Softwaresystems und den abstrakt als Plattform bezeichneten, technischen Gegebenheiten der Ausführungsumgebung des Softwaresystems ein [MM03]:

> *"The Model-Driven Architecture starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform."*

Wie bereits angedeutet, überführt eine vertikale Transformation ein Modell höheren Abstraktionsgrades in ein Modell geringeren Abstraktionsgrades, bis letztendlich die eigentliche Implementierung als Ergebnis dieser systematischen Verfeinerung vorliegt.

Die modellgetriebene Softwareentwicklung trifft hier jedoch keine Aussage darüber, wie viele Abstraktionsgrade bis zur Erreichung der Implementierung notwendig sind. Die OMG hingegen konkretisiert die benötigten Abstraktionsebenen, indem sie durch das MDA-Rahmenwerk Modelle auf den folgenden Abstraktionsebenen vorsieht [MM03, KW+03, PM06, SV05, Uh06]:

- Das **berechnungsunabhängige Modell** (engl. *Computation Independent Model*, CIM) beschreibt das zu entwickelnde Softwaresystem frei von umsetzungsspezifischen Details. Ein CIM dient somit zur Erfassung der Problemdomäne und den mit ihr einhergehenden Anforderungen, nicht aber der Struktur oder gar der Implementierung des Softwaresystems. Daher wird ein CIM auch als Domänenmodell (engl. *Domain Model*) oder Geschäftsmodell (engl. *Business Model*) bezeichnet. Die in einem CIM modellierten fachlichen Aspekte sind folglich auch dann noch gültig, wenn gar keine Software entwickelt werden soll. Diese abstrakte Sicht gestattet es, das CIM als Grundlage zur Diskussion zwischen dem Kunden bzw. dem späterem Anwender und den beteiligten Softwareentwicklern zu nutzen.

- Das **plattformunabhängige Modell** (engl. *Platform Independent Model*, PIM) beschreibt die Struktur und das Verhalten eines Softwaresystems unabhängig von der zur Umsetzung verwendeten konkreten Plattform. Das plattformunabhängige Modell stellt somit die Blaupause für ein zu entwickelndes Softwaresystem dar und kann für beliebige Plattformen ausgeprägt werden. Der Plattformbegriff wird von der MDA relativ eingeführt. Abhängig von der eigenen Perspektive kann ein Modell in einem Kontext plattformunabhängig, in einem weiteren Kontext bereits plattformspezifisch sein. Spezifiziert ein Modell beispielsweise Details im Kontext der Java Standard Edition, so ist dasselbe Modell jedoch unabhängig von der logisch unterhalb angeordneten virtuellen Java-Laufzeitumgebung.

- Als drittes Modell spezifiziert die MDA das **plattformspezifische Modell** (engl. *Platform Specific Model*, PSM), das im Gegensatz zu einem PIM konkrete Details in Bezug zu einer Plattform spezifiziert. Durch die Vielzahl der heute vorhandenen Plattformen existiert zu einem PIM in der Regel mehr als ein PSM. Jedes PSM spezifiziert somit das gleiche Softwaresystem, jeweils ergänzt um die Charakteristika einer speziellen Plattform. Der Übergang von PIM zu PSM findet dabei durch Modell-zu-Modell-Transformation statt, die im Folgenden genauer beleuchtet werden. Durch die Relativität der Plattform kann der Übergang von PIM zu PSM mitunter auch mehrstufig sein. Ein erzeugtes PSM kann somit aus Sicht der nächsttieferen Plattformebene wiederum als PIM genutzt werden.

Während die MDA ein plattformspezifisches Modell als Modell der Implementierung und damit als die Implementierung selbst ansieht, ist eine weitere Transformation vom plattformspezifischen Modell zur eigentlichen Implementierung hin notwendig. Diese letzte Modell-zu-Text-Transformation, die in der plattformspezifischen Implementierung (engl. *Platform Specific Implementation*, PSI) [PM06] mündet, ist, wie bereits in Abschnitt 2.1.2 erläutert, rein pragmatischer Natur. Das Ziel der OMG, mit der MDA ein Modell direkt ausführbar als sogenanntes *Executable Model* zu spezifizieren, wurde bisher noch nicht erreicht. Daher kann aus einem PSM nur ein gewisser Teil der notwendigen Implementierung gewonnen werden. Häufig sind manuelle Nachbearbeitungen der Implementierung notwendig, was somit die Transformation auf die plattformspezifische Implementierung motiviert. Abbildung 7 stellt die eingeführten Modelle und Transformationen im Zusammenhang dar.

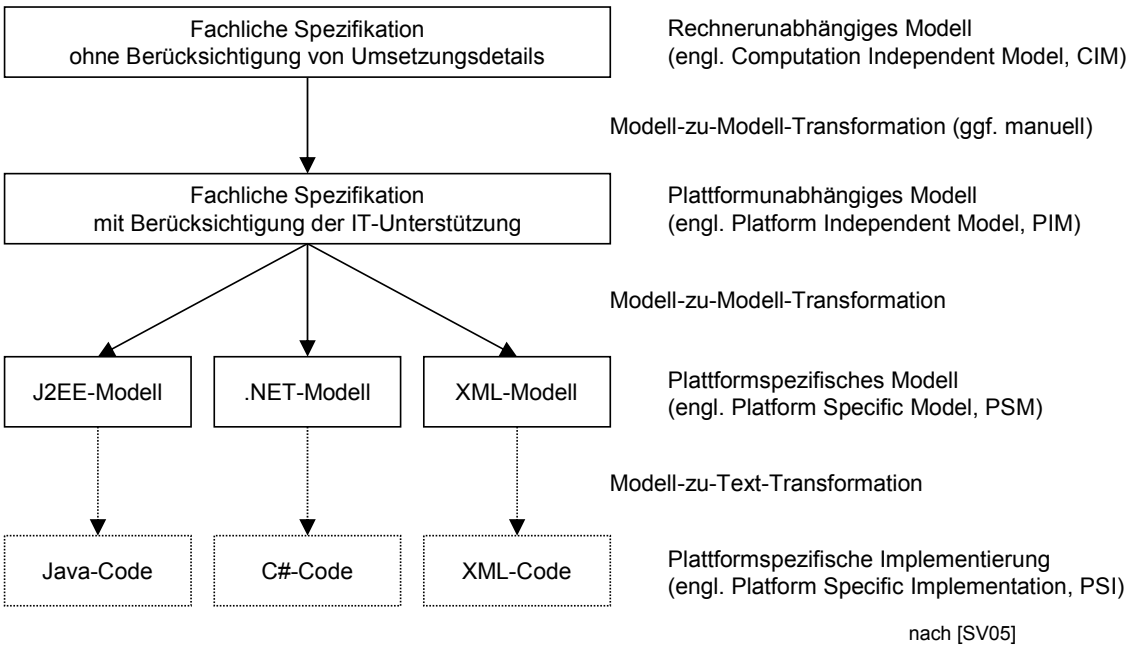

Abbildung 7: Abstraktionsebenen der modellgetriebenen Architektur

Wie angeführt und in Abbildung 7 dargestellt, kennt die MDA eine Modell-zu-Text-Transformation nicht, da aus ihrer Sicht ein PSM bereits den fertigen Quellcode im Sinne eines *Executable Models* darstellt. Ebenso trifft die MDA keine Aussage über eine Modell-zu-Modell-Transformation zwischen einem CIM und dem zugehörigen PIM. Sie empfiehlt lediglich [MM03]:

> *"In an MDA specification of a system CIM requirements should be traceable to the PIM and PSM constructs that implement them, and vice versa."*

Die geforderte Nachvollziehbarkeit der durch das CIM spezifizierten Anforderungen erhält unter dem Gedanken, dass ein CIM nicht den formalen Ansprüchen eines PIM genügen muss, besondere Bedeutung [PM06]. Ein CIM kann im einfachsten Falle auch nur durch ein Blatt Papier mit einigen Skizzen darauf spezifiziert werden. Das CIM dient dann zwar als Grundlage zur Erstellung eines PIM, die Umsetzung dieser Spezifikationen auf ein PIM kann jedoch nicht automatisiert erfolgen.

Die sich anschließende Transformation von PIM zu PSM kann automatisiert durch eine Modell-zu-Modell-Transformation erfolgen. Hierzu muss allerdings die Zielplattform der Transformation als formale Spezifikation in Form eines Metamodells vorliegen. Häufig existiert eine Spezifikation der Zielplattform jedoch nur in Form eines Betriebshandbuches oder nur im Wissen des Softwarearchitekten [PM06]. Da diese Spezifikationen einer automatischen Transformation nicht genügen, fordert die

MDA hier eine als Plattformmodell (engl. *Platform Model*, PM) bezeichnete formale Spezifikation der Zielplattform, die unabhängig von PIM oder PSM die Konzepte der Plattform beschreibt [MM03]. Ein PM kann daher als das Metamodell der Zielplattform verstanden werden. Durch das PM der Zielplattform und dem zugehörigen PIM kann so mittels einer Modelltransformation das entsprechende PSM erzeugt werden.

An dieser Stelle sei angemerkt, dass in der Existenz des plattformspezifischen Modells ein zentraler Unterschied zu anderen Ansätzen der Codegenerierung wie beispielsweise der generativen Programmierung (engl. *Generative Programming,* GP) [CE00] liegt. Die generative Programmierung konzentriert sich auf die Erzeugung von Quellcode, die modellgetriebene Architektur auf die Erzeugung von Modellen. Eine direkte Transformation eines plattformunabhängigen Modells zu Quellcode ist zunächst zwar denkbar, hat aber wesentliche Nachteile [PM06]: Einerseits benötigt das PIM so viele Zusatzinformationen auch im Sinne von plattformspezifischen Details, dass es kaum noch als plattformunabhängig bezeichnet werden kann. Andererseits geht sowohl die Skalierbarkeit als auch die Handhabbarkeit der Modelle verloren, da im Falle einer direkten Transformation von PIM zu Quellcode nur noch ein PIM für alle unterschiedlichen Aspekte eines Softwaresystems existiert.

Die Erstellung eines CIM kann informal mittels Bleistift und Papier erfolgen. Um derartige Spezifikationen jedoch automatisiert weiterverarbeiten zu können, müssen sie formal erfasst werden. Daher bedarf es einer Sprache, die einerseits für den Menschen handhabbar ist und andererseits über ein geeignetes Metamodell verfügt. Modellierungssprachen, die über eine grafische Notation verfügen, haben sich als besonders hilfreich für den Menschen herausgestellt [HT06, Fo04, QP06]. Insbesondere im Bereich der Geschäftsprozessmodellierung, die neben der Informatik auch in vielen weiteren Fachdomänen eine zentrale Rolle spielt, haben sich unterschiedlichste grafische Modellierungssprachen etabliert. Die *Business Process Modeling Notation* (BPMN) [OMG-BPMN] oder die *Unified Modeling Language* (UML) [OMG-UML2-Super] seien als zwei der bekanntesten Vertreter dieser Modellierungssprachen genannt [RH06]. Durch deren Bekanntheit gibt es viele Werkzeuge, welche die Modellierung in diesen Sprachen ermöglichen und den menschlichen Modellierer in der Erstellung von korrekten Modellen unterstützen. Im Bereich der Modellierung von Softwarearchitekturen als zweite in dieser Arbeit betrachteten Domäne hat sich die UML ebenfalls als Standard etablieren können [Oe06]. Ferner wird die UML aufgrund ihrer Ausrichtung an MOF und ihrer Erweiterbarkeit von der MDA als Modellierungssprache empfohlen [MM03]. Die UML bildet daher die Modellierungsgrundlage dieser Arbeit und wird im folgenden Abschnitt detailliert betrachtet.

2.1.4 Unified Modeling Language

Die *Unified Modeling Language* (UML) ist eine durch die *Object Management Group* (OMG) standardisierte Modellierungssprache, die im Zuge des Zusammenschlusses verschiedener Modellierungssprachen entstand. Ihre ursprünglichen Autoren Booch, Jacobson und Rumbaugh hatten jeweils eigene Modellierungssprachen zur objektorientierten Systemmodellierung entwickelt. Gemeinsam flossen diese in die erste Spezifikation von UML ein. Seit 1996 wird die Standardisierung, die Pflege und die Weiterentwicklung der UML durch die OMG vorangetrieben, die sie 1997 in der Version 1.0 zum ersten Mal als Standard veröffentlichte. Im Jahre 2005 fasste die OMG tief gehende Änderungen an der UML zu einer neuen, als UML2 bezeichneten Version der UML zusammen. Eine wesentliche Änderung, die mit Version 2 etabliert wurde, ist die Strukturierung der UML in drei Teilstandards:

- Die **UML-*Infrastructure*** [OMG-UML2-Infra] stellt die Basis der UML2 durch die Spezifikation der Kernelemente dar. Zusätzlich liefert sie einige Möglichkeiten zur Spracherweiterung, die im Folgenden noch konkreter beleuchtet werden.

- Die **UML-*Superstructure*** [OMG-UML2-Super] liefert aufbauend auf der UML-*Infrastructure* verschiedene Sprachelemente, die im Kontext unterschiedliche Diagrammtypen verwendet und visualisiert werden können.

- Mit der ***Object Constraint Language*** [OMG-OCL2] können Modelle um zusätzliche Einschränkungen (engl. *Constraints*) wie beispielsweise Invarianten, Vor- oder Nachbedingungen erweitert werden (vgl. hierzu Abschnitt 2.1.3.1). Diese verändern das Modell nicht, sondern sind aufgrund ihrer deklarativen Spezifikation automatisiert zu einem booleschen Wert auswertbar. Wird ein OCL-Ausdruck zu `wahr` ausgewertet, so befindet sich das Modell in einem zu dieser Einschränkung konsistenten Zustand.

Aktuell ist die UML in der Version 2.1.2 durch die OMG als Standard final verabschiedet. Auf diese Version bezieht sich diese Arbeit im Folgenden mit der Bezeichnung „UML“.

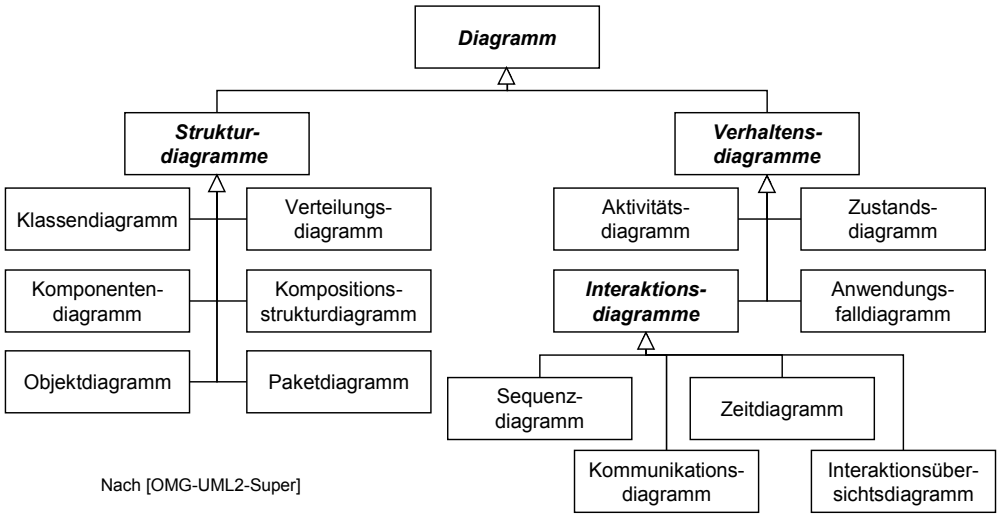

Abbildung 8: Die Diagrammtypen der Unified Modeling Language

Ein besonderes Augenmerk der OMG liegt auf der Portabilität, der Interoperabilität und der Wiederverwendbarkeit der UML. Daher stellt die UML keinen Bezug zu einer spezifischen Programmiersprache, einer Domäne oder auch einem speziellen Entwicklungsvorgehen her. Sie ist somit keine Methode, sondern muss vielmehr als Familie grafischer Notationen auf der Basis eines einzigen Metamodells verstanden werden [Oe06]. Dabei ist zu unterstreichen, dass die UML verschiedenste sowohl statische als auch dynamische Diagrammtypen kennt, die in die beiden Gruppen der **Struktur- und Verhaltensdiagramme** unterteilt werden können. Strukturdiagramme stellen statische Sachverhalte dar, beispielsweise die Struktur eines Softwaresystems zu einem bestimmten Zeitpunkt. Die zweite Gruppe der Verhaltensdiagramme wird zur Modellierung von dynamischen Aspekten verwendet. Folglich betrachten Verhaltensdiagramme immer einen gewissen Zeitraum, in dem das Handeln und die Aktionen einzelner Objekte, insbesondere aber auch die Interaktion von Objekten untereinander fokussiert werden. Abbildung 8 zeigt gemäß der eben vorgestellten Unterteilung die dreizehn Diagrammtypen der UML, von denen in dieser Arbeit insbesondere das Klassen-, Aktivitäts-, Anwendungsfall- und Sequenzdiagramm Verwendung finden. Eine umfassende und detaillierte Betrachtung der einzelnen Diagrammtypen ist in unterschiedlichen Literaturquellen wie [Ke06, Oe06, Fo04] zu finden und daher nicht Gegenstand dieser Arbeit.

Obwohl die UML in ihrer Spezifikation ein breites Spektrum an Modellierungsmöglichkeiten bietet, können durch ihre Erweiterungsmechanismen auch domänenspezifische Anforderungen unterstützt

werden, die über den eigentlichen Spezifikationsumfang der UML hinausgehen. Grundlegend für diese Erweiterungsmechanismen ist das formale Metamodell der UML. Wie die MDA nutzt die UML die *Meta Object Facility* (MOF) als Meta-Metamodell (M3). Die UML-*Superstructure* instanziiert MOF und stellt damit das Metamodell (M2) für die UML bereit (vgl. Abbildung 9). Auf dieser Modellebene lässt die UML zwei mögliche Erweiterungsmechanismen zu: Eine **schwergewichtige Erweiterung** der UML verändert deren Metamodell, eine **leichtgewichtige Erweiterung** nutzt das bestehende UML-Metamodell und stellt auf diesem aufbauend weitere Modellelemente durch die Spezifikation eines sogenannten **UML-Profils** zur Verfügung.

Eine schwergewichtige Erweiterung der UML nimmt eine direkte Veränderung des Metamodells der UML vor. Diese kann von der Veränderung bestehender Metaklassen bis hin zu gänzlich neuen Metaklassen und Konzepten auf der Metaebene reichen. Durch eine schwergewichtige Erweiterung kann somit einerseits ein eigenes Metamodell spezifiziert werden, das – frei von den durch das UML-Metamodell vorgegebenen Einschränkungen – eine hohe Anpassbarkeit an die eigene Domäne ermöglicht. Andererseits bringt eine schwergewichtige Erweiterung den Verlust der Werkzeugunterstützung durch herkömmliche Modellierungswerkzeuge mit sich, da diese mit dem veränderten oder neuen Metamodell nicht umgehen können. Es ist also abzuwägen, ob die Vorteile eines auf die eigenen Bedürfnisse angepassten Metamodells den Aufwand der Implementierung einer eigenen Werkzeugunterstützung rechtfertigen.

Da schwergewichtige Erweiterungen in vielen Fällen mit einem zu großen Aufwand verbunden sind, bietet die UML zusätzlich einen als UML-Profil bezeichneten, leichtgewichtigen Erweiterungsmechanismus an. Der Einsatz eines UML-Profils eignet sich besonders dann, wenn die Spezifika einer speziellen Domäne im Modell reflektiert werden sollen und im UML-Metamodell dazu keine geeigneten Metaklassen vorhanden sind. Die durch ein UML-Profil möglich werdenden Erweiterungen des bestehenden UML-Metamodells sind durch das *Profiles*-Paket der *Infrastructure Library* in der UML-*Superstructure* definiert. In einem UML-Profil dürfen keine eigenen Metaklassen spezifiziert, sondern lediglich vorhandene Metaklassen des UML-Metamodells erweitert werden. Daher heben UML-Profile bestehende Einschränkungen (engl. *Constraints*) durch das UML-Metamodell nicht auf, sondern erweitern diese lediglich [SV05]. Hierzu stehen die drei Konzepte *Stereotypes*, *Tagged Values* und *Constraints* zur Verfügung.

Das Konzept des Stereotyps (engl. *Stereotype*) erweitert eine bestehende UML-Metaklasse (M2). Ein Pfeil mit ausgefüllter Spitze spezifiziert die dazu notwendige Erweiterungsbeziehung (*Extension*) wie in Abbildung 9 beispielhaft an der Erweiterung der UML-Metaklasse `Class` durch einen Stereotyp `MyClass` gezeigt wird. Wird dieser Stereotyp instanziiert und in einem Modell (M1) verwendet, so hat das entsprechende Modellelement den Stereotyp `MyClass`. Als grafische Repräsentation eines Stereotyps wird auf M1-Ebene häufig das UML-Klassensymbol verwendet. Das in Abbildung 9 gezeigte Modellelement `Person` hat somit den Stereotyp `MyClass` und wird durch ein UML-Klassensymbol repräsentiert. Zur grafischen Repräsentation eines Stereotyps kann jedoch auch eine eigene konkrete Syntax verwendet werden, sofern das zum Einsatz kommende Entwicklungswerkzeug dies unterstützt. Die OMG beschreibt in [OMG-UML2-Super] einen Stereotyp wie folgt:

> *"Stereotype is a kind of Class that extends Classes through Extensions. Just like a class, a stereotype may have properties, which may be referred to as tag definitions. When a stereotype is applied to a model element, the values of the properties may be referred to as tagged values."*

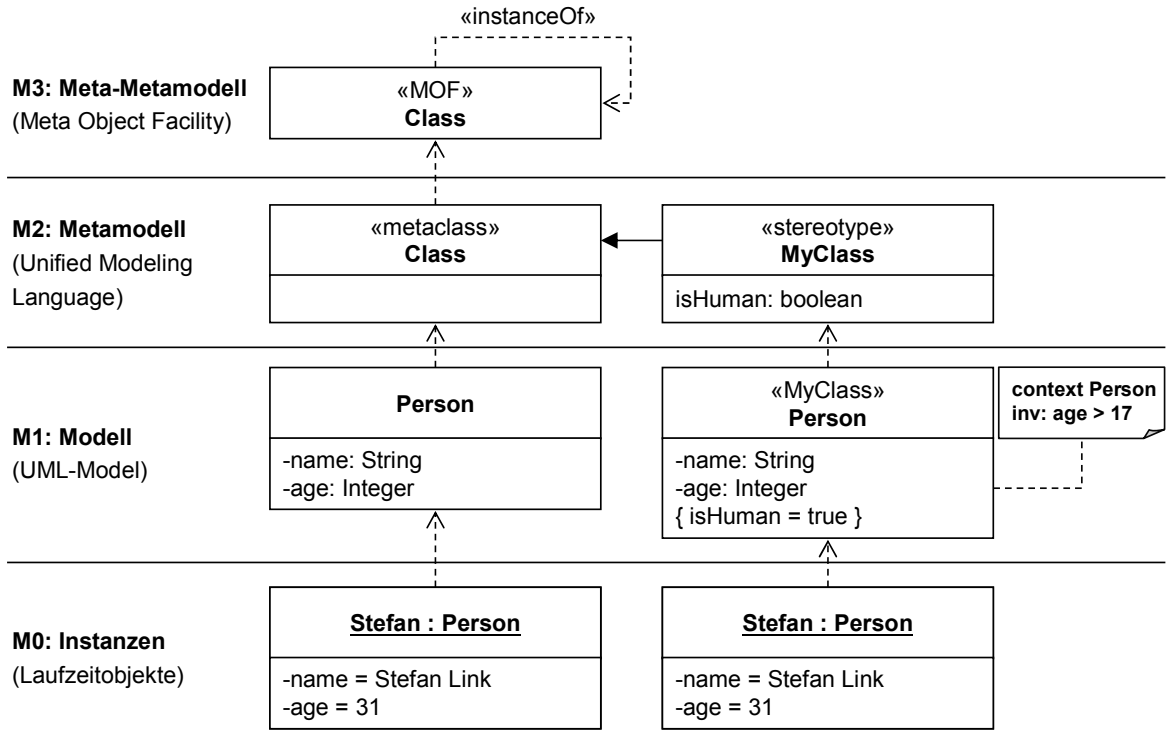

Abbildung 9: Modellierungshierarchie der Unified Modeling Language

Jedem Stereotyp können zusätzlich Eigenschaften durch sogenannte *Tag Definitions* zugewiesen werden. So hat der Stereotyp `MyClass` in oben angeführtem Beispiel die Eigenschaft `isHuman`. Wird der Stereotyp in einem Modell instanziiert, so müssen diese Eigenschaften des Stereotyps mit entsprechenden als *Tagged Values* bezeichneten Werten belegt werden. Diese Eigenschaften sind dann im Gegensatz zu den Attributen einer normalen Klasse nicht optional, woraus auch die Bezeichnung *Tagged Value* (dt. wörtlich „angehefteter Wert“) hergeleitet werden kann. Wird ein Stereotyp mit einer Eigenschaft wie beispielsweise `isHuman` versehen (eine *Tag Definition*), so tragen alle Instanzen dieses Stereotyps diese Eigenschaft. Im Sinne eines Name-Wert-Paares muss diese mit einem entsprechenden Eigenschaftswert (ein *Tagged Value*) je Instanz belegt werden. Im konkreten Beispiel der *Tag Definition* `isHuman` kommen durch den angegebenen Typ `Boolean` nur die beiden *tagged values* `true` oder `false` infrage.

Einschränkungen (engl. *Constraints*) dienen als drittes Konzept der UML-Profile dazu, die Semantik eines Stereotyps noch präziser spezifizieren zu können. Auch hier gilt, dass die vorhandenen Einschränkungen des UML-Metamodells nicht aufgeweicht, sondern lediglich erweitert werden können. Einschränkungen müssen im Sinne der booleschen Algebra immer zu `wahr` oder `falsch` ausgewertet werden können. Hinsichtlich der Syntax kommt im Kontext der MOF-basierten Sprachen wie der UML häufig die in Abschnitt 2.1.3.1 vorgestellte *Object Constraint Language* (OCL) zum Einsatz. In Abbildung 9 ist beispielhaft eine Einschränkung als OCL-Ausdruck angegeben. Diese setzt im Kontext der Klasse `Person` vom Stereotyp `MyClass` als Invariante fest, dass eine Person immer mindestens 18 Jahre alt sein muss. In OCL formulierte Einschränkungen sind deklarativ und daher unabhängig von einer konkreten Modellierungssprache. Damit können OCL-Einschränkungen auf M2-Ebene oder theoretisch auch auf M3-Ebene verwendet werden. Allgemeiner formuliert wird eine Einschränkung auf Ebene M_n spezifiziert und wirkt sich dann auf Ebene M_{n-1} aus [SV05]. Jedoch ist insbesondere auf M2-Ebene der Einsatz einer maschinell verarbeitbaren Spezifikationssprache für Bedingungen von Vorteil, da auf dieser Ebene Einschränkungen eine präzisere Spezifikation von Metamodellen, die trotzdem noch maschinell verarbeitbar bleiben, gestatten.

Die Konzepte der modellgetriebenen Entwicklung und insbesondere der modellgetriebenen Architektur können durch die UML spezifiziert und für den Menschen visualisiert werden. Als zentraler Bestandteil dieser Arbeit muss die modellgetriebene Entwicklung und die Verwendung der UML nun im Kontext der Geschäftsprozesse, der Benutzerinteraktion und der dienstorientierten Architekturen betrachtet werden. Alle drei Bereiche werden in den folgenden Abschnitten 2.2 bis 2.4 beleuchtet.

2.2 Modellgetriebene Entwicklung von Geschäftsprozessen

Bedingt durch eine zunehmende Globalisierung der Märkte sind Unternehmen zunehmend dazu gezwungen, ihre Produkte und Dienstleistungen einer fortwährenden Evolution zu unterziehen, um wettbewerbsfähig zu bleiben [CB+04]. Hierzu ist ein klares Verständnis über die in einem Unternehmen vorhandenen Geschäftsprozesse notwendig, welche der Erzeugung eines Produkts oder der Erbringung einer Dienstleistung dienen. Damit stellen Geschäftsprozesse den zentralen Faktor der Wertschöpfung eines Unternehmens dar [SS+07b]. Bei der zur Unterstützung der Geschäftsprozesse zum Einsatz kommenden IT hingegen spiegelt sich die zentrale Rolle der Geschäftsprozesse häufig nicht wider. Eine heterogene und nur schwer an sich ändernde Geschäftsprozesse anpassbare IT ist die Folge. Diese fehlende Flexibilität führt dazu, dass aktuell rund 80 % der Ausgaben eines Unternehmens im Bereich der IT nur für die Erweiterung und Wartung bestehender Anwendungen benötigt werden [Mi05]. Folglich ist eine neue Herangehensweise an die Bereitstellung der benötigten IT notwendig, um diesen hohen Anteil reduzieren zu können.

Der Ansatz der geschäftsgetriebenen Entwicklung (engl. *Business-Driven Development*, BDD) rückt die für ein Unternehmen zentralen Geschäftsprozesse in den Mittelpunkt der IT und strebt einen engen Bezug zwischen den Geschäftsprozessen und deren IT-Unterstützung an. Dienen IT-Systeme heute häufig einer rein funktionalen Unterstützung einzelner Aufgaben, so adressiert die geschäftsgetriebene Entwicklung eine gesamtheitliche Ausrichtung der bestehenden, aber auch künftig zu entwickelnden IT-Systeme an den Geschäftsprozessen. Bevor nun die Konzepte der geschäftsgetriebenen Entwicklung detailliert betrachtet werden, sollen Definitionen aus dem Bereich der Geschäftsprozesse und Workflows gegeben werden, die für den weiteren Verlauf dieser Arbeit wesentlich sind.

2.2.1 Geschäftsprozesse und Workflows

Die für Unternehmen zentrale Rolle der Geschäftsprozesse hat in den letzten Jahren zu einer Vielzahl von Definitionen des Begriffs Geschäftsprozess geführt. In DIN 66201 wird der abstraktere Begriff **Prozess** definiert als eine Gesamtheit von aufeinander einwirkenden Vorgängen in einem System, durch die Materie, Energie oder Information umgeformt oder gespeichert wird. Die Informatik fasst den Begriff Prozess formal als die Abfolge von Aktionen in einem Zustandsraum auf [Ba00]. Der aus der Wirtschaft bereits lange bekannte Begriff des Geschäftsprozesses wurde Anfang der 90er Jahre im Kontext der Informatik etabliert [OW+03]. In diesem Zeitraum entstanden unter anderem durch Hammer und Champy [HC93], van der Aalst und van Hee [AH02] oder Davenport [Da93] unterschiedliche Definitionen eines Geschäftsprozesses aus Sicht der Informatik [RS04]. Davenport definiert in [Da93] einen **Geschäftsprozess** wie folgt:

> *"A process is a thus a specific ordering of working activities across time and place, with a beginning, an end, and clearly identified inputs and outputs: a structure for action."*

Diese Definition kann als Grundlage für viele weitere teils verfeinerte Definitionen verstanden werden, die unter anderem in [RS04] oder [OW+03] zu finden sind. Neben der Organisationsstruktur

eines Geschäftsprozesses adressiert die *Workflow Management Coalition* (WfMC) in ihrer Definition das einem Geschäftsprozess zugrunde liegende Geschäftsziel [WfMC-T&G3]:

> *"A set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships."*

Das einem Geschäftsprozess zugrunde liegende Geschäftsziel grenzt einen Geschäftsprozess damit auch von einem einmalig durchzuführenden Projekt ab. Die folglich gewünschte Wiederholbarkeit der Durchführung eines Geschäftsprozesses erfordert dessen präzise Definition. Die genaue Abfolge der Aktivitäten eines Geschäftsprozesses, die ihn startenden bzw. beendenden Ereignisse, die beteiligen organisatorischen Einheiten etc. werden mittels einer **Prozessdefinition** spezifiziert.

Die in obiger Definition allgemein angesprochene Menge der Aktivitäten eines Geschäftsprozesses muss präziser differenziert werden. Kann eine Aktivität teilweise oder vollständig durch IT unterstützt werden, so liegt nach [WfMC-T&G3] eine **automatisierte Aktivität** vor. Dementsprechend bezeichnet [WfMC-T&G3] eine Aktivität als **manuelle Aktivität**, wenn diese nicht durch IT unterstützt werden kann. Das Führen eines Beratungsgesprächs kann daher beispielsweise als eine manuelle Aktivität spezifiziert werden, die von einem Menschen ohne die Unterstützung der IT ausgeführt werden muss. Diejenigen Teile eines Geschäftsprozesses, die nur aus automatisierten, im Sinne von IT-unterstützten Aktivitäten bestehen, werden als **Workflow-Prozesse** oder kurz **Workflows** bezeichnet [RS04]. Workflows als Untermenge der Geschäftsprozesse können daher durchgängig durch entsprechende IT-Systeme (vollständig oder teilweise automatisiert) ausgeführt und in ihrer Ausführung überwacht werden. Nach [Aa98] muss demnach ein Workflow aus drei unterschiedlichen Dimensionen betrachtet werden, die in Abbildung 10 dargestellt sind.

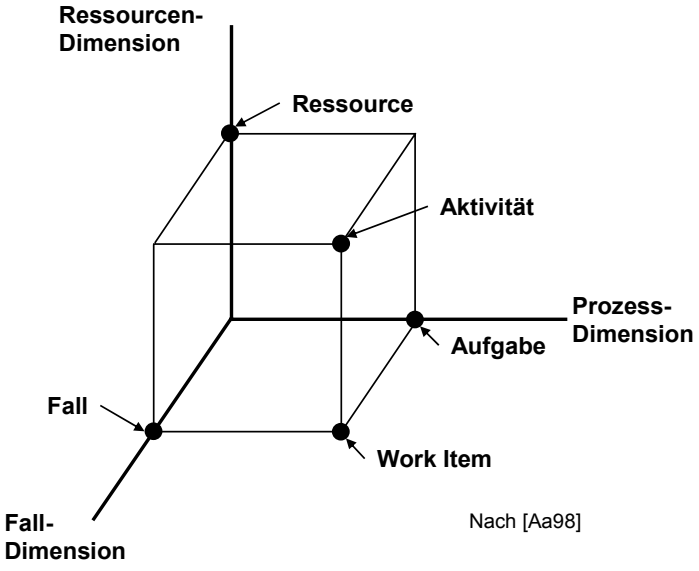

Abbildung 10: Dreidimensionale Sicht eines Workflows

Die Ausführung eines Workflows wird immer dann initiiert, wenn ein neuer **Fall** (engl. *Case*) zu bearbeiten ist. Dementsprechend wird für jeden Fall ein geeigneter Workflow instanziiert und ausgeführt. Ein Workflow besteht nach van der Aalst [Aa98] in der Prozessdimension zunächst aus einer Aneinanderreihung von Aufgaben (engl. *Tasks*), die zur Abarbeitung eines Falls notwendig sind. Da diese Abarbeitung in einer festgelegten Reihenfolge stattfinden muss, ist jede Aufgabe mit Bedingungen (engl. *Conditions*) verknüpft, die zu einem booleschen Wert ausgewertet werden können. Somit

kann durch die Spezifikation von Vor- oder Nachbedingungen der kausale Zusammenhang einzelner Aufgaben untereinander spezifiziert werden.

Da ein Workflow immer im Kontext eines Falles ausgeführt wird, stellt eine Aufgabe in Relation zu einem Fall ein sogenanntes ***Work Item*** dar. *Work Items* wiederum müssen durch eine **Ressource** bearbeitet werden. Eine Ressource kann dabei eine Person, wie beispielsweise ein Arbeiter, Angestellter oder Teilnehmer, aber auch eine durch van der Aalst als *machine* bezeichnete technische Ressource, wie etwa ein Drucker oder Faxgerät, sein.

Um die Zuteilung von Ressourcen zu *Work Items* zu vereinfachen, werden Ressourcen in Klassen eingeteilt. Eine **Ressourcenklasse** ist demnach eine Gruppe von Ressourcen, welche die gleichen oder ähnlichen Charakteristika aufweisen. Erfolgt die Zuteilung einer Ressource zu einer Gruppe aufgrund der Fähigkeiten der einzelnen Mitglieder der Gruppe, so wird diese Gruppe als **Rolle** (engl. *Role*) bezeichnet. Erfolgt diese Klassifikation jedoch auf Basis der Struktur einer Organisation, so werden diese Ressourcenklassen als **organisatorische Einheiten** (engl. *Organizational Units*) bezeichnet. Wird nun eine Aufgabe in Bezug zu einem Fall und einer Ressource betrachtet, so spricht [Aa98] von einer **Aktivität** (engl. *Activity*). Die WfMC definiert auf Basis dieser Grundbegriffe einen Workflow in [WfMC-T&G3] als:

> *"The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules."*

Workflow-Management-Systeme ermöglichen somit die Ausführung von Aktivitäten, in dem sie Prozessdefinitionen für jeden Fall erneut instanziieren und jede Aktivität durch geeignete Ressourcen untermauern. Heute findet sich in der Literatur auch der abstraktere Begriff der **Prozessausführungsumgebungen** zur Bezeichnung eines IT-Systems, das Prozessdefinitionen interpretieren, instanziieren und ausführen kann. Eine Prozessdefinition kann, da sie für jeden Fall neu instanziiert wird, innerhalb einer Prozessausführungsumgebung auch in mehreren **Prozessinstanzen** existieren. Eine Prozessinstanz wiederum setzt sich aus unterschiedlichen **Aktivitätsinstanzen** zusammen, die von Ressourcen für je einen konkreten Fall abgearbeitet werden.

Als Verfeinerung zur Definition von van der Aalst soll für diese Arbeit in Anlehnung an die Definition der WfMC eine technische Ressource durch eine **Anwendung** realisiert werden. Kann eine Aktivitätsinstanz vollständig automatisiert abgearbeitet werden, so ruft diese eine entsprechende Anwendung auf. Bedarf es der menschlichen Interaktion zur Abarbeitung der Aktivitätsinstanz, so wird eine Aufgabe für einen Menschen erstellt. Die WfMC definiert diese **Aufgabe** (engl. *Task*) folgendermaßen [WfMC-T&G3]:

> *"The representation of the work to be processed (by a workflow participant) in the context of an activity within a process instance."*

Diese Definition muss für diese Arbeit weiter verfeinert werden. Im Fokus der Betrachtung stehen Menschen, die in einem Geschäftsprozess resp. Workflow mit einem IT-System interagieren. Dementsprechend nutzt ein Mensch ein IT-System, um seine Aufgabe erfüllen zu können. Um diesen zentralen Aspekt der Benutzung eines IT-Systems hervorzuheben, spricht diese Arbeit im Folgenden immer dann von einer **Benutzeraufgabe**, wenn ein Mensch zur Abarbeitung einer Aufgabe ein IT-System nutzt. Abbildung 11 fasst die bisher eingeführten Begriffe und ihre Beziehungen in Anlehnung an die Definitionen der WfMC zusammen.

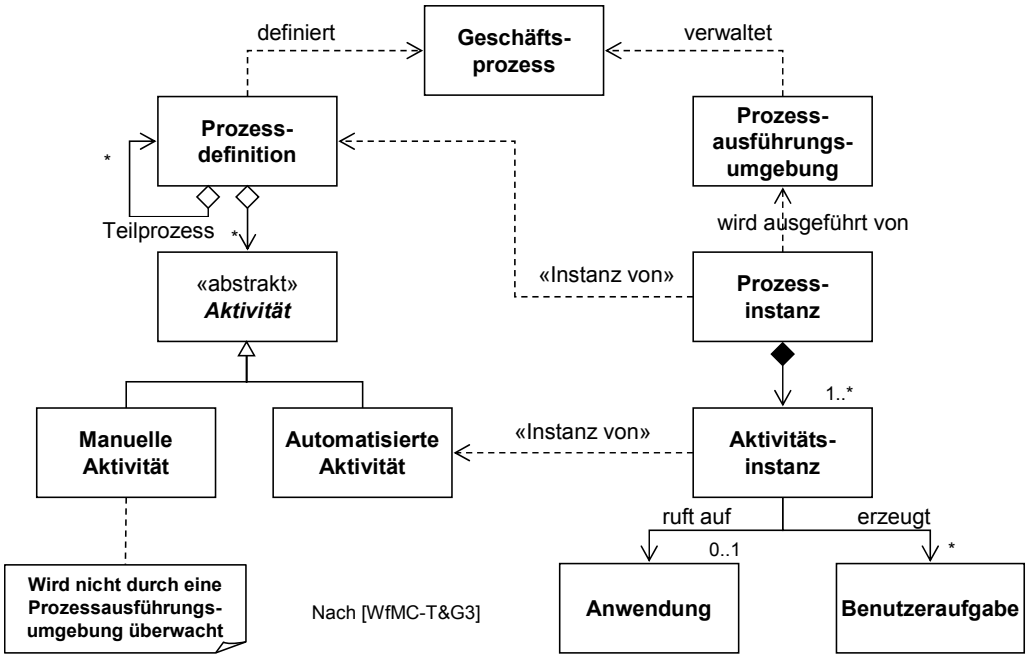

Abbildung 11: Abstrakte Syntax eines Geschäftsprozesses

Die in einem Workflow spezifizierten Aktivitäten können entweder durch eine Anwendung (automatisierte Aktivität) oder durch einen Menschen als Nutzer eines IT-Systems (Benutzeraufgabe) ausgeführt werden. Eine durch einen Benutzer durchzuführende Aktivität wird durch die WfMC ursprünglich als *Work Item* (dt. Arbeitseinheit), der Benutzer an sich als ***Workflow Participant*** bezeichnet. Der hierzu synonym angegebene Begriff ***Human Task*** findet sich ebenso in neueren Arbeiten wie etwa [AA+07b]. Im weiteren Verlauf wird daher weiter der deutsche Begriff „Benutzeraufgabe“ verwendet.

Eine Benutzeraufgabe beschreibt eine Arbeitseinheit, die durch einen Menschen unter Benutzung eines IT-Systems ausgeführt wird und dabei von der Bestätigung einer Entscheidung bis zur Eingabe komplexer Daten reichen kann. Obwohl die eigentliche Bearbeitung einer Benutzeraufgabe durch einen Menschen erfolgt, handelt es sich bei Benutzeraufgaben nicht um manuelle Aktivitäten (vgl. Abbildung 11), da deren Bereitstellung, Koordination und Überwachung durch ein IT-System unterstützt wird, mit welchem der Mensch folglich interagieren muss.

Einige Workflow-Management-Systeme unterscheiden bei menschlichen Interaktionen zusätzlich zwischen eigentlichen Benutzeraufgaben und sogenannten **Benachrichtigungen** (engl. *Notifications*). Bei Benutzeraufgaben wird vom Benutzer die Eingabe von Daten erwartet und dadurch die Ausführung des Workflows so lange angehalten, bis der Benutzer die Erledigung der Benutzeraufgabe signalisiert und alle Nachbedingungen der Benutzeraufgabe erfüllt sind. Im Gegensatz zu dieser synchronen Ausführung erfordern Benachrichtigungen keine Reaktion des Benutzers und können daher asynchron ausgeführt werden. Gelegentlich werden Benutzeraufgaben und Benachrichtigungen wie beispielsweise in [AA+07a] unter dem Obergriff *People Activity* zusammengefasst. Da sich beide Konzepte im Hinblick auf die Ausführung eines Workflows jedoch wesentlich unterscheiden, betrachtet diese Arbeit eine Benutzeraufgabe und eine Benachrichtigung weiterhin getrennt voneinander.

2.2.2 Workflow-Management-Systeme

Das Aufkommen der Workflow-Management-Systeme Anfang der 90er Jahre führte zu einem Paradigmenwechsel auf dem Gebiet der Informationstechnik [RS04]. Konnten zuvor nur einzelne Arbeitseinheiten durch IT-Systeme unterstützt werden, so wurde mit den Workflow-Management-Systemen

die Unterstützung ganzer Geschäftsprozesse und die dazu notwendige Steuerung der Arbeitsflüsse möglich. Dieser wesentliche Mehrwert der Workflow-Management-Systeme führte zu einer raschen Umsetzung der Konzepte in Form vieler unterschiedlicher Implementierungen, deren Interoperabilität zunächst nicht gegeben war. Daher verabschiedete die WfMC 1995 mit dem *Workflow Reference Model* (WRM) eine Referenzarchitektur für Workflow-Management-Systeme [WfMC-WRM1.1], welche die gemeinsamen Charakteristika damals bestehender Lösungen zusammenfasste und so einen klaren Schwerpunkt auf die Interoperabilität und Integrationsfähigkeit künftiger Lösungen setzte. In dieser Spezifikation definiert die WfMC ein **Workflow-Management-System** als

> *"a system that completely defines, manages and executes "workflows" through the execution of software whose order of execution is driven by a computer representation of the workflow logic."*

Gemäß dieser Definition zeigt die WfMC in ihrer in Abbildung 12 dargestellten Referenzarchitektur unterschiedliche Softwarekomponenten auf, die zur Ausführung eines Workflows benötigt werden.

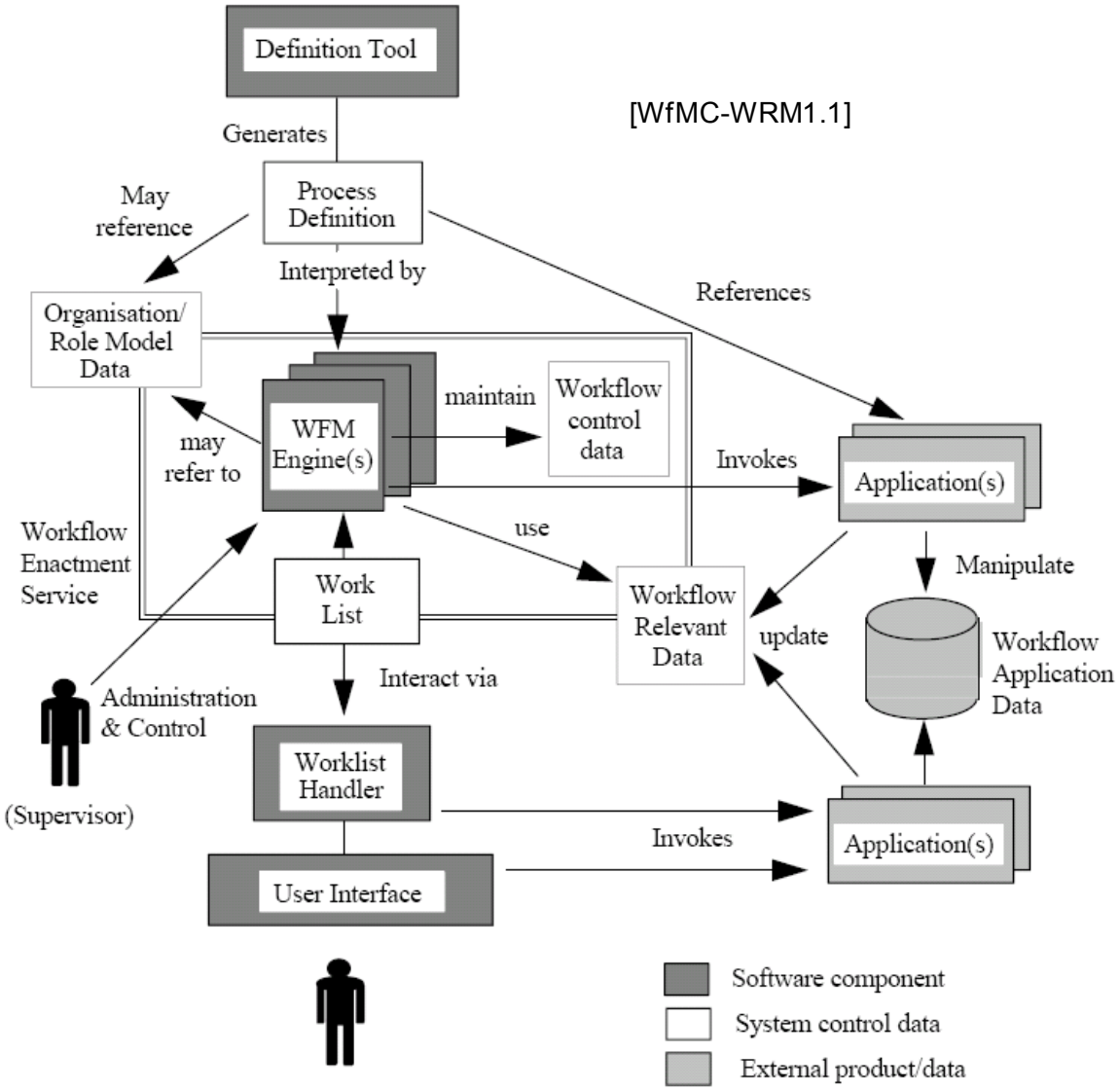

Abbildung 12: Referenzarchitektur eines Workflow-Management-Systems

In einem Definitionswerkzeug (engl. *Definition Tool*) werden ausführbare Prozessbeschreibungen erzeugt, die dann durch die zentrale Workflow-Management-Ausführungsmaschine (engl. *Workflow*

Management Engine, WFM-*Engine*) ausgeführt wird. Die WFM-*Engine* wiederum ruft unterschiedliche externe Anwendungen auf und zeigt über einen *Worklist Handler* dem Menschen über eine Benutzerschnittstelle seine in der *Worklist* vorgehaltenen Benutzeraufgaben an. Die zuletzt angesprochenen Softwarekomponenten sind von zentraler Bedeutung für diese Arbeit und werden daher im Folgenden einzeln betrachtet.

2.2.2.1 Workflow-Ausführungsumgebung

Die von der WfMC als WFM-*Engine* bezeichnete Softwarekomponente bildet das Herzstück eines Workflow-Management-Systems und ist für die eigentliche Ausführung und Überwachung der Workflows zuständig. Die WFM-*Engine* kann Prozessdefinitionen interpretieren und diese zur Ausführung bringen, weshalb im Folgenden der deutsche Begriff „Workflow-Ausführungsumgebung“ für diese Softwarekomponente verwendet wird. Die Workflow-Ausführungsumgebung ruft unterschiedliche externe Anwendungen auf (*invoke*), die zur Abarbeitung einzelner Arbeitseinheiten eines Workflows benötigt werden. Ferner greift die Workflow-Ausführungsumgebung auf das Organisations- und Rollenmodell des Unternehmens, in dem das Workflow-Management-System eingesetzt wird, zu, um hierüber Arbeitseinheiten, die durch einen Menschen in Form von Benutzeraufgaben ausgeführt werden sollen, an die geeigneten Rollen delegieren zu können. Ferner kann die Workflow-Ausführungsumgebung über sogenannte *Worklists* (dt. Arbeitslisten) den vorhandenen menschlichen Ressourcen ihre zu erledigenden Arbeitseinheiten zuordnen.

Im Kontext der dienstorientierten Architekturen, die eine wesentliche Grundlage dieser Arbeit bilden, haben sich sogenannte BPEL-*Engines* als Workflow-Ausführungsumgebungen für Workflows etabliert. Die *Business Process Execution Language* (BPEL) [OASIS-WS-BPEL2] dient als Spezifikationssprache für Workflows. BPEL-*Engine*s als Workflow-Ausführungsumgebungen für Workflows werden daher in Abschnitt 2.4 erneut aufgegriffen.

2.2.2.2 Arbeitsliste

Die Workflow-Ausführungsumgebung eines Workflow-Management-Systems ordnet die einzelnen Arbeitseinheiten eines Workflows geeigneten technischen oder menschlichen Ressourcen zu, die als Workflow-Teilnehmer (engl. *Workflow Participant*) an der Ausführung des Workflows beteiligt sind. Um diese Zuordnung durchführen zu können, sieht das *Workflow Reference Model* eine Arbeitsliste (engl. *Worklist*) vor, die in [WfMC-T&G3] wie folgt definiert wird:

> *“A list of work items associated with a given workflow participant (or in some cases with a group of workflow participants who may share a common Worklist). The worklist forms part of the interface between a workflow engine and the worklist handler.”*

Eine Arbeitsliste ist demnach für jeden Workflow-Teilnehmer individuell. In der Literatur finden sich zu *Worklist* synonym verwendete Begriffe wie etwa *Tasklist*, *To-Do-List*, *Work Queue*.

2.2.2.3 Benutzeraufgabenverwaltung

Während die Arbeitsliste als Teil der Workflow-Ausführungsumgebung die Persistierung der Arbeitseinheiten für alle Workflow-Teilnehmer übernimmt, muss durch eine weitere Softwarekomponente die Verbindung zwischen der Arbeitsliste und den menschlichen Workflow-Teilnehmern über die Benutzerschnittstelle hergestellt werden. Diese Aufgabe übernimmt eine als *Worklist Handler* bezeichnete

Softwarekomponente, die Schnittstellen für die Workflow-Ausführungsumgebung und die Benutzerschnittstelle bereitstellt. Die WfMC definiert den *Worklist Handler* als [WfMC-T&G3]:

> *"A software component that manages the interaction between the user (or group of users) and the worklist maintained by the workflow engine. In enables work items to be passed from the workflow management system to users and notifications of completion or other work status conditions to be passed between the user and the workflow management system."*

Da der *Worklist Handler* im Bezug auf menschliche Workflow-Teilnehmer diejenigen Arbeitseinheiten verwaltet, die in Form von Benutzeraufgaben abgearbeitet werden müssen, wird der *Worklist Handler* im weiteren Verlauf als „Benutzeraufgabenverwaltung" bezeichnet.

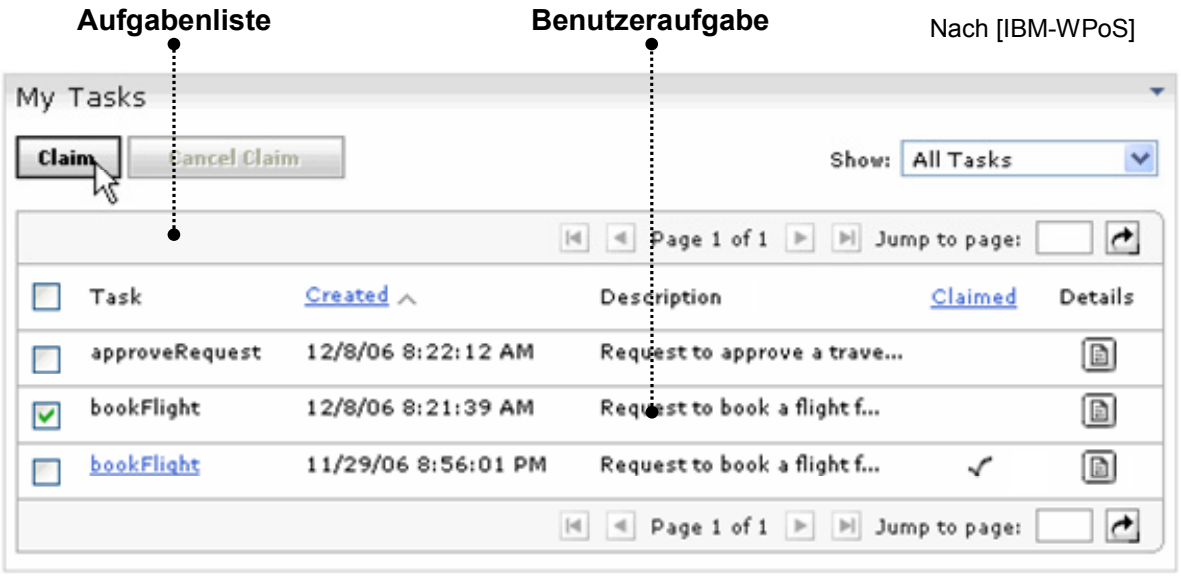

Abbildung 13: Konkretes Beispiel für die Visualisierung einer Aufgabenliste

Bestehende Implementierungen von Benutzeraufgabenverwaltungen, wie beispielsweise im WebSphere Portal Server von IBM [IBM-WPoS], visualisieren die durch einen menschlichen Workflow-Teilnehmer abzuarbeitenden Benutzeraufgaben häufig in Form einer tabellarischen Aufgabenliste (engl. *Tasklist*), aus welcher der Benutzer eine Benutzeraufgabe zur Bearbeitung auswählen kann. Abbildung 13 zeigt als Beispiel die Aufgabenliste des WebSphere Portal Servers, zu welcher der Benutzer über die Benutzerschnittstelle Zugang erhält.

2.2.2.4 Lebenszyklus einer Benutzeraufgabe

Zur Ausführung eines Workflows instanziiert eine Workflow-Ausführungsumgebung die zugeordnete Prozessdefinition und arbeitet anschließend diese Instanz schrittweise ab. Das gleiche Grundprinzip kommt bei der Abarbeitung einer Benutzeraufgabe zum Einsatz. Erreicht die Workflow-Ausführungsumgebung bei der Abarbeitung einer Workflow-Instanz eine Aufgabe, die als Benutzeraufgabe spezifiziert wurde, so erzeugt die Workflow-Ausführungsumgebung eine neue Instanz dieser Benutzeraufgabe. Der Lebenszyklus einer Benutzeraufgabeninstanz beginnt daher mit ihrer Erstellung durch eine Workflow-Ausführungsumgebung und endet mit der Vervollständigung durch einen Benutzer. Jede Benutzeraufgabeninstanz kann während ihrer Existenz eine Reihe verschiedener Zustände annehmen. Diese Zustände werden nach [RA+05] als Lebenszyklus einer Benutzeraufgabe in Abbildung 14 in Form eines UML-Zustandsdiagramms dargestellt.

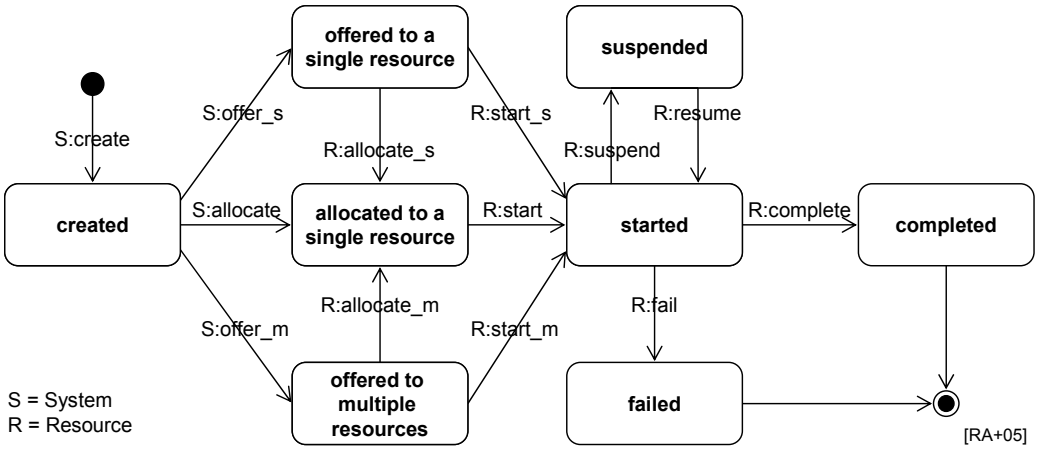

Abbildung 14: Lebenszyklus einer Benutzeraufgabe

Der Lebenszyklus einer Benutzeraufgabe wird durch Zustände und Übergänge zwischen diesen beschrieben. Den Bezeichnungen der Übergänge ist ein „S“ für System bzw. ein „R“ für Ressource vorangestellt, um zu kennzeichnen, ob der Zustandswechsel durch das System oder durch die Ressource Mensch als Benutzer eingeleitet wird. Wird eine Benutzeraufgabe von der Workflow-Ausführungsumgebung initiiert (`S:create`), befindet sich diese im Zustand `created` (erzeugt). In diesem Zustand sind noch keine konkreten Benutzer zur Bearbeitung der Benutzeraufgabe ermittelt oder bekannt. Die Auswahl einer geeigneten Ressource geschieht durch den Übergang in einen der drei möglichen Folgezustände. Wird die Benutzeraufgabe genau einem Benutzer zugewiesen, wechselt diese in den Zustand `offered to a single resource` (einer Ressource angeboten). Bei mehreren möglichen Benutzern wechselt die Benutzeraufgabe dagegen in den Zustand `offered to multiple resources`. In diesen beiden Zuständen ist die Benutzeraufgabe in den Aufgabenlisten der zugewiesenen Benutzer verfügbar und kann nun von dem Benutzer (`R:start_s`) bzw. von einem der möglichen Benutzer (`R:start_m`) entweder direkt gestartet und bearbeitet oder zur späteren Bearbeitung zunächst reserviert werden (`R:allocate_s` oder `R:allocate_m`). Eine durch einen Benutzer reservierte Benutzeraufgabe befindet sich danach im Zustand `allocated to a single resource` und kann nur noch von diesem Benutzer gestartet werden (`R:start`). Genau in diesem Zustand befindet sich die dritte Benutzeraufgabe in der in Abbildung 13 (Seite 31) gezeigten Aufgabenliste, wie an dem gesetzten Auswahlfeld in der Spalte *Claimed* (dt. beansprucht) zu erkennen ist. Zugeordnete oder gestartete Benutzeraufgaben stehen anderen möglichen Benutzern der gleichen Ressourcenklasse nicht mehr zur Verfügung und werden in deren Aufgabenlisten dementsprechend nicht mehr geführt. Der dritte mögliche Übergang von Zustand `created` aus ist der direkte Wechsel in den Zustand `allocated to a single resource` über den Übergang `S:allocate`. Dieser Übergang spezifiziert eine vom System ausgeführte direkte Zuweisung und Reservierung der Benutzeraufgabe zu einem Benutzer. Auf diese Weise wird sichergestellt, dass eine Benutzeraufgabe direkt einer dedizierten Ressource zugewiesen wird. Ist eine Benutzeraufgabe gestartet, befindet sie sich im Zustand `started`. Die Bearbeitung einer Benutzeraufgabe kann durch die zuständigen Benutzer unterbrochen werden (`R:suspend`) und wechselt dazu in den Zustand `suspended`. Aus diesem Zustand heraus kann die Benutzeraufgabe durch den Benutzer zu einem späteren Zeitpunkt wieder aufgenommen werden (`R:resume`). Bringt ein Benutzer eine ihm zugewiesene Benutzeraufgabe nicht erfolgreich zum Abschluss, erfolgt der Übergang dieser Benutzeraufgabe (`R:fail`) in den Endzustand `failed` und gilt damit als nicht erfolgreich abgeschlossen. Der Zustand `completed` hingegen kennzeichnet einen Endzustand, der durch den erfolgreichen Abschluss einer Benutzeraufgabe erreicht werden kann (`R:complete`).

2.2.3 Workflow-Perspektiven und Workflow-Muster

Workflows stellen in der ursprünglichen Definition durch die WfMC die Automatisierung von Geschäftsprozessen dar, in welchen nach festgelegten Regeln Dokumente, Informationen oder Aufgaben zwischen den Workflow-Teilnehmern weitergegeben werden (vgl. Abschnitt 2.2.1). Obwohl Unternehmen unterschiedliche Workflows ausführen, stellen sich dennoch immer wieder gleiche oder ähnliche Anforderungen an die Spezifikation und die Ausführung von Workflows. Mit diesen Anforderungen beschäftigt sich seit 1999 die *Workflow Patterns Initiative* [AH-WPI], indem sie verschiedene Modellierungssprachen hinsichtlich deren Eignung und Ausdrucksmächtigkeit zur Spezifikation von Workflows und Geschäftsprozessen untersucht und bewertet [AH+03]. Hierzu betrachtet die *Workflow Patterns Initiative* verschiedene Perspektiven der Workflows und stellt unterschiedliche Workflow-Muster für diese Perspektiven bereit, die immer wiederkehrende Anforderungen an Workflows ähnlich den aus der Softwaretechnik bekannten Analyse- oder Entwurfsmustern [Ei07] manifestieren. Die folgenden vier Perspektiven werden dazu eingeführt:

- Die **Kontrollfluss- / Prozessperspektive** (engl. *control-flow / process perspective*) beschreibt einzelne Aktivitäten und die Reihenfolge ihrer Ausführung. Hierbei kommen unterschiedliche Muster wie beispielsweise die Sequenz, die Vereinigung oder die Parallelität zum Einsatz, mit welchen die Reihenfolge der Abarbeitung festgelegt werden kann. Aktivitäten können dabei als atomare Arbeitseinheiten oder als Verbund einzelner Aktivitäten spezifiziert werden. Eine detaillierte Betrachtung aller aktuellen Workflow-Muster dieser Perspektive sowie eine Bewertung aktueller Sprachen zur Spezifikation von Workflows hinsichtlich deren Grad der Unterstützung dieser Workflow-Muster findet sich in [RH+06, AH+03, AH02].

- Die **Datenperspektive** umfasst zwei Arten von Daten, die innerhalb eines Workflows auftreten. Zum einen dienen Geschäftsdaten zur Spezifikation von Geschäftsobjekten, die in einem Workflow verarbeitet werden. Zum anderen nutzt ein Workflow Daten wie beispielsweise lokale Variable, die lediglich zur Steuerung der Ausführung des Workflows benötigt werden. In [RH+05] ist eine ausführliche Betrachtung der in dieser Perspektive angesiedelten Workflow-Muster sowie die Bewertung der Unterstützung dieser Workflow-Muster durch bestehende Sprachen zur Workflow-Modellierung zu finden.

- Die **Ressourcenperspektive** stellt den Bezug zu der Organisationsstruktur her, in welcher der Workflow ausgeführt wird [RA+05, RH+04]. Sie liefert daher einen Einblick in die unterschiedlichen Ressourcenklassen, die zur Ausführung eines Workflows beitragen. Einige dieser Ressourcenklassen werden durch Rollen gebildet, die durch Menschen belegt werden. Daher steht diese Perspektive in direktem Bezug zur Benutzerinteraktion. Zwei Workflow-Muster der Ressourcenperspektive, die im weiteren Verlauf dieser Arbeit verwendet werden, betrachtet der folgende Abschnitt detailliert.

- Die **Perspektive der Ausnahmebehandlung** adressiert die Tatsache, dass während der Ausführung eines Workflows Ausnahmen auftreten können, die zu Abweichungen von der eigentlich vorgesehenen Ausführungsreihenfolge führen können. Um im Falle einer Ausnahme mit der Ausführung eines Workflows in keinen undefinierten Zustand zu geraten, müssen entsprechende Ausnahmebehandlungen spezifiziert werden. Eine vollständige Übersicht aller Workflow-Muster dieser Perspektive gibt [RA+06b].

Der Mensch als Benutzer der IT stellt eine Ressource dar, deren Handlungen im Vergleich zu einer technischen Ressource nicht vorhersagbar sind. Wird einer menschlichen Ressource eine Aufgabe

zugeteilt, dann kann die Bearbeitung dieser Aufgabe unnötig lange dauern, da der entsprechende Benutzer beispielsweise gerade nicht verfügbar ist oder die Aufgabe nicht bearbeiten möchte. In diesen Fällen stockt, bedingt durch den synchronen Aufruf einer Benutzeraufgabe (vgl. Abschnitt 2.2.1), die Abarbeitung des gesamten Workflows und wirkt sich nachteilig auf die Ausführungsdauer des Workflows aus. Demnach sollen im Folgenden Auszüge der Workflow-Muster für die Ressourcenmodellierung und Ausnahmebehandlung gesondert betrachtet werden, die in solchen Situationen eine Fortführung des Workflows sicherstellen können. Diese nehmen im weiteren Verlauf der Arbeit eine besondere Stellung ein, da die Ressourcenperspektive eines Workflows durch die in Kapitel 4 entwickelten Metamodelle in den Mittelpunkt der Arbeit gerückt werden.

2.2.3.1 Ressourcenmuster

Wie bereits deutlich gemacht, kommen in einem Workflow abstrakt betrachtet technische und menschliche Ressourcen zum Einsatz [AH+03]. Mit Fokus auf menschliche Ressourcen stellen Russel et al. in [RA+05, RH+04] insgesamt 43 Workflow-Muster für die Ressourcenperspektive vor, welche sie als *Workflow Resource Patterns* bezeichnen und in sieben Gruppen unterteilen. Als eine dieser sieben Gruppen definieren beispielsweise sogenannte *Creation Patterns* (dt. Erzeugungsmuster) wie die Identifizierung der Benutzer erfolgt, die zu einer Benutzeraufgabe zugeordnet werden können. Sie werden zur Entwurfszeit spezifiziert und zur Laufzeit bei der Erstellung von Benutzeraufgaben (`S:create`, vgl. Abbildung 14) ausgewertet. Hier findet sich unter anderem das Muster *Role-Based Allocation*, das bereits zur Entwurfszeit festlegt, dass eine Benutzeraufgabe nur durch eine spezielle Rolle ausgeführt werden darf. Bestehende Modellierungssprachen, wie etwa die UML oder BPMN, unterstützen dieses Muster, indem einer Partition bzw. *Lane* Rollen zugeordnet werden können (vgl. dazu [Oe06] und [Al08]).

Eine weitere Gruppe bilden die neun *Detour Patterns* (dt. Umleitungsmuster). Diese behandeln abweichende Übergänge zwischen den Zuständen des Lebenszyklus einer Benutzeraufgabe, die sowohl vom System als auch vom Benutzer initiiert werden können. Die meisten der durch die *Detour Patterns* genutzten Übergänge gehören nicht zum Standard-Lebenszyklus einer Benutzeraufgabe und sind daher auch nicht in Abbildung 14 zu finden. Zwei *Detour Patterns* seien im Folgenden detailliert beleuchtet. Eine vollständige Betrachtung findet sich in [RA+05, RH+04].

Delegation

Das Workflow-Muster *Delegation* beschreibt die Möglichkeit der Delegation einer reservierten, aber noch nicht gestarteten Benutzeraufgabe eines Benutzers an einen weiteren Benutzer, da der eigentliche Eigentümer der Benutzeraufgabe beispielsweise nicht imstande ist, die Benutzeraufgabe selbst durchzuführen. Eine Delegation wird über die Aufgabenliste desjenigen Benutzers initiiert, welcher die Benutzeraufgabe für sich reserviert hat. Die Benutzeraufgabenverwaltung entfernt nach Erhalt einer entsprechenden Delegationsanweisung die Aufgabe aus der Aufgabenliste des ersten Benutzers und fügt sie der Aufgabenliste des delegierten Benutzers als eine bereits reservierte Benutzeraufgabe hinzu. Folglich ändert sich nur der zugewiesene Benutzer, nicht aber der Zustand der Benutzeraufgabe. Dieses Workflow-Muster wird durch die bekannten grafischen Modellierungssprachen UML und BPMN nicht unterstützt [RA+06a, WA+06].

Eskalation

Das Eskalationsmuster gestattet der Benutzeraufgabenverwaltung, Benutzer über ein Ereignis durch eine Benachrichtigung zur Laufzeit einer Benutzeraufgabeninstanz zu informieren oder der Benutzer-

aufgabeninstanz neue Benutzer zuzuweisen. Eine solche Eskalation wird typischerweise von der Benutzeraufgabenverwaltung nach Ablauf einer bestimmten, vorher festgelegten zeitlichen Restriktion (engl. *Deadline*) initiiert. Details über solche Fristen und die involvierten Benutzer müssen sich häufig bereits zur Entwurfszeit der Benutzeraufgabe festlegen lassen. Eskalationen können sowohl im gestarteten (`started`), im reservierten (`allocated`) als auch in beiden Angebotszuständen (`offered`) einer Benutzeraufgabe erfolgen (vgl. Abbildung 14). Ferner kann eine Eskalation eine Benutzeraufgabe in einen der vorherigen Zustände überführen, beispielsweise vom gestarteten Zustand in den reservierten oder in den Angebotszustand. Eine erneute Zuweisung der Benutzeraufgabe kann durch die Eskalation sowohl an einen einzelnen als auch an mehrere Benutzer erfolgen. Wurde eine gestartete Benutzeraufgabe nach Ablauf einer gesetzten Frist nicht abgeschlossen, so kann die Benutzeraufgabenverwaltung die Benutzeraufgabe einer Benutzergruppe oder einem in der Hierarchie der Organisationseinheit höher stehenden Benutzer zuweisen. Auch dieses Workflow-Muster wird weder durch UML noch durch BPMN unterstützt [RA+06a, WA+06].

Neben den beiden vorgestellten Workflow-Mustern können viele weitere Workflow-Muster nicht durch die heute vorhandenen Sprachen zur Modellierung aber auch zur Ausführung von Workflows spezifiziert werden [WA+06, RA+06a]. Da diese Workflow-Muster jedoch wichtige Anforderungen der Geschäftsebene widerspiegeln, müssen vorhandene Sprachen entsprechend erweitert werden, um die jeweils wichtigen Workflow-Muster zu unterstützen. Stehen diese Workflow-Muster über geeignete Sprachelemente zur Verfügung, so können sie in einem entsprechenden Entwicklungsvorgehen eingesetzt werden, um die Geschäftsanforderungen, die sich beispielsweise auf die Ressourcenperspektive eines Workflows auswirken, zu spezifizieren.

Die Erkenntnis, dass Geschäftsprozesse die zentralen Artefakte der Wertschöpfung und Wettbewerbsfähigkeit eines Unternehmens darstellen, hat eine zunehmende Orientierung der IT an den Geschäftsprozessen und deren Anforderungen zur Folge. Die dazu notwendige enge Verzahnung der Geschäftsprozesse und der IT erfordert ein Umdenken aus Sicht der IT. Zwar können, wie in Abbildung 11 angedeutet, nicht alle Aktivitäten eines Geschäftsprozesses durch IT unterstützt werden. Dennoch kann mit einer zunehmenden Durchdringung der einzelnen Fachbereiche der Unternehmen durch IT ein immer größerer Teil der Geschäftsprozesse als Workflows IT-gestützt realisiert werden. Moderne Entwicklungsvorgehen, wie die geschäftsgetriebene Entwicklung, forcieren daher ein Umdenken in der Entwicklungsphilosophie und stellen die Geschäftsprozesse in den Mittelpunkt der Entwicklung. Da diese Arbeit ein an Geschäftsprozessen orientiertes, modellgetriebenes Verfahren zur Unterstützung von Benutzerinteraktion in dienstorientierten Architekturen verfolgt, stellt das Vorgehensmodell der geschäftsgetriebenen Entwicklung eine zentrale Grundlage für diese Arbeit dar und wird entsprechend im folgenden Abschnitt beleuchtet.

2.2.4 Business-Driven Development

Unternehmen beschäftigen sich seit geraumer Zeit damit, die eigenen Geschäftsprozesse zu analysieren, zu formalisieren und daraus abgeleitet Optimierungen an den Geschäftsprozessen zu erreichen [CS04]. Die stetige Evolution der Geschäftsprozesse mit dem Ziel der Optimierung soll letzten Endes die Kosten der Ausführung eines Geschäftsprozesses senken und damit die Wettbewerbsfähigkeit des Unternehmens sichern. Die Umsetzung einer solchen Optimierung auf die unterstützende IT ist jedoch mit erheblichem Aufwand verbunden. Dieser liegt unter anderem darin begründet, dass die Entwicklung von Geschäftsprozessen und der unterstützenden IT-Systeme trotz umfassender Entwicklungsrahmenwerke wie dem *Rational Unified Process* bisher nicht geeignet unterstützt werden [SK+05, KG+07, Mi05]. Daher ist nach [KR05] eine Anpassung der durch den *Rational Unified Process*

[Kr03] ursprünglich verfolgten *Best Practices* der Softwareentwicklung notwendig, um die zunehmende Fokussierung der Softwareentwicklung auf die Geschäftsprozesse in neuen Vorgehensmodellen reflektieren zu können. Kroll und Royce definieren dementsprechend sechs Schlüsselprinzipien der geschäftsgetriebenen Entwicklung, die in [KR05] ausführlich beschrieben sind und im Zusammenhang mit den im weiteren Verlauf der Arbeit entwickelten Lösungen zur Anwendung kommen:

- ***Adapt the process.*** Es gibt kein Vorgehensmodell, das für alle Softwareprojekte gleichermaßen geeignet ist. Es muss daher für jedes Projekt ein individuell passendes Maß an Kontrolle und Formalität ermittelt und das Vorgehensmodell entsprechend ausgerichtet werden.

- ***Balance competing stakeholder priorities.*** Bei vielen Softwareentwicklungsprojekten stehen Geschäftsanforderungen und zur Entwicklung verfügbare Ressourcen im Widerspruch. Daher empfiehlt es sich oftmals, Teile der benötigten Fachfunktionalität nicht selbst zu entwickeln, sondern diese durch Standardsoftware einzukaufen. Deren Einsatz verkürzt die Entwicklungszeit und senkt damit die Entwicklungskosten.

- ***Collaborate across teams.*** Entscheidend bei der Zusammenarbeit mehrerer Entwicklungsteams ist nicht nur eine gute Kommunikation, sondern auch die gemeinsame Teilhabe am Erfolg oder auch Misserfolg eines Projektes. Dies motiviert alle Beteiligten über die verschiedenen Rollen hinweg und verspricht eine effektivere Arbeitsweise.

- ***Demonstrate value iteratively.*** Um das Risiko einer Fehlentwicklung zu senken und gleichzeitig das Vertrauen der Beteiligten in die Lösung zu stärken, sollte durch eine iterative Entwicklung früh und in aller Regelmäßigkeit der Entwicklungsfortschritt demonstriert werden. Auf diese Weise kann eine Übereinstimmung der Vorstellungen des Entwicklerteams und der Erwartungen des Kunden verifiziert und damit das Risiko einer Fehlentwicklung im Gegensatz zu klassischen Entwicklungsprozessen reduziert werden.

- ***Elevate the level of abstraction.*** Die hohe Komplexität heutiger Projekte zur Softwareentwicklung wirkt sich negativ auf die Produktivität aus. Eine hohe Abstraktionsebene verspricht, diese Komplexität zu reduzieren und damit die Kommunikation im Entwicklungsvorgehen zu verbessern. Eine Möglichkeit zur Reduktion der Komplexität stellt die Wiederverwendung von Geschäftsprozessen, Gebrauchsmustern oder Softwareartefakten dar. Der Grad der Wiederverwendbarkeit kann dabei durch offene Standards wie UML gesteigert werden.

- ***Focus continuously on quality.*** Eine kontinuierliche Verbesserung der Qualität kann durch die Definition hierfür geeigneter Maßnahmen erreicht werden. Zentraler Aspekt zur Umsetzung von qualitätssteigernden Maßnahmen bildet die Übertragung von Qualitätsverantwortung auf das gesamte Entwicklerteam. Hier wirkt sich ein iteratives Entwicklungsvorgehen ebenfalls positiv aus, da ein frühes Testen zur Vermeidung von Fehlern und damit zur Qualitätssicherung beiträgt.

Der Ansatz der geschäftsgetriebenen Entwicklung (*Business-Driven Development*, BDD) wird diesen Schlüsselprinzipien gerecht, indem er die Konzepte der modellgetriebenen Entwicklung aufgreift und Modelle von Geschäftsprozessen als Ausgangspunkt der Softwareentwicklung nutzt. Ziel der geschäftsgetriebenen Entwicklung ist es daher, ein methodisches Entwicklungsvorgehen zu spezifizieren, das eine kontinuierliche Verbesserung der Geschäftsprozesse und deren direkte Ausrichtung an der IT-Unterstützung adressiert [KH+08]. Wird eine Änderung eines Geschäftsprozesses notwendig, so wird diese Änderung im entsprechenden Modell des Geschäftsprozesses formalisiert und dieses

Modell als Ausgangspunkt für das weitere Entwicklungsvorgehen verwendet. Die Trennung zwischen Geschäftsprozess und IT wird aufgehoben; die Entwicklung der Geschäftsprozesse und der unterstützenden IT-Systeme laufen eng verzahnt ab.

Die Phasen der geschäftsgetriebenen Entwicklung werden in enger Anlehnung an den *Rational Unified Process* durch Mitra wie folgt definiert [Mi05]: In der **Modellierungsphase** (engl. *Model Phase*) werden Geschäftsziele und Geschäftsanforderungen identifiziert und die daraus resultierenden Geschäftsprozesse modelliert. Das Geschäftsprozessmodell ist damit das zentrale Verbindungselement zwischen Geschäftsanforderungen und den benötigten IT-Systemen. In der **Entwicklungsphase** (engl. *Develop Phase*) werden die modellierten Geschäftsprozesse schrittweise mittels Transformationen auf eine in die bestehende IT-Landschaft integrierbare Implementierung abgebildet und in der **Inbetriebnahmephase** (engl. *Deploy Phase*) installiert. Diese drei Phasen werden auch durch das in dieser Arbeit konzipierte Entwicklungsvorgehen verfolgt. In der anschließenden **Monitorphase** (engl. *Monitor Phase*) wird die Lösung überwacht, um in der abschließenden **Analyse- und Adaptionsphase** (engl. *Analyze and Adapt Phase*) die zur vollständigen Erfüllung der Geschäftsziele notwendigen Anpassungen ermitteln zu können. Die daraus resultierenden Änderungsanforderungen gehen erneut in die Modellierungsphase ein, womit der Entwicklungskreislauf von vorne beginnt.

Die Fokussierung der Geschäftsprozesse in einem Entwicklungsvorgehen ist nach [BI+06] jedoch nur ein erster, wenn auch wesentlicher Schritt hin zu flexiblen und adaptiven IT-Systemen. Die Fähigkeit eines Unternehmens, adäquat auf rapide und kontinuierliche Änderungen der Rahmenbedingungen reagieren zu können, wird durch Brown et al. neben der geschäftsgetriebenen Entwicklung auf drei zusätzliche Grundprinzipien zurückgeführt [BI+06]:

- ***Composite application assembly.*** Ein Softwaresystem muss aus Komponenten bestehen, die immer wieder zu neuen Lösungen verschaltet werden können, um so den sich ständig verändernden Anforderungen genügen zu können.

- ***Efficiency through systematic reuse.*** Die entwickelten Komponenten müssen systematisch wiederverwendet werden. Durch Wiederverwendung von Komponenten können die Kosten der IT-Unterstützung nachhaltig reduziert werden.

- ***Explicit support for governance to aid compliance.*** Der gesamte Lebenszyklus eines IT-Systems erfordert eine ständige Überwachung der Geschäftsziele und der unterstützenden IT. Zur effizienten Steuerung der Entwicklungszyklen müssen geeignete Aufgaben und Rollen vorgesehen werden. Ein modellgetriebenes Vorgehen hilft auch hier, Entwurfsentscheidungen oder Qualitätsanforderungen langfristig nachvollziehen und iterativ überarbeiten zu können.

In einem zweiten Schritt muss folglich ein besonderes Augenmerk auf die Entwurfsentscheidung bezüglich der zu verwendenden Softwarearchitektur gelegt werden. Diese muss die im Entwicklungsvorgehen angestrebte enge Verzahnung zwischen Geschäftsdomäne und IT unterstützen. Dienstorientierte Architekturen als modernes Entwurfsparadigma für Softwarearchitekturen versprechen, dieser Anforderung gerecht zu werden (vgl. Abschnitt 2.4). Um in diesem Kontext auch die Anforderungen der Benutzerinteraktion berücksichtigen zu können, muss jede Softwarearchitektur Komponenten vorsehen, welche die Interaktion des Menschen mit der IT-Unterstützung ermöglichen. Diese Anforderungen resultieren zunächst in der schlichten Voraussetzung des Vorhandenseins einer Benutzerschnittstelle. Da diese Arbeit sich auf grafische Benutzerschnittstellen fokussiert (vgl. Prämisse P2), wird in Abschnitt 2.3 eine Einführung in die Modellierung und modellgetriebene Entwicklung von grafischen Benutzerschnittstellen gegeben.

2.3 Grafische Benutzerschnittstellen

Um einem menschlichen Benutzer die Interaktion mit einem Anwendungssystem zu ermöglichen, ist eine Benutzerschnittstelle erforderlich [KF93]. Im Laufe der Jahre haben sich unterschiedliche Formen von Benutzerschnittstellen etabliert, die je nach Einsatzgebiet besonders geeignet sind. Um dabei eine Standardisierung der verwendeten Elemente zu erreichen, sind unterschiedliche Direktiven und ISO-Normen entstanden, die in [BV96] übersichtlich zusammengefasst sind und hier nicht weiter verfolgt werden sollen.

Zur Umsetzung dieser Standards und zur Unterstützung der Entwicklung von Benutzerschnittstellen wurden verschiedenste Werkzeuge und Methoden untersucht. Myers et al. geben in [MK+00] einen detaillierten Überblick über diese Werkzeuge und Methoden und konstatieren, dass zukünftige Methoden eine flexible Erzeugung von Benutzerschnittstellen fokussieren müssen. Im Folgenden werden Benutzerschnittstellen näher betrachtet, indem zunächst eine abstrakte, technologieunabhängige Sicht auf die Benutzerschnittstelle eingenommen wird. Bedingt durch die Ziele dieser Arbeit müssen Benutzerschnittstellen in einem modellgetriebenen Entwicklungsvorgehen berücksichtigt und dementsprechend durch Modelle formalisiert werden können. Daher gibt der folgende Abschnitt einen Überblick über heute etablierte Modelltypen zur formalen Spezifikation einer Benutzerschnittstelle.

2.3.1 Einordnung der Benutzerschnittstelle

Wie in [Va96] postuliert, sind der Entwurf und die Entwicklung interaktiver Anwendungen ohne Werkzeugunterstützung im Sinne einer geeigneten Entwicklungsumgebung nicht denkbar. Die Entwicklung der für interaktive Anwendungen zentralen Benutzerschnittstelle muss folglich ebenfalls durch geeignete Methoden und Werkzeuge unterstützt werden. Diese Entwicklung durch die Formalisierung der Benutzerschnittstelle mittels unterschiedlicher Modelle zu erleichtern und die Qualität der erzielten Ergebnisse zu steigern, ist dabei ein wesentliches Ziel vieler Forschungsansätze, das bis heute weiterverfolgt wird (vgl. dazu Abschnitt 3.2). Im Forschungsgebiet des *Computer-Aided Design of User-Interfaces* (CADUI) entstand schnell die Bestrebung, funktionale Benutzerschnittstellen vollautomatisch zu erzeugen [Va96]. Wie [WK+00] jedoch anführt, konnten sich frühe Bestrebungen einer vollautomatischen Erzeugung von Benutzerschnittstellen wie etwa [BF+92] oder [KF93] nicht etablieren, da unter anderem durch die Entwickler eine jeweils neue Sprache zur Formalisierung der Benutzerschnittstelle erlernt werden musste und das gewünschte Ergebnis einer automatisiert erzeugten Benutzerschnittstelle nur unter hohem Aufwand erzeugbar war.

Durch das Aufkommen standardisierter Modellierungssprachen wie beispielsweise der UML kann dieses Hindernis überwunden werden. Hierbei kommen unterschiedliche Modelle zur Spezifikation der verschiedenen Aspekte einer Benutzerschnittstelle zum Einsatz, von welchen das Modell, das als Ausgangspunkt der Modellierung dient, von zentraler Bedeutung ist. Wie in [HA+97] festgestellt wird, bestimmt die Sicht, die auf das zu realisierende System eingenommen wird, diesen Ausgangspunkt maßgeblich. In frühen Ansätzen der Modellierung von Benutzerschnittstellen stand häufig die Visualisierung eines **Datenmodells** im Vordergrund, auf welches der Benutzer über eine Benutzerschnittstelle Zugriff erhalten sollte [BF+92, GB+01]. Der Begriff Datenmodell wurde insbesondere im Bereich der objektorientierten Softwareentwicklung weiter verfeinert hin zu den Objekten, die in einer Anwendung verarbeitet werden. Da diese Objekte den Kontext der Anwendung näher spezifizieren, hat sich hier der Begriff des **Domänenmodells** etabliert. In [JB+99] wird die in [NT+99] als allgemein akzeptiert bezeichnete Definition eines Domänenmodells gegeben:

> *"A domain model captures the most important types of objects in the context of the system. The domain objects represent "things" that exist or events that transpire in environment in which the system works."*

Das Domänenmodell einer Anwendung reflektiert nach dieser Definition, aus welchen Entitäten die Anwendungsdomäne besteht und in welchen statischen Beziehungen sie zueinander stehen. Eine Domäne bildet damit den Übergang von einer abstrakt-konzeptuellen Sicht einer Anwendung auf deren funktional-fachliche Ausprägung [Nu07]. Im Kontext der Benutzerinteraktion wird der Begriff des (benutzerzentrischen) Domänenmodells durch [Va05] adaptiert und definiert als:

> *"a description of the classes of objects manipulated by a user while interacting with a system. Typically, it could be a UML class diagram, an entity-relationship-attribute model, or an object-oriented model."*

Einige Ansätze wie etwa [Ko01] sehen daher das Domänenmodell einer Anwendung als das **konzeptionelle Modell** der Benutzerschnittstelle an. Die rein datenzentrische Sicht auf eine Anwendung, die sich in einigen Bereichen der Softwaretechnik etabliert hat, ist für die Entwicklung von interaktiven Anwendungen nicht ausreichend. Aus diesem Grund forcieren nutzerorientierte Vorgehensmodelle wie das *User-Centered Design* oder das *Human-Centered Software Engineering* [SG+05, CN07] eine starke Fokussierung der potenziellen Anwender. Dementsprechend fokussiert eine Vielzahl der heute existierenden Ansätze aus dem Bereich des *User-Centered Design* einen zusätzlichen, als **Aufgabenmodell** (engl. *Task Model*) bezeichneten Modelltyp [NT+99, LC+03, Va05, SP03, Co06]. In diesem Modell stehen die Aufgaben des Menschen, bei welchen er durch eine Anwendung unterstützt werden soll, im Mittelpunkt. Das Aufgabenmodell wird nach [NT+99] in Anlehnung an [HA+97] wie folgt definiert:

> *"Task models describe the activities in which users engage to achieve some goal […] A task model details the users' goals and the strategies adopted to achieve those goals, in terms of the actions that users perform, the objects involved in those actions and the sequencing of activities."*

Ein weiterer Modelltyp, in dem der Benutzer eine zentrale Rolle einnimmt, ist das bereits aus Abschnitt 2.2 bekannte **Prozessmodell** einer Anwendung [NT+99]. Bedingt durch den voranschreitenden Paradigmenwechsel von einer rein funktionalen hin zu einer prozessorientierten Sicht auf die IT stellen neuere Forschungsansätze wie beispielsweise [SK+05, SS+07a, BC+06] die Eignung von Prozessmodellen zur Aufgabenmodellierung fest und nutzen diese als Ausgangspunkt der Modellierung von Benutzerschnittstellen (vgl. dazu auch Abschnitt 3.2). Ein Modell eines Geschäftsprozesses dient damit nicht nur der Erfassung von Geschäftsanforderungen, sondern ist zusätzlich als Ausgangsmodell der Entwicklung der Benutzerschnittstellen zu sehen. Die Autoren von [SK+05] sehen in diesem Fokuswechsel, weg vom in der Regel eher datenzentrischen *User-Centered Design* hin zur Geschäftsprozessmodellierung, den Scheideweg der Erstellung schneller Prototypen im Bereich der Benutzerinteraktion, da die Geschäftsprozessmodellierung zwar sehr viele Gemeinsamkeiten mit dem *User-Centered Design* aufweist, zusätzlich jedoch den heute geforderten expliziten Bezug zu den Geschäftsanforderungen eines Unternehmens herstellt.

2.3.2 Modellierung grafischer Benutzerschnittstellen

Es herrscht in der Literatur allgemeiner Konsens darüber, dass eine Benutzerschnittstelle durch unterschiedlichste statische und dynamische Aspekte beschrieben werden muss [PP02, Br06, Ko01,

Nu07, LC+03, WF+05]. Die hierzu benötigten Modelle lassen sich nach dem Prinzip der getrennten Zuständigkeiten (engl. *Separation of Concerns*) [Di82] ebenfalls in unterschiedliche Aspekte unterteilen. So unterscheiden viele Ansätze wie auszugsweise [KK08, MF+07] zunächst die Aspekte Inhalt, Navigation und Präsentation. Ein **Inhaltsmodell** (engl. *Content Model*) umfasst die Menge aller Daten, die durch den Benutzer über eine Benutzerschnittstelle manipuliert werden können. Daher steht das Inhaltsmodell im direkten Bezug zum in Abschnitt 2.3.1 eingeführten Domänenmodell. Wird das Domänenmodell im Kontext der Benutzerschnittstelle nach der Definition von [Va05] als Menge aller Klassen und Objekte, die in einer Interaktion durch den Menschen manipuliert werden können, aufgefasst, so kann der Begriff Inhaltsmodell synonym dazu verwendet werden [SP03]. Aufbauend auf einem Inhalts- resp. Domänenmodell spezifiziert ein **Präsentationsmodell** (engl. *Presentation Model*) die Struktur des äußeren Erscheinungsbilds sowie das Aussehen (engl. *Layout*) der Benutzerschnittstelle. Durch die heute vorhandene Vielzahl von Endgeräten ist eine klare Trennung von Struktur und Layout eine Grundvoraussetzung, um eine Benutzerschnittstelle auf unterschiedlichen Endgeräten in geeigneter Form darstellen zu können. Dementsprechend findet sich in der Literatur als Verfeinerung eines Präsentationsmodells der Begriff des **Strukturmodells** [PP02], das lediglich die statische Struktur einer Benutzerschnittstelle ohne Details über deren Layout wiedergibt. Im Strukturmodell kann daher definiert werden, dass die Elemente der Benutzerschnittstelle in einem gewissen Bezug zueinander stehen und somit einer Struktur folgen, nicht aber, in welcher Farbe oder Größe ein Element letztendlich dargestellt wird.

Der dynamische Aspekt einer Benutzerschnittstelle umfasst die Möglichkeiten des Benutzers, innerhalb eines festgelegten Zustandsraumes zu navigieren. Ein entsprechendes **Navigationsmodell** formalisiert folglich diesen Zustandsraum und hält fest, von welchem Zustand aus der Benutzer durch welche Aktion zu welchem Folgezustand gelangen kann [Ko01]. Zusätzlich findet sich beispielsweise in [PP02] der Begriff des **Verhaltensmodells**, welches konkrete Details des Ablaufes der Interaktion eines Benutzers mit einer Benutzerschnittstelle spezifiziert.

Bedingt durch die Vielzahl der vorhandenen Endgeräte und Plattformen sehen mehrere Ansätze eine weitere Unterteilung der verwendeten Modelle in abstrakte und konkrete Modelle vor [KK+07a, WF+05, Co06, LC+03, PP03]. Ein Modell einer **abstrakten Benutzerschnittstelle** (engl. *Abstract User Interface,* AUI) hat keinen Bezug zu konkreten Technologien oder Plattformen und soll auf diese Weise die Portabilität der Modelle sicherstellen. Dementsprechend kann ein Modell einer abstrakten Benutzerschnittstelle zu mehreren Modellen **konkreter Benutzerschnittstellen** (engl. *Concrete User Interface,* CUI) ausgeprägt werden. Diese differenzierte Betrachtung kann analog zu der durch die modellgetriebene Architektur forcierten Unterscheidung in plattformunabhängige und plattformspezifische Modelle verstanden werden. [Va05] und [MA+06] führen diese Herangehensweise über die Modellebenen hinaus bis auf die Ebene des Quellcodes weiter und definieren eine **finale Benutzerschnittstelle** (engl. *Final User Interface*, FUI) schließlich als den lauffähigen Quellcode einer Benutzerschnittstelle. Auch hier kann erneut der Bezug zu der plattformspezifischen Implementierung aus dem Kontext der modellgetriebenen Architektur hergestellt werden. Die eingeführten Modelltypen dienen als Grundlage zur Einordnung der in Kapitel 4 konzipierten Modelle, die im Rahmen der modellgetriebenen Softwareentwicklung in dieser Arbeit verwendet werden.

Um eine Benutzerschnittstelle letztendlich realisieren zu können, bedarf es einerseits einer Softwarearchitektur, in welcher die Benutzerschnittstelle verankert wird, und andererseits einer konkreten Programmiersprache, in welcher die Benutzerschnittstelle implementiert wird. Im Bereich der dienstorientierten Architekturen hat die Benutzerschnittstelle lange Zeit nur sehr geringe Beachtung gefunden. So hält bereits [Ar04] fest, dass die Präsentationsschicht bisher zwar kaum im Kontext einer

dienstorientierten Architektur diskutiert wird, sie aber sehr wohl zunehmend an Bedeutung gewinnt. Dennoch ist in der Literatur wie etwa [Li07, BB+05, WM06, AL+07, HH+06] bis heute kein einheitliches Verständnis über den Aufbau und die interne Architektur der Präsentationsschicht sowie deren Integration in eine dienstorientierte Architektur vorhanden. Dementsprechend beleuchtet der folgende Abschnitt die dienstorientierte Architektur als die dieser Arbeit zugrunde liegende Softwarearchitektur und nimmt dann Bezug zu bestehenden Konzepten für Benutzerschnittstellen in dienstorientierten Architekturen.

2.4 Dienste und dienstorientierte Architekturen

Die heute geforderte Flexibilität der IT in der Unterstützung von Geschäftsprozessen muss neben dem in Abschnitt 2.1 und 2.2 betrachteten Entwicklungsvorgehen auch durch die zum Einsatz kommende Softwarearchitektur erbracht werden. In diesem Kontext hat der Fachbegriff des Dienstes und daraus abgeleitet das Konzept der dienstorientierten Architektur eine zentrale Bedeutung für diese Arbeit. Daher werden in den folgenden Abschnitten zunächst der Dienstbegriff beleuchtet und anschließend die notwendigen Basistechnologien zum Aufbau einer dienstorientierten Architektur eingeführt.

2.4.1 Der Dienstbegriff

Eine domänenunabhängige Definition des Dienstbegriffes gibt das Deutsche Institut für Normung (DIN) in DIN EN ISO 9000:2005. Nach dieser Definition ist ein **Dienst** bzw. synonym eine Dienstleistung

> *„das Ergebnis mindestens einer Tätigkeit, die notwendigerweise an der Schnittstelle zwischen dem Lieferanten und dem Kunden ausgeführt wird und üblicherweise immateriell ist.“*

Das Grundprinzip des Dienstes, der für einen Dienstnehmer (Kunde) von einem Dienstgeber (Lieferant) an der Schnittstelle zwischen diesen beiden Rollen erbracht wird, kann auch auf die IT ausgeprägt werden. Insbesondere im Kontext der verteilten Softwaresysteme hat der Dienstbegriff eine zentrale Bedeutung. So spezifiziert das OSI-Referenzmodell den Begriff „Dienst“ (engl. *Service*) durch die Unterteilung eines IT-Systems in fachlich logische Schichten. Jede dieser Schichten bietet ihre Funktionalität der logisch darüber liegenden Schicht als Dienst über einen Dienstzugangspunkt (engl. *Service Access Point*, SAP) an [AL+02, Bl91]. Hierdurch wird eine gesteigerte Flexibilität der Gesamtarchitektur erreicht, indem konkrete Implementierungsdetails von Schicht zu Schicht verschattet werden. Bleiben die Dienstzugangspunkte konstant, so ist die interne Implementierung der jeweiligen Schicht nicht von Bedeutung; sie kann ausgetauscht werden.

In [Ro03] wird der auf die IT bezogene Dienstbegriff um eine Dimension erweitert, indem die Funktionalität, die durch einen Dienst erbracht wird, feiner spezifiziert wird:

> *“The functionality of a service consists of two parts: (i) the functionality of the service itself, and (ii) the functionality of sub-services that are involved in the service provisioning with respect to the service hierarchy.”*

Die durch Dienste erbrachte Funktionalität kann demnach durch die Dienstkomposition (d. i. das Verschalten einzelner Dienste) als höherwertige Funktionalität in Form von komponierten Diensten bereitgestellt werden [AC+04, Ar04]. Für den Dienstnehmer ist der interne Aufbau eines Dienstes, ob komponiert oder atomar, nicht von Belang.

2.4.2 Dienstorientierte Architekturen

In der Informatik beschreibt eine Softwarearchitektur die interne Struktur eines Softwaresystems. Diese Struktur setzt sich aus verschiedenen Systemkomponenten und deren Bezug untereinander zusammen. Eine **Systemkomponente** ist nach [Ba00]

> *„ein abgeschlossener, binärer Software-Baustein, der eine anwendungsorientierte, semantisch zusammengehörende Funktionalität besitzt, die nach außen über Schnittstellen zur Verfügung gestellt wird."*

Auf dem Begriff der (System-)Komponente aufbauend definiert das Handbuch der Softwarearchitektur [RH06] eine **Softwarearchitektur** als

> *„die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zueinander und zur Umgebung, sowie die Prinzipien, die den Entwurf und die Evolution des Systems beeinflussen."*

Werden die Systemkomponenten in Form von Diensten in einer Softwarearchitektur realisiert, so wird diese als dienstorientierte Architektur (engl. *Service-Oriented Architecture*, SOA) bezeichnet. In [WS06] wird eine Softwarearchitektur als dienstorientiert angesehen, wenn:

> *„die Funktionalitäten in Form von Diensten gekapselt sind, die über standardisierte, publizierte Schnittstellen verfügen. Weiterhin müssen die so gekapselten Funktionalitäten lose gekoppelt und atomar sein."*

In dieser Definition spiegeln sich die unter anderem in [DJ+05] und [Jo08] aufgeführten Merkmale einer dienstorientierten Architektur wider. Die Forderung nach einer gekapselten Funktionalität ist mit der oben angeführten Verschattung der internen Implementierung einer Systemkomponente gleichzusetzen. Funktionalität wird über eine Schnittstelle bereitgestellt, welche die interne Komplexität dieser Komponente verbirgt. Wird die Spezifikation eines Dienstes auf diese Schnittstelle reduziert, so kann die ebenfalls geforderte lose Kopplung der Dienste erreicht werden [HH+06]. Als Kopplung zweier Komponenten wird dabei die Menge der zwischen diesen Komponenten getroffenen Annahmen und Vereinbarungen verstanden [ST07]. Je geringer der Grad der Abhängigkeit durch Annahmen oder Vereinbarungen zwischen zwei Komponenten ist, desto geringer sind die Auswirkungen von Änderungen [Jo08]. Die geringe Abhängigkeit der einzelnen Dienste untereinander wird daher als einer der zentralen Erfolgsfaktoren für eine dienstorientierte Architektur gesehen [BB+05]. Eine lose Kopplung atomarer, in sich abgeschlossener Dienste führt somit zu einer erhöhten Wiederverwendbarkeit dieser Dienste, da diese ohne größeren Aufwand ersetzt und in beliebiger Form zu neuen Diensten verschaltet werden können. Das Ziel, Funktionalität wiederverwendbar als Dienste über standardisierte Schnittstellen bereitzustellen, ist gleichzeitig die Grundlage der integrativen Eigenschaften einer dienstorientierten Architektur. Dementsprechend definiert [DJ+05] eine **dienstorientierte Architektur** wie folgt:

> *„Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht."*

Der in dieser Definition enthaltene Aspekt der Wiederverwendung von Diensten bekommt unter der Betrachtung heutiger IT-Systemlandschaften ein besonderes Gewicht. Über viele Jahre hinweg sind in

den Unternehmen heterogene IT-Systemlandschaften entstanden, deren einzelne IT-Systeme speziell zur Unterstützung der Bewältigung einer konkreten Aufgabe angeschafft wurden. Durch die Schnelllebigkeit heutiger Märkte ändern sich diese Aufgaben jedoch rasch. Die Anschaffung zusätzlicher IT-Systeme zur Bewältigung neuer Aufgaben führt somit schnell zu einer heterogenen IT-Landschaft, die neben hohen Anschaffungskosten nur schwer zu überblicken und zu warten ist. Hieraus entsteht die Anforderung, bestehende IT-Systeme zu konsolidieren und deren Einsatzmöglichkeiten zu flexibilisieren [BB+05]. Ein Umdenken von einer rein funktionalen Sicht auf die IT hin zu deren Ausrichtung an den Geschäftsprozessen des Unternehmens ist notwendig (vgl. Abschnitt 2.2). Durch eine flexible Bereitstellung der Funktionalität bestehender IT-Systeme in Form von Diensten ermöglicht der Ansatz der dienstorientierten Architektur eine enge Verzahnung von Geschäftsprozessen und IT [AL+07]. Die Ausrichtung der IT am Geschäftsprozess stellt den wesentlichen Unterschied zwischen bisherigen komponenten- oder objektorientierten Ansätzen und der dienstorientierten Architektur dar [OASIS-SOA-RMCD]. Der damit einhergehende Paradigmenwechsel von einer reinen funktionalen Sicht der IT hin zu einer abstrakteren, geschäftsgetriebenen Sicht schlägt sich bereits, wie in Abschnitt 2.2 erörtert, auf neuere Vorgehensmodelle zur Softwareentwicklung nieder. Dienstorientierte Architekturen ermöglichen die notwendige Flexibilität, um diesen Paradigmenwechsel auch auf der Ebene der Softwarearchitektur zu realisieren. Durch die Bereitstellung von (Fach-)Funktionalität als Dienst können bestehende Altsysteme (engl. *Legacy Systems*), sofern eine Migration dieser Altsysteme sinnvoll und wirtschaftlich ist, in eine dienstorientierte Architektur integriert werden.

Neben den in Form von fachfunktionalen Diensten bereitgestellten Altanwendungen besteht eine dienstorientierte Architektur aus weiteren Architekturkomponenten. Die genaue Art dieser Architekturkomponenten ist durch aktuelle Forschungsansätze sowie durch die etablierte Fachliteratur bis heute nicht abschließend diskutiert. Kern einer dienstorientierten Architektur bilden die **fachfunktionalen Dienste**, deren Funktionalität durch eine interne Implementierung an einer fest definierten Schnittstelle bereitgestellt wird. Die Implementierung kann dabei je nach Ausprägung der bereitgestellten Funktionalität weiter in Geschäftslogik und Datenhaltung verfeinert werden. Die Außensicht eines Dienstes über dessen Schnittstelle wird durch eine **Dienstleistungsvereinbarung** ergänzt. Wie durch den Autor in [AL+05] publiziert, trifft diese je nach konkretem Einsatzszenario des Dienstes Aussagen über die Qualität der Erbringung des Dienstes zwischen einem Dienstnehmer und einem Dienstgeber (vgl. dazu [Ro03], [Ma01]). Um die Menge der vorhandenen Dienste der Dienstgeber erfassen und potenziellen Dienstnehmern anbieten zu können, wird ein **Dienstverzeichnis** benötigt [Kr07].

Die Kommunikationsbeziehung zwischen den einzelnen Komponenten einer dienstorientierten Architektur wird in manchen Quellen durch einen ***Enterprise Service Bus*** (ESB) [Ch04, AK+05] realisiert. In der Literatur reicht die Mächtigkeit dieser Komponente dabei von einer einfachen Kommunikationsvermittlung zwischen Diensten bis hin zu einer hochkomplexen Managementanwendung zur dynamischen Verschaltung von Diensten zur Laufzeit [KA+04]. Ferner benötigt eine dienstorientierte Architektur eine Benutzerschnittstelle, über die (menschliche) Benutzer auf Dienste zugreifen und diese in Anspruch nehmen können. Da die Benutzerschnittstelle eine zentrale Rolle in dieser Arbeit einnimmt, wird diese Architekturkomponente im folgenden Abschnitt detailliert betrachtet.

Durch das Dienstparadigma abstrahiert eine dienstorientierte Architektur zunächst von konkreten Implementierungsdetails. Somit können dienstorientierte Architekturen unabhängig von konkreten Technologien als Entwurfsparadigma in der Entwurfsphase der Softwareentwicklung betrachtet werden [WM06]. Die sich anschließende Implementierung einer dienstorientierten Architektur bedarf jedoch sehr wohl konkreter Technologien, um beispielsweise die Kommunikation zwischen den

einzelnen Diensten zu ermöglichen oder die Schnittstellen der Dienste zu spezifizieren. In jüngster Vergangenheit haben sich die durch das *Word Wide Web Consortium* (W3C) geprägten Webservices sowie die den Webservices zugrunde liegenden Technologien und Standards, wie beispielsweise die *Web Service Description Language* (WSDL) [W3C-WSDL1.1], als Beschreibungssprache für die Schnittstellen der Dienste oder SOAP (vormals *Simple Object Access Protocol*) [W3C-SOAP1.2] als Format zum Nachrichtenaustausch in unterschiedlichsten Bereichen etabliert. Durch die allgemeine Akzeptanz der Webservices als Set konkreter Technologien zur Implementierung einer dienstorientierten Architektur ist eine Vielzahl an Literaturquellen über Webservices wie etwa [AC+04], [WC+05] oder [DJ+05] vorhanden, auf die an dieser Stelle für eine detaillierte Betrachtung der Webservices und der zugeordneten Basistechnologien verwiesen sei.

Neben der Frage nach den Architekturkomponenten muss auch die Frage nach der logischen Struktur einer dienstorientierten Architektur beantwortet werden. Auch hier hat sich in der Literatur bisher keine einheitliche Sichtweise etablieren können. Getrieben durch die unterschiedlichsten Sichtweisen auf eine dienstorientierte Architektur, bestehen vielmehr verschiedenste Ansichten über die logische Struktur einer dienstorientierten Architektur [Li07]. Einen abstrakten, aber richtungweisenden Hinweis auf die logische Struktur einer dienstorientierten Architektur ist in [Ar04] gegeben. Auf diesem aufbauend stellt der in [AL+07] zu findende *Service Solution Stack* (S3), der in Abbildung 15 dargestellt ist, eine Referenzarchitektur der dienstorientierten Architekturen dar, die auf Basis verschiedener Projekte konzipiert wurde.

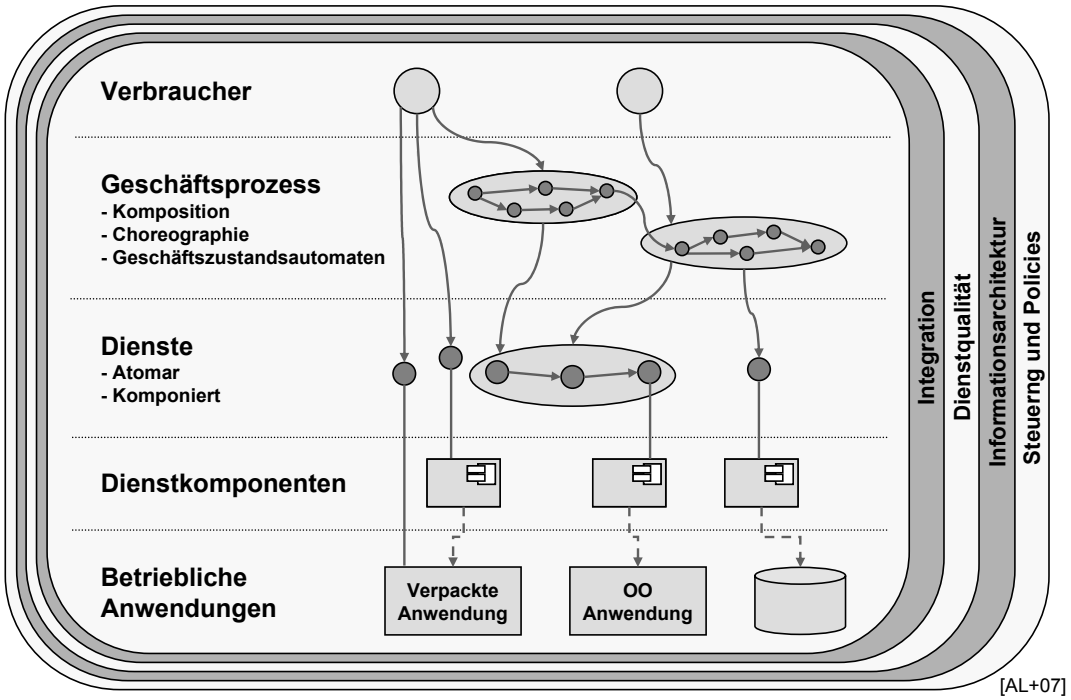

Abbildung 15: Logische Struktur des Service Solution Stack

Sehr ähnlich wird die oberste Ebene des *Service Solution Stacks* als die logische Struktur einer dienstorientierten Architektur in der Forschungsgruppe Cooperation & Management (C&M, Prof. Abeck) gesehen, in der diese Arbeit entstanden ist. Abbildung 16 zeigt das durch den Autor in verschiedenen Publikationen wie [EL+06] und [LH+08] präsentierte Gesamtverständnis über die logische Struktur einer dienstorientierten Architektur, das ausgehend von der in [Re05] publizierten Referenzarchitektur von C&M kontinuierlich weiterentwickelt wurde.

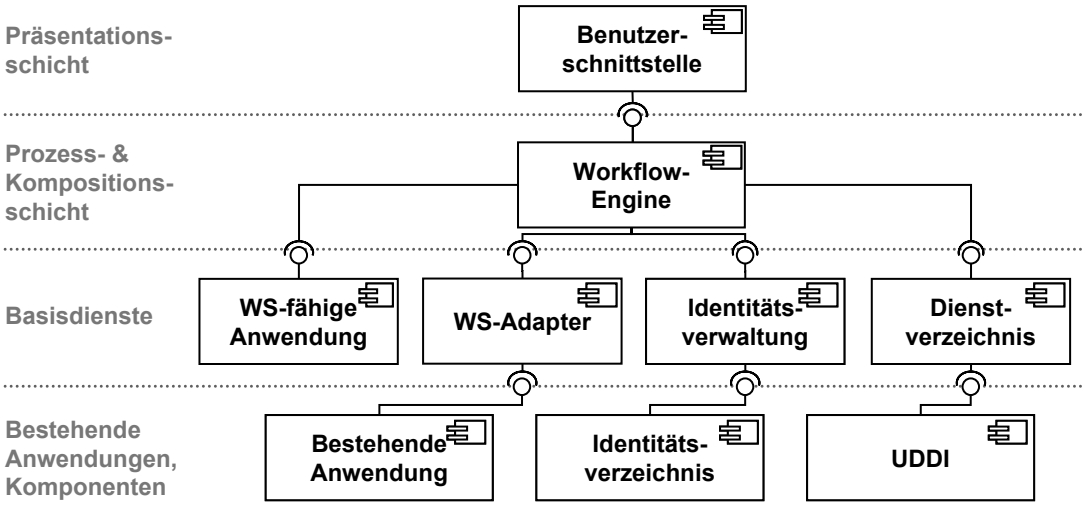

Abbildung 16: Logische Struktur der dienstorientierten Architektur von C&M

Wie in Prämisse P4 festgehalten, liegt der Fokus dieser Arbeit auf einer Webservice-basierten dienstorientierten Architektur. Folglich werden Dienste in Form von Webservices realisiert und beide Begriffe im Folgenden synonym verwendet. Auf der untersten Schicht der C&M-Referenzarchitektur sind bestehende Anwendungen angesiedelt, die zur Integration in die dienstorientierte Architektur entweder durch entsprechende Webservice-Adapter erweitert werden müssen oder im Idealfall bereits per se Webservice-fähig sind. Die bestehenden Anwendungen stellen die als atomar oder auch elementar bezeichneten Webservices bereit, welche die Basisdienste einer dienstorientierten Architektur darstellen. Zu diesen gehören unter anderem ein Dienstverzeichnis, das in Abbildung 16 in Form eines *Universal Description, Discovery and Integration* (UDDI) [OASIS-UDDI2] Verzeichnisdienstes realisiert ist und ein Identitätsmanager, der den Zugriff auf die zugrunde liegende Organisationsstruktur und damit eine Zugriffskontrolle in dienstorientierten Architekturen ermöglicht. Eine ausführliche Betrachtung der beiden letztgenannten Komponenten findet sich in [Kr07] und [Em08].

Die vorhandenen Basisdienste können durch Webservice-Kompositionen rekursiv zu komponierten Diensten verschaltet und zur Unterstützung von Workflows verwendet werden. Eine Workflow-Ausführungsumgebung (vgl. Abschnitt 2.2.2.1) instanziiert die hinterlegten Workflow-Definitionen, ruft benötigte Webservices (atomar oder komponiert) auf und überwacht schließlich die Ausführung der Workflows. Daher wird diese Komponente in der Literatur häufig auch als Workflow-*Engine* bezeichnet. Eine entsprechende Benutzerschnittstelle ermöglicht die Integration des menschlichen Benutzers und gestattet dessen Interaktion im Kontext unterschiedlicher Workflows. Aus diesem Grund beschäftigt sich der folgende Abschnitt im Detail mit der Benutzerschnittstelle als zentrale Komponente einer dienstorientierten Architektur, die den Zugriff des Menschen ermöglicht, und betrachtet vorhandene Standards und Rahmenwerke für deren Realisierung.

2.4.3 Benutzerschnittstellen für dienstorientierte Architekturen

In einer dienstorientierten Architektur wird Fachfunktionalität in Form von Diensten an einheitlichen Schnittstellen angeboten. Auf diese Schnittstellen kann der Mensch über unterschiedliche Benutzerschnittstellen zugreifen. [Li07] nimmt eine Einteilung der Benutzerschnittstellen für dienstorientierte Architekturen in drei Klassen vor. Der Zugriff auf Dienste durch sogenannte ***Client*-Anwendungen** folgt den Prinzipien traditioneller Anwendungen, die nach herkömmlichen Architekturmodellen in logische Schichten wie beispielsweise Daten, Geschäftslogik und Präsentation unterteilt werden und damit vollständige Anwendungen darstellen (vgl. [AL+02]). In besonders Dokument-getriebenen

Unternehmen können sogenannte **Office-Anwendungen** wie etwa Microsoft Office oder GoogleDocs den Zugriff auf eine dienstorientierte Architektur realisieren. Dokumente und Dokumentflüsse stellen in diesen dienstorientierten Architekturen die zentralen Artefakte dar. Eine dritte Realisierungsform der Benutzerschnittstelle für eine dienstorientierte Architektur bieten die **Portale**. Mit einem Portal kann das Paradigma der Dienstorientierung auch auf der Präsentationsebene einer dienstorientierten Architektur konsequent fortgeführt werden. Deshalb werden Portale in dieser Arbeit als Realisierungsform der Benutzerschnittstelle eingesetzt und im folgenden Abschnitt detailliert eingeführt.

2.4.3.1 Portale

Der Begriff „Portal“, der ursprünglich aus dem Lateinischen *porta* (dt. Pforte oder großes Tor) [Du06] stammt, wird seit Ende des vergangenen Jahrtausends in der Informatik diskutiert [Gl00]. Unter dem Begriff des Portals wird allgemein ein Konzept für den personalisierten Zugriff auf Informationen und Applikationen verstanden [Pu04]. In [Sm04] wird ein **Portal** definiert als:

> *„an infrastructure providing secure, customizable, personalizable, integrated access to dynamic content from a variety of sources, in a variety of source formats, wherever it is needed.“*

In dieser Definition finden sich die zentralen Aspekte eines Portals wieder, über die in der Literatur aus unterschiedlichen Fachdomänen der Informatik heute Konsens besteht [Ba08, Gu06, Zö06, AA+03, BS05]:

- **Integration**. Der zentralste Aspekt eines Portals ist dessen Integrationsfähigkeit [Gl00]. Diese umfasst neben der Aggregation von Inhalten aus verschiedenen verteilten Quellen auch die Bereitstellung von Anwendungen für die Benutzer eines Portals [Zö06]. Anstatt sich mit vielen unterschiedlichen Benutzerschnittstellen beschäftigen zu müssen, ist in einem Portal ein einheitlicher Zugriff auf die benötigten Anwendungen realisierbar [Be08].

- **Personalisierung.** Portale gestatten es den Anwendern, die angebotenen Anwendungen und Inhalte bei Bedarf selbstständig an die eigenen Präferenzen anzupassen [Rü01]. Eine individuelle und damit optimale Unterstützung der Anwender wird so ermöglicht. Ferner sind Portale in der Lage, die zur Verfügung gestellten Inhalte und Anwendungen in Abhängigkeit der durch den Anwender genutzten Endgeräte in unterschiedlicher Form auszugeben. Portale bieten somit für jeden Benutzer eine auf ihn individuell zugeschnittene optimale Darstellung ihrer Inhalte [Zö06].

- **Einfacher Zugriff.** Die Integrations- und Personalisierungseigenschaften eines Portals ermöglichen einen einfachen Zugriff auf unterschiedliche Anwendungen und Daten. Die Authentifizierung und anschließende Autorisierung eines Anwenders am Portal erfolgt nur einmal. Dieser als *Single Sign-On* [GK+03] bezeichnete Mechanismus (vgl. dazu auch [Em08]) gestattet dem Anwender, durch eine einmalige Anwendung Zugriff zu allen ihm zugedachten Anwendungen und Daten zu erhalten. Eine ohne Portal notwendige mehrfache Anmeldung an unterschiedlichen Anwendungen mit gegebenenfalls unterschiedlichen Passwörtern entfällt.

Da ein Portal den Zugriff auf unterschiedliche Anwendungen ermöglicht, können Portale auch zur Abarbeitung eines Geschäftsprozesses herangezogen werden. Dementsprechend muss ein Portal für den Menschen die jeweils passende Benutzerschnittstelle für die unterschiedlichen, durch ihn zu

erledigenden Aufgaben bereitstellen. Die dazu notwendigen präsentationsorientierten Softwarekomponenten werden als *Portlets* bezeichnet und im folgenden Abschnitt betrachtet.

2.4.3.2 Portlets

Ein Portal stellt einem Benutzer über eine Portalseite (engl. *Portal Page*) einen einheitlichen Zugriff auf unterschiedliche Anwendungen zur Verfügung. Eine Portalseite wiederum setzt sich aus den als *Portlets* bezeichneten einzelnen Benutzerschnittstellen der Anwendungen zusammen. Öffnet ein Benutzer eine Portalseite (beispielsweise durch seinen Webbrowser), so greift das Portal auf die hinterlegte Definition der Seite zu und integriert alle zum Seitenaufbau benötigten *Portlets*, indem es von einem sogenannten *Portlet Container* die entsprechenden *Portlets* abruft. Es sei an dieser Stelle angemerkt, dass der Begriff *Portlet* aufgrund seiner offensichtlichen Nähe zu anderen Konzepten wie *Applet* oder *Servlet* häufig mit einer konkreten Java-Implementierung verknüpft wird. Allerdings entstand der Begriff *Portlet* bereits vor der ersten Java-*Portlet*-Spezifikation und wird daher in der Fachliteratur technologieneutral verwendet [DR04, TB+07]. Dennoch wird das *Portlet*-Konzept maßgeblich im Java-Kontext vorangetrieben, weshalb sich in der Java-*Portlet*-Spezifikation 2.0 [JCP-JSR286] eine aussagekräftige Definition des Begriffs ***Portlet*** findet:

> *"A portlet is an application that provides a specific piece of content (information or service) to be included as part of a portal page. It is managed by a portlet container, that processes requests and generates dynamic content. Portlets are used by portals as pluggable user interface components that provide a presentation layer to information systems."*

Ein *Portlet* kann in unterschiedlichen Sprachen wie beispielsweise Java oder .NET realisiert werden. Somit ist ein *Portlet* zunächst herstellerspezifisch und an ein konkretes Portal gebunden, das die entsprechende Implementierung unterstützt [AC+05].

Diese Einschränkung kann erneut durch das Paradigma der Dienstorientierung auch auf der Präsentationsebene überwunden werden. Wird ein *Portlet* als Webservice angeboten, so kann die Benutzerschnittstelle in Form einer Portalseite in der logischen Fortsetzung der Dienstorientierung als Verschaltung von (präsentationsorientierten) Webservices angeboten werden. Dieses Ziel verfolgt der OASIS-Standard *Web Service for Remote Portlets* (WSRP) [OASIS-WSRP2], der im folgenden Abschnitt eingeführt wird.

2.4.3.3 Web Services for Remote Portlets

Portalseiten werden aus einzelnen *Portlets* aggregiert und können somit an individuelle Bedürfnisse angepasst werden. Da zur Implementierung von *Portlets* Spezifikationen wie beispielsweise die Java-*Portlet*-Spezifikation [JCP-JSR168] zum Einsatz kommen, ist die Wiederverwendung von *Portlets* in unterschiedlichen Portalseiten zunächst auf diejenigen Portale beschränkt, die den entsprechenden Standard implementieren. Diese Einschränkung adressiert der OASIS-Standard *Web Services for Remote Portlets* (kurz: WSRP) [OASIS-WSRP2], indem er Konzepte für eine plattformunabhängige Implementierung von *Portlets* in Form von Webservices spezifiziert. Somit beschreibt WSRP die Anwendung des Dienstparadigmas auf der Präsentationsebene einer Softwarearchitektur, indem er unter anderem eine lose Kopplung und die Wiederverwendung von *Portlets* adressiert [AC+05].

Hierzu erweitert WSRP die herkömmliche Architektur eines Portals um zwei zusätzliche Komponenten und entkoppelt durch diese das Portal von dessen *Portlet Container* [Ca05]:

- **WSRP-*Consumer*:** In einer herkömmlichen Portalarchitektur baut das Portal eine Portalseite auf, indem es Anfragen an den eigenen *Portlet Container* schickt und die auf diese Weise erhaltenen Seitenfragmente zu einer kompletten Portalseite zusammenstellt. In der durch WSRP erweiterten Portalarchitektur (vgl. Abbildung 17) sendet das Portal seine Anfragen nun nicht mehr direkt an den *Portlet Container*, sondern an den sogenannten WSRP-Konsumenten (engl. WSRP-*Consumer*). Dieser ist als Webservice-*Client* spezifiziert, der Webservices von beliebigen WSRP-Produzenten (engl. *WSRP-Producer*) aufrufen kann und damit eine lose Kopplung zwischen Portal und unterschiedlichen WSRP-Produzenten ermöglicht.

- **WSRP-*Producer*:** Der WSRP-Produzent (engl. WSRP-*Producer*) ist als Webservice realisiert, der unterschiedliche Operationen auf *Portlets* für WSRP-Konsumenten zur Verfügung stellt. Je nach Implementierung kann ein WSRP-Produzent nur ein *Portlet* oder mehrere *Portlets* in einem eigenen *Portlet Container* als Laufzeitumgebung zur Verfügung stellen. Ein WSRP-Produzent kann durch einen WSRP-Konsumenten über WSDL-Schnittstellen und entsprechende Endpunkte angesprochen werden.

Zusätzlich definiert WSRP ein **WSRP-*Portlet*** als ein als Webservice realisiertes *Portlet*, das in einer durch den WSRP-Produzenten bereitgestellten Laufzeitumgebung existiert. Demnach kann das WSRP-*Portlet* nicht direkt als Webservice aufgerufen werden. Der Zugriff muss immer über den WSRP-Produzenten erfolgen, dem das WSRP-*Portlet* zugeordnet ist. Abbildung 17 zeigt die nach den Konzepten des OASIS-Standards erweiterte Präsentationsschicht einer dienstorientierten Architektur.

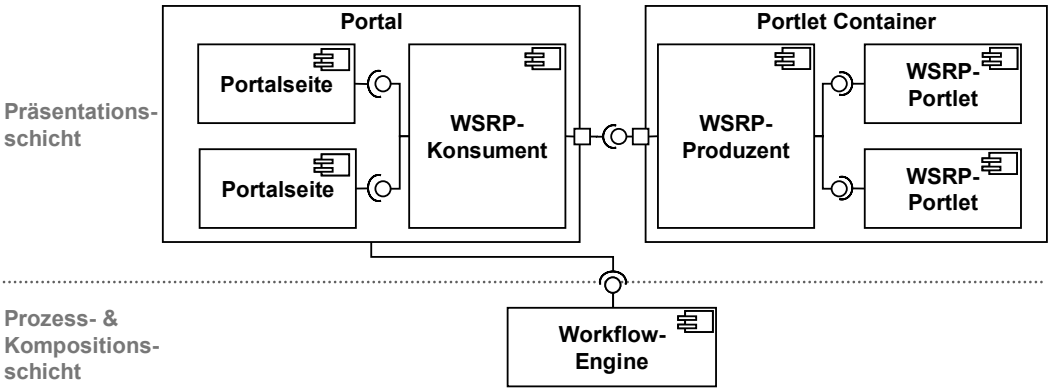

Abbildung 17: Erweiterte Portalarchitektur von WSRP

Die in Abbildung 17 schematisch angedeutete Schnittstelle zwischen WSRP-Konsument und WSRP-Produzent muss in jeder konkreten Implementierung eines WSRP-Produzenten durch zwei erforderliche Schnittstellen ausgeprägt werden [OASIS-WSRP2, AA+05]:

- ***Service Description*:** Da ein WSRP-Produzent zunächst ein Webservice ist, kann dieser über Mechanismen wie beispielsweise *Universal Description, Discovery and Integration* (UDDI) [OASIS-UDDI2] gefunden werden und seine Fähigkeiten publizieren. Da nicht alle Softwarearchitekturen ein UDDI vorsehen und implementieren, schreibt der Standard die Schnittstelle *Service Description* (dt. Dienstbeschreibung) vor. Über diese Schnittstelle kann ein WSRP-Konsument einerseits Metadaten des WSRP-Produzenten selbst, aber andererseits auch die von ihm zur Verfügung gestellten WSRP-*Portlet*s abfragen. Die Metadaten enthalten beispielsweise Informationen darüber, ob eine Registrierung des WSRP-Konsumenten erforderlich ist oder ob HTTP-*Cookies* initialisiert werden müssen, bevor der WSRP-Konsument mit den WSRP-*Portlets* interagieren kann.

- ***Markup***: Über die *Markup*-Schnittstelle kann ein WSRP-Konsument mit einem WSRP-Produzenten interagieren. Hierzu stellt die Schnittstelle Operationen zur Abfrage des aktuellen *Markup* eines WSPR-*Portlet* und zur Verarbeitung von Benutzerinteraktionen auf diesem *Markup* zur Verfügung.

Neben den beiden erforderlichen Schnittstellen *Service Description* und *Markup* sieht der Standard zwei weitere optionale Schnittstellen vor [OASIS-WSRP2, AA+05]:

- ***Registration***: Über die Registrierungs-Schnittstelle kann der WSRP-Produzent fordern, dass ein potenzieller WSRP-Konsument sich zunächst registrieren muss, bevor eine Interaktion über die beiden Schnittstellen *Service Description* und *Markup* gestattet wird. Durch eine Registrierung wird es dem WSRP-Produzenten möglich, seine WSRP-*Portlets* individuell an den WSRP-Konsumenten anzupassen und beispielsweise gewissen WSRP-Konsumenten nur einen gewissen Teil seiner WSRP-*Portlets* zugänglich zu machen oder den Zugriff sogar vollständig zu verweigern. Dementsprechend stellt diese Schnittstelle verschiedene Operationen zur Initialisierung, Abänderung und Löschung von Registrierungen bereit. Jede Registrierung repräsentiert eine spezielle Beziehung zwischen einem WSRP-Konsumenten und einem WSRP-Produzenten.

- ***Portlet Management***: Mittels dieser Schnittstelle stehen dem WSRP-Konsumenten Operationen zur Verfügung, mit denen er den Lebenszyklus eines WSRP-*Portlet*, das im WSRP-Produzenten ausgeführt wird, beeinflussen kann. Ein WSRP-Konsument kann daher über diese Schnittstelle das Verhalten eines WSRP-*Portlet* beeinflussen oder gar eine konkrete Instanz eines WSRP-*Portlet* zerstören.

Der Einsatz von WSRP ermöglicht die Wiederverwendung von *Portlets* unabhängig von der zur Implementierung gewählten Programmiersprache. WSRP-*Portlets* werden analog zum Paradigma der Dienstorientierung als präsentationsorientierte Fachfunktionalität an einer Dienstschnittstelle bereitgestellt und können so in unterschiedlichsten Portalen wiederverwendet werden. Da Benutzerschnittstellen einheitliche Gestaltungs- und Layoutrichtlinien zugrunde liegen sollten (vgl. [St07]), muss ein Portal das Aussehen der entfernt laufenden WSRP-*Portlets* beeinflussen können. WSRP definiert daher einige *Cascading Style Sheet*-Klassen [W3C-CSS2.1], die seitens der *Portlet*-Entwickler verwendet werden sollten, um ein einheitliches Layout der einzelnen WSRP-*Portlets* zu erreichen. Die einzelnen CSS-Klassen orientieren sich dabei an den verschiedenen Elementen eines *Portlet*, beispielsweise Verweise, Schriften, Tabellen oder Menüs. Nutzen die *Portlet*-Entwickler die durch WSRP standardisierten CSS-Klassen, so kann der WSRP-Konsument über alle WSRP-*Portlets* hinweg, die von unterschiedlichen WSRP-Konsumenten angeboten werden, ein einheitliches Layout erreichen.

2.5 Zusammenfassung

In diesem Kapitel wurden die für das Verständnis dieser Arbeit notwendigen Grundlagen eingeführt. Ausgehend von Modellen und Metamodellen, die den formalen Kern der Arbeit bilden, wurden die Konzepte der Geschäftsprozesse und Workflows eingeführt und im Kontext der geschäftsgetriebenen Entwicklung ausgeprägt. Ziel der geschäftsgetriebenen Entwicklung ist die Abbildung von Geschäftsprozessen auf eine geeignete IT-Unterstützung. Da diese Arbeit ein besonderes Augenmerk auf die Aspekte der Benutzerinteraktion sowohl in der Modellierung als auch in der IT-Unterstützung legt, wurde zunächst die formale Spezifikation grafischer Benutzerschnittstellen durch Modelle beleuchtet.

Anschließend folgte eine Einführung des Dienstbegriffs und der dienstorientierten Architektur als eine flexible Softwarearchitektur zur Unterstützung von Geschäftsprozessen. Ein besonderer Schwerpunkt bei der Betrachtung der dienstorientierten Architektur lag erneut auf den Aspekten der Benutzerinteraktion. Das sich anschließende Kapitel baut auf diesen Grundlagen auf und untersucht in deren Kontext den aktuellen Stand der Forschung.

3 Stand der Forschung

Das Ziel dieser Arbeit ist die Abbildung von Geschäftsprozessen unter der Berücksichtigung der Benutzerinteraktion auf eine dienstorientierte Architektur. Ausgehend von dieser Zielsetzung und den in Abschnitt 1.3 identifizierten Problemstellungen motiviert dieses Kapitel Anforderungen an eine benötigte Lösung und fasst diese in einem Anforderungskatalog zusammen. Der Anforderungskatalog dient anschließend als Grundlage zur Untersuchung und Bewertung bestehender Forschungsansätze aus dem Umfeld der Arbeit. Ein abschließendes Resümee fasst die Ergebnisse der Untersuchung verwandter Forschungsansätze zusammen und stellt den Handlungsbedarf für diese Arbeit heraus.

3.1 Anforderungen

Die bei der Abbildung von Geschäftsprozessen mit Benutzerinteraktion auf eine dienstorientierte Architektur identifizierten und als wesentlich erachteten Problemstellungen werden in Abschnitt 1.3 vorgestellt. Anhand dieser Problemstellungen lassen sich Anforderungen an eine Lösung motivieren, die zwei Kategorien zugeteilt werden können. Die erste Kategorie umfasst Anforderungen an die Spezifikation der Aspekte der Benutzerinteraktion und deren Umsetzung auf ausführbare Softwareartefakte. Die zweite Kategorie stellt Anforderungen an eine dienstorientierte Architektur, die zur Realisierung einer flexiblen IT-Unterstützung benötigt wird. Die folgenden Abschnitte geben entsprechend dieser Kategorisierung die Anforderungen der Arbeit wieder.

3.1.1 Spezifikation der Benutzerinteraktion

Damit die verschiedenen Aspekte der Benutzerinteraktion integriert im Rahmen eines modellgetriebenen Entwicklungsvorgehens berücksichtigt werden können, bedarf es formaler, ausdrucksstarker Spezifikationssprachen, die eine Spezifikation der Anforderungen der Benutzerinteraktion durch Modelle ermöglichen. Im dabei verfolgten Entwicklungsvorgehen muss zusätzlich auf die Portabilität der entwickelten Modelle geachtet werden, um der heute vorhandenen Heterogenität der IT-Systeme im Bereich der Benutzerinteraktion entgegenwirken zu können. Da ferner mit der Berücksichtigung der Benutzerinteraktion eine nicht unerhebliche Steigerung der Komplexität des gesamten Entwicklungsvorgehens einhergeht, muss das verfolgte Entwicklungsvorgehen geeignete Mechanismen zur Reduktion der Komplexität vorsehen. Diese wesentlichen Anforderungen an die Spezifikation der Benutzerinteraktion können wie folgt präzisiert werden.

Anforderung A1.1: Integration in gemeinsames Vorgehensmodell

Ohne ein zugrunde liegendes Vorgehensmodell ist die Durchführung eines Projektes zur Entwicklung eines Anwendungssystems bei der heute vorhandenen Komplexität dieser Systeme nicht sinnvoll durchführbar. Um die Nachvollziehbarkeit und Konsistenz der zu entwickelnden Softwareartefakte eines Anwendungssystems gewährleisten zu können, ist die Ausrichtung an einem durchgängigen Vorgehensmodell sogar unabdingbar [So06]. Demnach müssen auch diejenigen Entwicklungsaktivitäten, die durch die Berücksichtigung der Aspekte der Benutzerinteraktion zusätzlich erforderlich werden, in ein gemeinsames Vorgehensmodell integriert werden, um die Nachvollziehbarkeit und Konsistenz der entstehenden Softwareartefakte der Benutzerinteraktion ebenfalls gewährleisten zu können [Bi06, WF+05, Ko01].

Anforderung A1.2: Plattformunabhängigkeit

Die benötigten Modelle zur Spezifikation der Benutzerinteraktion müssen von jeglichen Details abstrahieren, die nur durch die verwendete Plattform impliziert werden. Der Plattformbegriff fasst in dieser Arbeit analog zur Definition in [MM03] und [KW+03] die vorhandenen Softwarearchitekturen, wie beispielsweise dienstorientierte Architekturen, sowie die unterschiedlichen Rahmenwerke für die Softwareentwicklung wie JEE oder .NET zusammen. Die Forderung nach einer plattformunabhängigen Betrachtung der Modelle leitet sich, wie beispielsweise [PS+08] oder [Bi06] konstatieren, aus der Anzahl der sich im Einsatz befindenden Plattformen und Endgeräte ab, die speziell im Kontext der Benutzerinteraktion stetig zunimmt und somit den Komplexitätsgrad der Softwareentwicklung kontinuierlich steigert. Um diese Heterogenität überwinden zu können, ist eine analytische und plattformunabhängige Betrachtung der Benutzerinteraktion notwendig [PT+07, TP+07, MA+06, HT06]. Durch die hiermit angestrebte Portabilität der Modelle kann im Umkehrschluss eine multimodale Partizipation des Menschen ermöglicht und zusätzlich eine Reduktion der Komplexität im Entwicklungsvorgehen erreicht werden [PP03, AI05].

Anforderung A1.3: Spezifikation der Geschäftsanforderungen

Die in Unternehmen ausgeführten Geschäftsprozesse müssen unterschiedlichen Geschäftsanforderungen genügen. Um diese Geschäftsanforderungen präzise auf die IT in Form von Workflows abbilden zu können, haben sich vier zentrale Anforderungsperspektiven für Workflows etabliert [KV06, AH+03] (vgl. Abschnitt 2.2.3). Diese Anforderungsperspektiven werden von bestehenden Spezifikationssprachen nicht oder nur unzureichend berücksichtigt [RA+05, WA+06, RA+06a]. Wie bereits deutlich gemacht wurde, wirkt sich die Berücksichtigung der Aspekte der Benutzerinteraktion auf alle Perspektiven eines Workflows, von der Ressourcenperspektive, in welcher der Mensch als Ressource erfasst werden muss, bis hin zur Datenperspektive, in welcher die durch den Menschen zu manipulierenden Daten konkretisiert werden müssen, aus. Um diese Auswirkungen nachvollziehbar in einem modellgetriebenen Entwicklungsvorgehen erfassen zu können, wird, wie auch [KV06] fordert, eine ausdrucksmächtige Spezifikationssprache benötigt, welche die unterschiedlichen Perspektiven eines Workflows berücksichtigt. Zusätzlich muss die Spezifikationssprache die Überprüfung der entwickelten Modelle auf syntaktische Korrektheit durch ein Metamodell unterstützen und durch eine grafische Notation zur Abschwächung der Komplexitätssteigerung, die durch die zusätzliche Berücksichtigung der verschiedenen Perspektiven eines Workflows hervorgerufen wird, beitragen (vgl. [Ha05, Ko01, PS+08]).

Anforderung A1.4: Aspekte einer Benutzerschnittstelle

Menschen werden zur Abarbeitung von Arbeitseinheiten in Workflows benötigt. Um die hierzu notwendige Interaktion des Menschen mit den IT-Systemen zu ermöglichen, bedarf es einer Benutzerschnittstelle [KF93]. Eine solche Benutzerschnittstelle umfasst, wie Abschnitt 2.3 erläutert, unterschiedliche dynamische und statische Aspekte, die bei der Entwicklung der Benutzerschnittstelle berücksichtigt werden müssen. Die benötigte Spezifikationssprache muss daher in Anlehnung an [KK08] und [MF+07] ausdrucksmächtige Modellelemente für eine nachvollziehbare Spezifikation des Inhalts-, Präsentations- und Navigationsaspektes einer Benutzerschnittstelle in einem modellgetriebenen Entwicklungsvorgehen bereitstellen. Analog zu Anforderung A1.3 muss die Spezifikationssprache den Entwickler bei der Überprüfung der Modelle auf syntaktische Korrektheit durch ein Metamodell unterstützen und durch eine grafische Notation die Komplexitätssteigerung, die durch die Berücksichtigung der Aspekte der Benutzerschnittstelle im Entwicklungsvorgehen entsteht, mindern.

Anforderung A1.5: Integration der Fachentwickler

Die Entwicklungsaktivitäten, die zur Berücksichtigung der Benutzerinteraktion in einem Entwicklungsvorgehen zusätzlich notwendig werden, müssen entsprechend Anforderung 1.1 in das gesamte Entwicklungsvorgehen integriert werden, um die Qualität der entwickelten Softwareartefakte gewährleisten zu können. Wie in Anforderung 1.3 und 1.4 festgehalten, müssen dabei unterschiedliche Aspekte der Benutzerinteraktion durch Modelle spezifiziert werden. Damit die Komplexität dieser Modelle für jeden einzelnen Entwickler beherrschbar bleibt, muss das modellgetriebene Entwicklungsvorgehen den Entwicklern nach dem Prinzip der Trennung der Zuständigkeiten (engl. *Separation of Concerns*) (vgl. [FR07, CF+00, Di82]) für den jeweils betrachteten Aspekt der Benutzeraktion angepasste und geeignete Modelle zur Verfügung stellen [SS+07b]. Durch dieses Prinzip wird auch die Integration von Fachentwicklern vereinfacht, die zur Umsetzung einzelner Teilaspekte, wie beispielsweise dem Layout der Benutzerschnittstelle, besonders qualifiziert sind. Gleichzeitig kann durch die Qualifikation der Fachentwickler die Qualität der entwickelten Softwareartefakte weiter gesteigert werden. Ferner gestattet eine getrennte Zuständigkeit der Fachentwickler eine entkoppelte Umsetzung einzelner Teilaspekte der Benutzerinteraktion auf lauffähige Softwareartefakte, sodass eine frühzeitige Verifikation der Umsetzung einzelner Teilaspekte gegenüber dem Kunden ermöglicht und das Risiko einer Fehlentwicklung gemindert werden kann.

Anforderung A1.6: Automatisierte Ausführung der Abbildung

Die Abbildung eines Geschäftsprozesses auf die IT-Systeme der IT-Unterstützung erfolgt im Kontext dieser Arbeit entlang der durch die modellgetriebene Architektur vorgegebenen Modellhierarchie. Eine manuell durch den Menschen durchgeführte schrittweise Abbildung der Modelle stellt für das Entwicklungsvorgehen ein hohes Fehlerpotenzial dar, da die Abbildung der Modelle alleine auf deren Interpretation durch den Menschen beruht. Dieses Fehlerpotenzial muss durch den Einsatz automatisierter Transformationen, die nach den Prinzipien der modellgetriebenen Entwicklung auf den bestehenden Modellen operieren, reduziert werden [PM06, SV05] (vgl. Abschnitt 2.1.2). Durch modellbasierte Transformationen wird die Nachvollziehbarkeit der einzelnen Abbildungen erhöht und durch deren automatisierte Ausführung die Gesamtkomplexität des Entwicklungsvorgehens reduziert. Werden die benötigten Transformationsregeln konform zu bestehenden Standards entwickelt, kann zusätzlich der Heterogenität der vorhandenen Entwicklungswerkzeuge begegnet werden.

3.1.2 Dienstorientierte Architektur zur Unterstützung der Benutzerinteraktion

Um die zur Spezifikation der Benutzerinteraktion verwendeten Modelle entlang eines modellgetriebenen Entwicklungsvorgehens auf eine dienstorientierte Architektur abbilden zu können, wird eine geeignete dienstorientierte Architektur als IT-Unterstützung benötigt. Bestehende dienstorientierte Architekturen unterstützen die Interaktion des Menschen per se jedoch nicht und müssen daher zunächst um zusätzliche Fachfunktionalität erweitert werden. Um eine dienstorientierte Architektur trotz dieser Erweiterungen weiterhin als flexible IT-Unterstützung einsetzen zu können, muss der Sicherstellung der Interoperabilität im Hinblick auf die zusätzlich benötigte Fachfunktionalität ein besonderer Stellenwert beigemessen werden. Diese Anforderungen an eine dienstorientierte Architektur können wie folgt konkretisiert werden.

Anforderung A2.1: Unterstützung von Benutzerinteraktion

Die Unterstützung von Geschäftsprozessen mit Benutzerinteraktion muss durch eine geeignete dienstorientierte Architektur ermöglicht werden, welche die spezifischen Anforderungen der Benutzerinteraktion berücksichtigt. In Anlehnung an die in Abschnitt 2.2 eingeführten Konzepte des Workflow-Referenzmodells der *Workflow Management Coalition* (WfMC) [WfMC-WRM1.1] müssen hierzu zwei funktionale Anforderungen gesondert betrachtet werden. Einerseits findet die Interaktion eines Benutzers immer über eine Benutzerschnittstelle statt, weshalb die dienstorientierte Architektur dementsprechend eine Benutzerschnittstelle zu Verfügung stellen muss. Andererseits müssen die verschiedenen Benutzeraufgaben unterschiedlicher Benutzer durch die dienstorientierte Architektur verwaltet und gesteuert werden können. Hierzu ist im Vergleich zu bestehenden Lösungen, wie sie auszugsweise in [KB+05] oder [Li07] gegeben werden, eine Erweiterung um zusätzliche Architekturkomponenten zur Unterstützung dieser funktionalen Anforderungen notwendig [TP+07, RM06].

Anforderung A2.2: Interoperable Dienstschnittstellen

Interoperable Dienstschnittstellen sind eine Grundvoraussetzung, die zum gegenwärtigen Erfolg der dienstorientierten Architekturen beitragen und eine flexible IT-Unterstützung von Geschäftsprozessen ermöglichen [DJ+05]. Zur Unterstützung von Geschäftsprozessen, die der Interaktion eines Menschen bedürfen, werden zusätzliche Architekturkomponenten in einer dienstorientierten Architektur benötigt (Anforderung A2.1). Die Fachfunktionalität, die durch diese zusätzlichen Architekturkomponenten bereitgestellt wird, kann bedingt durch die heute vorhandene Heterogenität der IT-Landschaften durch vielerlei IT-Systeme erbracht werden. Um eine flexible Unterstützung der Geschäftsprozesse auch unter Berücksichtigung der Benutzerinteraktion aufrechterhalten zu können, muss das Prinzip der Dienstorientierung, wie auch [TP+07, AL+07, AA+05, DR04] fordern, konsequent auf alle neu benötigten Architekturkomponenten angewendet werden. Die zur Unterstützung der Benutzerinteraktion zusätzlich benötigte Fachfunktionalität muss daher lose gekoppelt und durch standardisierte Dienstschnittstellen bereitgestellt werden, um somit eine durchgängige Interoperabilität über alle logischen Schichten einer dienstorientierten Architektur hinweg erreichen zu können.

Anforderung A2.3: Integrationsfähigkeit und Flexibilität

Wie Anforderung A2.1 festhält, müssen bestehende Referenzarchitekturen einer dienstorientierten Architektur erweitert werden, um als Ausführungsumgebung für Geschäftsprozesse dienen zu können, die der Interaktion des Menschen bedürfen. Die hierzu notwendige Fachfunktionalität muss, wie in Anforderung A2.2 gefordert, über interoperable Schnittstellen bereitgestellt werden. Zusätzlich muss beachtet werden, dass eine dienstorientierte Architektur in unterschiedlichsten Anwendungsszenarien zum Einsatz kommen kann, in denen der Umfang der zusätzlich benötigten Fachfunktionalität variieren kann. Daher fordern [TP+07, Li07, BH04], die zur Unterstützung der Benutzerinteraktion notwendige Fachfunktionalität, wie beispielsweise eine Benutzeraufgabenverwaltung, als jeweils eigenständige Architekturkomponenten im Sinne einzelner Dienste zu betrachten. Hierdurch kann die Konzeption der dienstorientierten Architektur auf das jeweilige Anwendungsszenario angepasst und auf die benötigte Fachfunktionalität beschränkt werden. Neben der reduzierten Gesamtkomplexität der dienstorientierten Architektur kann ferner die Heterogenität bestehender IT-Landschaften durch die Integration mehrerer spezialisierter Fachkomponenten je nach benötigtem Umfang der Fachfunktionalität adressiert werden.

3.1.3 Anforderungskatalog

Tabelle 1 fasst die in den vorangegangenen Abschnitten spezifizierten Anforderungen an eine benötigte Lösung in Form eines Anforderungskatalogs zusammen. Dieser dient im weiteren Verlauf der Arbeit einerseits als Grundlage der Bewertung verwandter Forschungsansätze sowie anderseits zur Ausrichtung und Bewertung der Beiträge, die in dieser Arbeit entwickelt werden.

Anforderungskatalog	
A1 – Spezifikation der Benutzerinteraktion	
A1.1	Integration in gemeinsames Vorgehensmodell
A1.2	Plattformunabhängigkeit
A1.3	Spezifikation der Geschäftsanforderungen
A1.4	Aspekte einer Benutzerschnittstelle
A1.5	Integration der Fachentwickler
A1.6	Automatisierte Ausführung der Abbildung
A2 – Dienstorientierte Architektur zur Unterstützung der Benutzerinteraktion	
A2.1	Unterstützung von Benutzerinteraktion
A2.2	Interoperable Dienstschnittstellen
A2.3	Integrationsfähigkeit und Flexibilität

Tabelle 1: Anforderungskatalog

3.2 Spezifikation von Benutzerinteraktion

Das Ziel dieser Arbeit, die spezifischen Anforderungen der Benutzerinteraktion durch Modelle spezifizieren und auf eine geeignete IT-Unterstützung abbilden zu können, wird ebenfalls durch verschiedene verwandte Forschungsansätze verfolgt, die in den folgenden Abschnitten untersucht und entsprechend der Kategorie A1 des Anforderungskatalogs bewertet werden. Die betrachteten Forschungsansätze können dabei zwei eigenständigen Forschungsbereichen zugeordnet werden. Abschnitt 3.2.1 diskutiert zunächst Arbeiten aus dem Forschungsbereich der Softwaretechnik (engl. *Software Engineering*), während Abschnitt 3.2.2 Arbeiten aus dem Forschungsbereich des *Web Engineering* untersucht.

3.2.1 Forschungsansätze aus dem Bereich der Softwaretechnik

Die im Folgenden diskutierten Forschungsansätze fokussieren die Spezifikation verschiedener Aspekte der Benutzerinteraktion durch Modelle entlang eines systematischen Vorgehensmodells zur Softwareentwicklung und weisen daher eine hohe Relevanz für die Zielsetzung dieser Arbeit auf.

3.2.1.1 Almendros-Jiménez et al.: Designing GUI components from UML Use Cases

Die Spezifikation von Benutzerschnittstellen, ausgehend von einem UML-Anwendungsfalldiagramm, ist das Ziel des in [AI05, AI04a, AI04b] präsentierten Ansatzes. In einer ersten grobgranularen Spezi-

fikation werden entsprechend der Semantik eines UML-Anwendungsfalldiagramms die beteiligten Akteure sowie die zu unterstützenden Anwendungsfälle des betrachteten Anwendungssystems erfasst. In einem zweiten Entwicklungsschritt folgt für jeden Anwendungsfall die Erstellung eines oder mehrerer UML-Aktivitätsdiagramme, die das zum jeweiligen Anwendungsfall gehörende dynamische Systemverhalten detaillierter beschreiben. In Rückkopplung dazu sieht der Ansatz eine Verfeinerung des UML-Anwendungsfalldiagramms vor mit dem Ziel, einen Anwendungsfall direkt auf eine Benutzerschnittstelle in Form eines Java-*Applet* oder eines *Frames* abbilden zu können. Hierzu nutzt der Ansatz die *Include*- und Generalisierungsbeziehungen, welche die UML zwischen Akteuren und Anwendungsfällen vorsieht. Wird ein Anwendungsfall durch einen zweiten Anwendungsfall spezialisiert, so übernimmt dieser nicht nur das Verhalten, sondern auch die Benutzerschnittstelle des ersten Anwendungsfalls. Das Systemverhalten eines Anwendungsfalls wird durch den Ansatz mit Bezug auf die angestrebte Benutzerschnittstelle beschrieben. Hierzu nutzen die Autoren eigene Stereotypen basierend auf JavaSwing zur Erweiterung eines UML-Aktivitätsdiagramms. Abbildung 18 zeigt als Beispiel das benutzerzentrische Systemverhalten des Anwendungsfalls *Manage shopping cart*, durch das die benötigten Elemente der zugeordneten Benutzerschnittstelle spezifiziert werden. Die Stereotypen *JTextArea* oder *JButton* entsprechen den gleichnamigen Elementen aus JavaSwing.

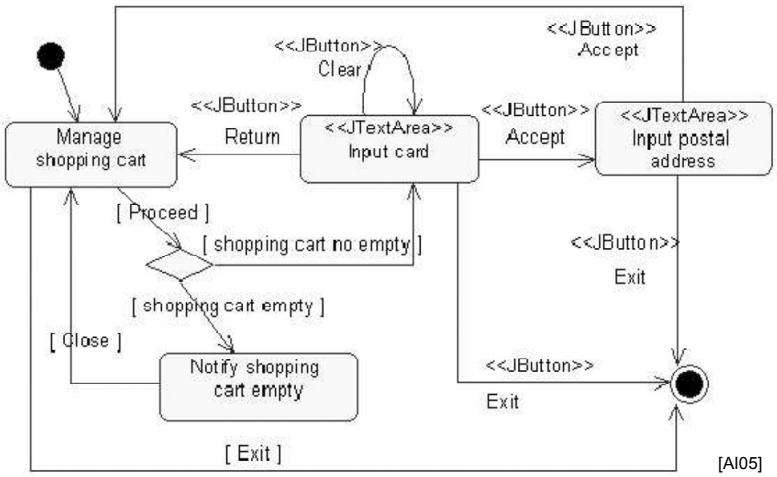

Abbildung 18: Stereotypisiertes UML-Aktivitätsdiagramm

Die einzelnen Benutzerschnittstellenelemente wie *JButton* oder *JTextArea* werden pro Anwendungsfall innerhalb einer als *Applet* realisierten Benutzerschnittstellenkomponente zusammengefasst. Abschließend erfolgt aus einem Anwendungsfall und den diesem zugeordneten Aktivitätsdiagrammen die Transformation auf ein die Benutzerschnittstelle spezifizierendes Klassendiagramm. Der Ansatz liefert zusätzlich eine mathematische Formalisierung der verwendeten Diagrammtypen, um einerseits die aus Sicht der Autoren unpräzise Semantik der in einem UML-Anwendungsfall- und UML-Aktivitätsdiagramm verwendeten Modellelemente zu präzisieren und um andererseits eine Definition für eine Benutzerschnittstelle aus Sicht der Autoren geben zu können. Ebenso ist diese Formalisierung die Grundlage für ein anschließend eingeführtes Transformationsregelwerk.

Bewertung des Ansatzes

Das präsentierte Vorgehen fokussiert die Entwicklung von Benutzerschnittstellen für JavaSwing auf Basis von UML-Modellen. Die verwendeten Modellelemente und Diagramme basieren auf dem Metamodell der UML und einem UML-Profil, wodurch eine nachvollziehbare Integration des Ansatzes in bestehende Entwicklungsvorgehen theoretisch möglich ist, von den Autoren in der Praxis aber noch nicht umgesetzt wurde (A1.1). Obwohl die Spezifikation der Anwendungsfälle noch ohne den

Bezug zu einer konkreten Plattform erfolgt, ist durch die anschließende Abbildung der Anwendungsfälle auf für JavaSwing stereotypisierte UML-Aktionen die Plattformunabhängigkeit der Spezifikationen nicht gegeben (A1.2). Die durch ein UML-Anwendungsfalldiagramm erfassten Anwendungsfälle der zu entwickelnden Anwendung werden ferner als Ausgangspunkt der vorgestellten Erweiterungen verwendet, sodass die Spezifikation weiterer Geschäftsanforderungen durch den Ansatz nicht betrachtet wird (A1.3). Ausgehend von einem Anwendungsfallmodell konzentriert sich der Ansatz auf das Navigations- und Präsentationsmodell einer Benutzerschnittstelle und blendet das Inhaltsmodell aus (A1.4). Hierbei bleibt die Frage nach einer syntaktisch korrekten Verwendung der für das Navigations- und Präsentationsmodell spezifizierten Modellelemente durch den nicht näher erläuterten Bezug zum Metamodell der UML unbeantwortet. Dennoch ermöglicht die getrennte Erfassung beider Aspekte der Benutzerschnittstelle (Navigation und Präsentation) in eigenständigen Modellen die Integration spezialisierter Fachentwickler (A1.5). Da nach Aussage der Autoren die Modelle der UML in ihrer Semantik nicht eindeutig beschrieben sind, stellen die Autoren die verwendeten Modelle auf eine eigene formale mathematische Grundlage. Auf dieser Grundlage aufbauend werden die benötigten Regeln für eine automatisierte Abbildung der Modelle zwar zunächst beschrieben, eine automatische Ausführung der Abbildung wird dann jedoch nicht näher erläutert (A1.6).

3.2.1.2 Pinheiro da Silva et al.: UML for interactive systems (UMLi)

UML for interactive systems (UMLi) [PP03, PP02, PP00] erweitert die UML mit dem Ziel, eine Verbesserung der Modellierung von interaktiven Systemen zu erreichen. Unter einem interaktiven System versteht UMLi eine Anwendung, welche die Interaktion des Menschen berücksichtigt. Das von UMLi zur Entwicklung einer Anwendung verfolgte Vorgehen beginnt mit der Anforderungsspezifikation durch ein UML-Anwendungsfalldiagramm, das als Ausgangspunkt für alle weiteren Modelle dient, die durch UMLi betrachtet werden. Der Ansatz unterscheidet zunächst zwischen drei unterschiedlichen Modellen zur Spezifikation der Benutzerschnittstelle, die auf die in Abschnitt 2.3.1 eingeführten Modelle zur Spezifikation des Inhalts, der Präsentation und der Navigation zurückgeführt werden können. Zur Spezifikation des Inhalts verwendet UMLi ein UML-Klassendiagramm, das die Entitäten der interaktiven Anwendung in Form eines Domänenmodells spezifiziert. Die Darstellung dieser Entitäten gegenüber dem Menschen wird im Präsentationsmodell in Form eines weiteren UML-Klassendiagramms, das die Struktur der verwendeten Elemente spezifiziert, näher beleuchtet. Da nach UMLi die UML für die Spezifikation des Präsentationsmodells jedoch keine ausreichende Unterstützung liefert, erweitert UMLi die UML um einen neuen als Benutzerschnittstellendiagramm (engl. *User Interface Diagram*) bezeichneten Diagrammtyp. Als eine Spezialisierung des UML-Klassendiagramms stellt das Benutzerschnittstellendiagramm wie in Abbildung 19 beispielhaft gezeigt eine eigene grafische Notation zur Spezifikation des Strukturmodells bereit.

Ein als UML-Aktivitätsdiagramm spezifiziertes Verhaltensmodel (engl. *Behavior Model*) fokussiert schließlich die dynamischen Aspekte der Anwendung und beschreibt das Verhalten der Anwendung zu einem konkreten Anwendungsfall näher. Ferner stellt das Verhaltensmodell einer Benutzerschnittstelle den Bezug zwischen den Modellelementen des Domänen- und Benutzerschnittstellendiagramms her, indem es die für die Beschreibung des Verhaltens benötigten Entitäten und Benutzerschnittstellenelemente über die einzelnen Aktionen miteinander verknüpft. Zur Entwicklung der Modelle kommt nach [PP01] das als Erweiterung von AgroUML [Tig-AgroUML] realisierte Modellierungswerkzeug AgroUMLi zum Einsatz, das die Verwendung der neu bereitgestellten Stereotypen und die Erstellung des neuen Benutzerschnittstellendiagramms ermöglicht und durch einen grafischen Editor unterstützt. Abbildung 19 zeigt das Strukturmodell einer Anmeldemaske als Benutzerschnittstellendiagramm.

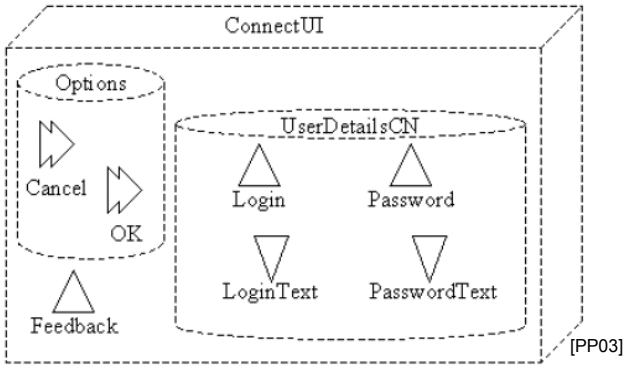

Abbildung 19: Benutzerschnittstellendiagramm nach UMLi

Bewertung des Ansatzes

Die Autoren nutzen zur Unterstützung der Entwicklung das Werkzeug AgroUMLi, wodurch eine Verknüpfung der einzelnen Modelle untereinander zu erkennen ist. Dennoch lässt der Ansatz die Frage nach einem genauen Vorgehensmodell, in dem die betrachteten Modelle integriert entwickelt werden, offen (A1.1). Die verschiedenen Modelle, die zur Spezifikation der durch UMLi betrachteten Aspekte der Benutzerinteraktion verwendet werden, abstrahieren von plattformspezifischen Details, wodurch eine plattformunabhängige Betrachtung der behandelten Aspekte der Benutzerinteraktion gegeben ist (A1.2). Hierbei betrachtet UMLi die Spezifikation der Geschäftsanforderungen nicht und konzentriert sich auf die Benutzerschnittstelle eines IT-Systems (A1.3). Von den hierzu benötigten drei Modelltypen fokussiert UMLi die integrierte Entwicklung des Präsentations- und Navigationsmodells, trifft aber zum Inhaltsmodell und dessen Spezifikation keine weiteren Aussagen. Zur Spezifikation des Präsentationsmodells nutzt UMLi einen eigenen als Benutzerschnittstellendiagramm bezeichneten Diagrammtyp, dessen Metamodell durch die Autoren nicht näher beschrieben wird und daher die Frage nach dessen Eignung zur Unterstützung des Entwicklers zur Erstellung syntaktisch korrekter Modelle offenlässt (A1.4). Da sowohl das Navigations- als auch das Präsentationsmodell von UMLi als jeweils eigenständige Modelle spezifiziert werden, ist eine Integration entsprechender Fachentwickler zur Spezifikation und Weiterentwicklung dieser Modelle der Benutzerschnittstelle möglich (A1.5). Eine automatisierte modellbasierte Abbildung wird durch UMLi nicht betrachtet (A1.6).

3.2.1.3 Lorenz: Anpassung von UML-Aktivitäten an den Prozess der Webapplikationsentwicklung

Lorenz präsentiert in [Lo06] einen Ansatz zur Erweiterung der UML, der eine bessere Eignung der UML-Aktivitätsdiagramme zur Spezifikation der Aspekte der Benutzerinteraktion anstrebt. Hierzu führt der Ansatz als Spezialisierung der UML-Aktion zwei zusätzliche, als System- und Akteuraktion bezeichnete Modellelemente ein. Diese gestatten eine bessere Unterscheidung zwischen Aktionen, die durch ein Anwendungssystem alleine oder durch einen menschlichen Benutzer mit Unterstützung der IT-Systeme durchzuführen sind. Dementsprechend wird ein durch ein Anwendungssystem zu unterstützender Workflow unter der Zuhilfenahme der beiden neuen Modellelemente System- und Akteuraktion als UML-Aktivitätsdiagramm modelliert. Im nächsten Entwicklungsschritt verwendet Lorenz eine UML-Annotation, um die Benutzerschnittstelle zu einer Akteuraktion zu spezifizieren. Diese Spezifikation erfolgt dabei in einem Notizentext der als Szene stereotypisierten UML-Annotation, die spezielle Schlüsselwörter wie beispielsweise „anzuzeigende Attribute“ oder „Buttons / Links” umfasst. Als zusätzliche Erweiterung führt [Lo06] die Unterscheidung zwischen anforderungsorientierten und entwurfsorientierten UML-Aktivitätsdiagrammen ein. Während das anforderungsorientierte

Aktivitätsdiagramm in der Anforderungsanalyse zum Einsatz kommt und damit nach [Lo06] einen abstrakteren Charakter haben muss, ist das entwurfsorientierte Aktivitätsdiagramm für den Einsatz in der Entwurfsphase eines Softwareentwicklungsprozesses konzipiert. Die Einfachheit der anforderungsorientierten Aktivitätsdiagramme ergibt sich nach Lorenz unter anderem daraus, dass Systemaktionen, die für den Benutzer von keinem oder nur geringem Interesse sind, einfach ausgeblendet werden. Die Überführung eines anforderungsorientierten Aktivitätsdiagramms in ein entwurfsorientiertes Aktivitätsdiagramm wird durch den Entwickler nach der Verifikation der Anforderungen des Kunden durchgeführt. Abbildung 20 zeigt ein aus [Lo06] entnommenes Beispiel für ein entwurfsorientiertes UML-Aktivitätsdiagramm.

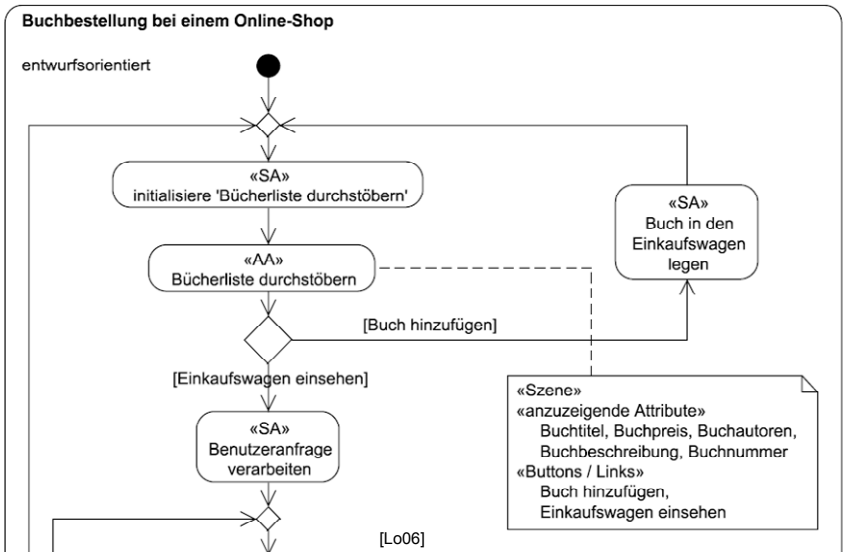

Abbildung 20: Beispiel eines entwurfsorientierten UML-Aktivitätsdiagramms

In einem letzten Entwicklungsschritt überführt der Ansatz das plattformunabhängig spezifizierte, entwurfsorientierte UML-Aktivitätsdiagramm durch sieben manuelle Abbildungsschritte auf ein Entwurfsklassendiagramm für J2EE und Struts.

Bewertung des Ansatzes

Der vorgestellte Ansatz folgt einem klar erkennbaren Entwicklungsvorgehen, in dem die zusätzlichen Entwicklungsaktivitäten, die durch die Berücksichtigung der Aspekte der Benutzerinteraktion entstehen, integriert betrachtet werden (A1.1). Das Entwicklungsvorgehen nutzt als Ausgangsmodell der Entwicklung ein Workflow-Modell, das von technischen Details abstrahiert, sodass eine plattformunabhängige Spezifikation der Aspekte der Benutzerinteraktion ermöglicht wird (A1.2). Ferner stellt der Ansatz durch das plattformunabhängige Workflow-Modell den Bezug zu den Geschäftsanforderungen eines Unternehmens über die Kontrollflussperspektive eines Workflows her. Obwohl bereits in den anforderungsorientierten Aktivitätsdiagrammen bereits der direkte Bezug zu den Benutzern über die beiden Stereotypen Akteuraktion und Szene hergestellt wird, nutzt der Ansatz die hier erfassten Informationen im Sinne eines integrierten Entwicklungsvorgehens nicht weiter und stellt daher auch keine Möglichkeit zur Ausprägung dieser Informationen auf die im Rahmen der Benutzerinteraktion zentralen Ressourcenperspektive eines Workflows zur Verfügung (A1.3). Im weiteren Entwicklungsvorgehen konzentriert sich der Ansatz insbesondere auf die Spezifikation einer Benutzerschnittstelle durch Modellelemente. Sowohl im anforderungsorientierten als auch entwurfsorientierten UML-Aktivitätsdiagramm (das Workflow-Modell) werden Modellelemente des Präsentations- und Inhaltsmodells wie „Szene“ oder „anzuzeigende Attribute“ integriert. Das Navigationsmodell hingegen wird

in einem eigenständigen Entwurfsklassendiagramm angedeutet (A1.4). Durch die enge Kopplung des Workflow-Modells an das Präsentations- und Inhaltsmodell ist eine getrennte Betrachtung der Aspekte durch die entsprechenden Fachentwickler in eigenständigen Modellen nicht möglich (A1.5). Die für eine automatische Abbildung der betrachteten Modelle auf die IT-Unterstützung benötigten Transformationsregeln werden durch den Ansatz zwar theoretisch in Form von textuellen Transformationsregeln skizziert, eine automatische Umsetzung dieser Transformationsregeln wird dann jedoch nicht konkretisiert (A1.6).

3.2.1.4 Scogings et al.: Linking Task and Dialogue Modeling – Toward an Integrated Software Engineering Method

Scogings und Phillips adressieren in [SP03] ein integriertes Vorgehen zur Beschreibung von Benutzeraufgaben und Benutzerschnittstellen. Sie stellen dazu eine aus [AS89] entnommene und als *Lean Cuisine+* bezeichnete grafische Notation vor, die neben der eigentlichen Modellierung eine ereignisbasierte Manipulation der Benutzerschnittstelle ermöglicht. Der Ansatz verfolgt hierzu ein Entwicklungsvorgehen, das mit der Erstellung eines gewöhnlichen UML-Anwendungsfalldiagramms beginnt, welches die durch das Anwendungssystem zu unterstützenden Anwendungsfälle näher spezifiziert. Anhand des gewählten Szenarios der Verwaltung eines Bibliothekskatalogs wird gezeigt, wie die identifizierten Anwendungsfälle, wie beispielsweise „Suche nach Buch“, textuell nach einem vorgegebenen Muster detailliert beschrieben werden. In einem nächsten Entwicklungsschritt wird aus diesen Beschreibungen das Domänenmodell der Anwendung in Form eines UML-Klassendiagramms abgeleitet. Hierbei betonen die Autoren, dass das vollständige Domänenmodell aufgrund der Fokussierung auf die Benutzerinteraktion noch nicht benötigt wird. Es stehen nur die Attribute der Domänenklassen im Mittelpunkt, die durch den Benutzer manipuliert werden können. Für jedes Attribut des benutzerzentrischen Domänenmodells, das durch den Benutzer später ausgewählt werden kann, wird ein als *Meneme* bezeichneter Eintrag in einem *Lean Cuisine+*-Diagramm hinzugefügt. Das *Lean Cuisine+*-Diagramm wird anschließend durch sogenannte *Floating Menemes* um einen dynamischen Aspekt erweitert, mit dessen Hilfe die Autoren Aufgaben-Aktions-Folgen modellieren, die direkt aus der textuellen Beschreibung der Anwendungsfälle abgeleitet werden können.

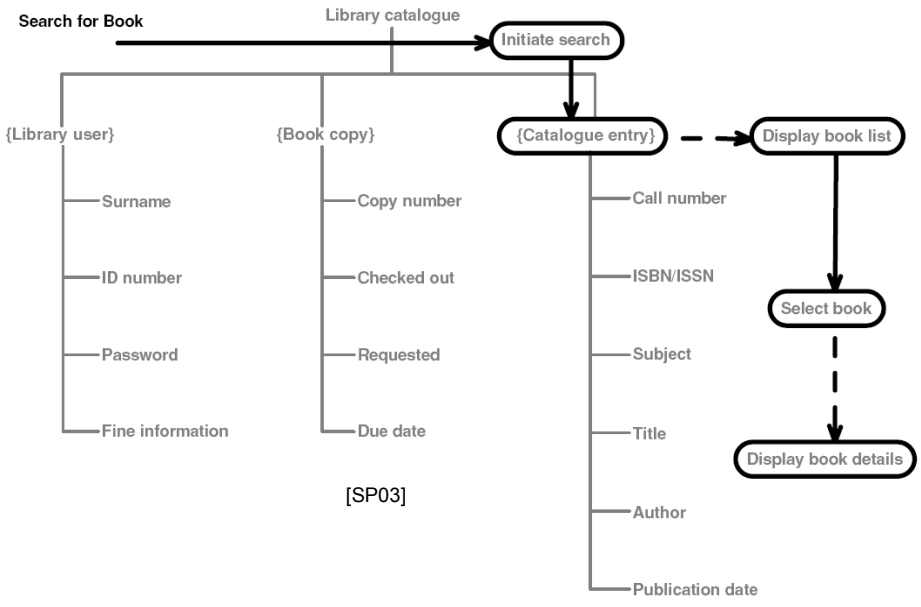

Abbildung 21: Erweitertes Lean Cuisine+-Diagramm zu Anwendungsfall „Suche nach Buch“

Abbildung 21 zeigt das um Aufgaben-Aktionsfolgen erweiterte *Lean Cuisine+*-Diagramm des Anwendungsfalls „Suche nach Buch“, in dem die dargestellten gestrichelten Pfeile jeweils auf eine folgende Systemaktion und die durchgezogenen Pfeile auf eine Benutzeraktion hinweisen. Der Ansatz sieht die Weiterentwicklung des *Lean Cuisine+*-Diagramms in drei iterativen Verfeinerungsschritten vor. In einem ersten Verfeinerungsschritt werden alle *Menemes* entfernt, die in keiner Aufgabe-Aktions-Folge verwendet werden und dadurch für den Benutzer nicht von Belang sind. In einem zweiten Verfeinerungsschritt werden alle Einträge des *Lean Cuisine+*-Diagramms benutzerfreundlich umbenannt, wodurch der Ansatz eine reine Benutzersicht auf einen Anwendungsfall extrahiert. In einem dritten Verfeinerungsschritt wird das *Lean Cuisine+*-Diagramm um sogenannte *Selection Triggers* erweitert, die den Bezug zu den Systemaktionen der Aufgabe-Aktions-Folgen herstellen. Diese Verfeinerung wird wiederholt, bis alle in Bezug auf die Benutzerinteraktion relevanten Informationen aus dem Domänenmodell in einem *Lean Cuisine+*-Diagramm formalisiert sind.

Bewertung des Ansatzes

Scogings und Phillips führen ihren Ansatz entlang eines detailliert beschriebenen Vorgehensmodells ein, das alle Entwicklungsaktivitäten zur Spezifikation der Benutzerinteraktion integriert berücksichtigt und die entwickelten Modelle in klaren Bezug zueinander stellt (A1.1). Die Spezifikation der durch den Ansatz betrachteten Aspekte der Benutzerinteraktion erfolgt durch die UML und die grafische Notationssprache *Lean Cuisine+*, die in [AS89] formal entwickelt und spezifiziert wird. Da sowohl die verwendeten UML-Modelle als auch die *Lean Cuisine+*-Modelle keinen Bezug zu technischen Details einer Plattform herstellen, kann die Anforderung nach einer plattformunabhängigen Spezifikation der Benutzeraktion durch den Ansatz als erfüllt betrachtet werden (A1.2). Ausgangspunkt des Entwicklungsvorgehens sind bereits auf ein Anwendungssystem zugeschnittene Anwendungsfälle. Daher fokussiert der Ansatz eine Anforderungsanalyse für genau ein Anwendungssystem und betrachtet die darüber liegenden Geschäftsprozesse eines Unternehmens nicht (A1.3). Das zu Beginn des Entwicklungsvorgehens spezifizierte Anwendungsfalldiagramm dient als Ausgangspunkt zur Ableitung eines Domänenmodells in Form eines UML-Klassendiagramms. Das Domänenmodell wiederum dient als Ausgangspunkt der Ableitung des in mehreren Stufen verfeinerten *Lean Cuisine+*-Diagramms, das in seiner finalen Version sowohl das Strukturmodell als auch das Navigationsmodell der Benutzerschnittstelle umfasst (A1.4). Durch die hieraus resultierende enge Verzahnung des Präsentations- und Navigationsaspektes der Benutzerschnittstelle in einem gemeinsamen *Lean Cuisine+*-Diagramm ist eine getrennte Betrachtung einzelner Entwicklungsaspekte in je einem eigenständigen Modell durch entsprechende Fachentwickler nicht realisierbar (A1.5). Eine automatisierte Abbildung zwischen den verwendeten UML- und *Lean Cuisine+*-Modellen wird durch den Ansatz nicht beleuchtet (A1.6).

3.2.1.5 Nunes et al.: The Wisdom Approach

Unter dem Projektnamen *The Wisdom Approach*, dessen Ursprung auf den *Workshop on Interactive System Design and Object Models* (WISDOM) [NT+99] zurückzuführen ist, zeigen Nunes und Falcão e Cunha in [NF00] und [NF01] sowie Campos und Nunes in [CN05a] und [CN05b] einen Ansatz zur UML-basierten Modellierung von interaktiven Systemen auf. Die Autoren stellen fest, dass die UML in ihrer ursprünglichen Ausprägung für die Modellierung von interaktiven Systemen nicht die notwendige Unterstützung bietet, um alle Aspekte der Benutzerinteraktion festzuhalten. Ferner konstatieren sie, dass die Modellierung einer Benutzerschnittstelle unter unterschiedlichen Aspekten betrachtet werden muss. Beispielsweise hat der Kunde und spätere Benutzer des zu entwickelnden Systems eine konkrete Vorstellung von der Benutzerschnittstelle. Diese Vorstellung muss durch ein konzeptionelles

Modell der Benutzerschnittstelle erfasst werden, das daher nur einen Bezug zu denjenigen Entitäten der Domäne herstellen darf, die für den Benutzer auch von Bedeutung sind. Aus einem solchen konzeptionellen Modell muss im Anschluss ein konkretes Modell der Benutzerschnittstelle abgeleitet werden, das um zusätzliche, systeminterne Informationen angereichert werden kann. Um diese Unterscheidung zu ermöglichen, trennt der Wisdom-Ansatz die Betrachtung der Aspekte der Benutzerinteraktion von einem allgemeinen Analysemodell, das nach den Prinzipien der objektorientierten Softwareentwicklung die drei Dimensionen Verhalten (*Behavior*), Information (*Information*) und Schnittstelle (*Interface*) umfasst [Ja92], ab und fasst diese in einem eigenständigen Interaktionsmodell zusammen. Diese Auftrennung führt nach [NF00] zu einer Systemmodellierung, die hinsichtlich auftretender Änderungen anpassbar gestaltet werden kann. Das eigenständige Interaktionsmodell umfasst daher ebenfalls die Dimension der Information, muss jedoch nach [NF00] für eine vollständige Betrachtung der Benutzerschnittstelle um zwei weitere getrennt behandelte Dimensionen erweitert werden. Neben der eigentlichen Informationsdimension werden zusätzlich die Präsentations- (*Presentation*) und Dialogdimension (*Dialogue*) im Interaktionsmodell betrachtet. Dem Analysemodell bleibt daher die Spezifikation von reinen Systemschnittstellen, während das Interaktionsmodell Schnittstellen für menschliche Nutzer näher beleuchtet.

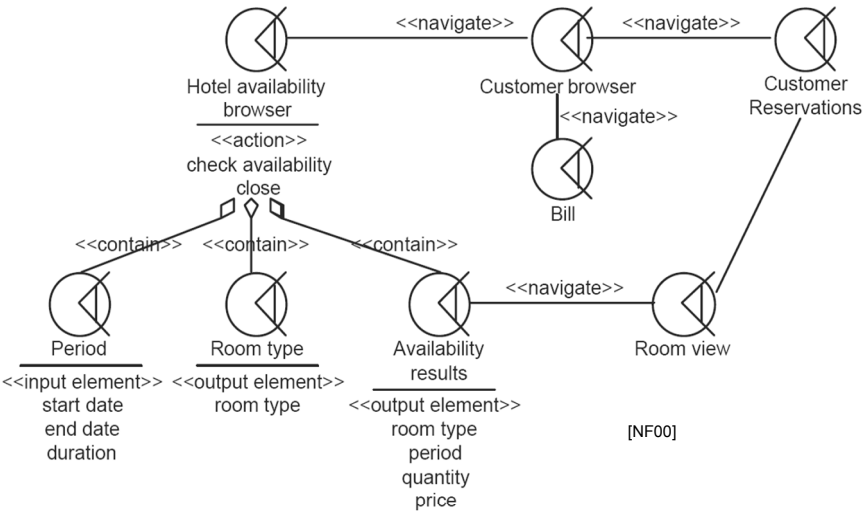

Abbildung 22: Struktur- und Navigationsmodell des Wisdom-Ansatzes

Zur Modellierung der Dialogdimension, welche die einzelnen Benutzerschnittstellen und deren Zusammenhang fokussiert, greift der Ansatz auf die in [Pa04] eingeführten *ConcurTaskTrees* (CTT) zurück. Nunes et al. stellen die zur Modellierung eines *ConcurTaskTrees* benötigten Elemente in Form eines UML-Profils zur Verfügung. In einem weiteren Schritt führen die Autoren das Wisdom-Profil ein, das neben den bereits erwähnten Stereotypen des *ConcurTaskTree*-Profils weitere Stereotypen definiert, mit welchen sowohl das Analyse- als auch das Interaktionsmodell erweitert wird. So sieht der Ansatz für das Präsentationsmodell beispielsweise *Input*- und *Output*-Elemente vor, mit deren Hilfe Ein- und Ausgaben des Benutzers modelliert werden können. Abbildung 22 zeigt ein aus [NF00] entnommenes Struktur- und Navigationsmodell, in dem die Stereotypen des *ConcurTaskTree*- und des Wisdom-Profils angewendet werden.

In [CN05a, CN05b] führen Campos und Nunes den Wisdom-Ansatz fort und erweitern ihn um die von Constantine vorgeschlagenen kanonischen abstrakten Prototypen (engl. *Canonical Abstract Prototypes*) des *User-Centered Design*-Ansatzes [Co03]. Diese basieren auf zwei universellen Objekten, einem generischen Container und einer generischen Aktion. Aus diesen beiden leitet Constantine eine Serie von Elementen ab, welche die Grundlage für die Modellierung einer konkreten Benutzerschnitt-

stelle liefern. Campos greift diesen Ansatz auf und ordnet beide Objekte im Präsentationsmodell des Wisdom-Ansatzes ein. Auf diese Weise präsentieren die Autoren mit CanonSketch ein Werkzeug, mit dem schnelle Entwürfe (engl. S*ketches*) im Sinne von Prototypen der Benutzerschnittstelle erstellt werden können.

Bewertung des Ansatzes

Dem vorgestellten Wisdom-Ansatz liegt ein Vorgehensmodell zugrunde, das alle Entwicklungsaktivitäten zur Berücksichtigung der Benutzerinteraktion integriert und somit eine nachvollziehbare und konsistente Entwicklung ermöglicht (A1.1). Ferner setzen die Autoren auf dem im *Workshop on Interactive System Design and Object Models* [NT+99] spezifizierten Wisdom-Rahmenwerk für interaktive Systeme auf, das alle verwendeten Modelle auf abstrakter Ebene in einen direkten Bezug zueinander setzt (A1.1) und unter anderem auch ein Geschäfts- und Aufgabenmodell umfasst. Dennoch verfolgt der Wisdom-Ansatz das Geschäftsmodell nicht weiter, sondern beginnt das Entwicklungsvorgehen mit der Spezifikation der Anwendungsfälle eines einzelnen Anwendungssystems (A1.3). Zur Spezifikation der verschiedenen Aspekte der Benutzerinteraktion verwendet der Ansatz die UML und stellt zusätzlich eigene Stereotypen durch UML-Profile bereit. Die durch diese UML-Profile zur Verfügung gestellten Modellelemente stellen keinen Bezug zu plattformspezifischen Details her, womit die Plattformunabhängigkeit der entwickelten Modelle uneingeschränkt gegeben ist (A1.2). Zur Spezifikation des Strukturmodells verwendet der Ansatz ein eigenes UML-Profil, das unterschiedliche Stereotypen wie Ein- und Ausgabeelemente zur Modellierung der Struktur einer Benutzerschnittstelle bereitstellt. Das Navigationsmodell spezifiziert der Wisdom-Ansatz durch die Erweiterung des Präsentationsmodells um einen speziellen Stereotyp zur Modellierung von Navigationsbeziehungen. Das Inhaltsmodell schließlich bildet den Übergang zu den Analysemodellen, in denen ein Domänenmodell alle Entitäten des Anwendungssystems umfasst (A1.4). Trotz dieser klaren Trennung der einzelnen Aspekte einer Benutzerschnittstelle lässt der Ansatz einige Fragen nach den konkreten Zusammenhängen der einzelnen Modelle, wie etwa dem Bezug zwischen dem Informations- und Dialogmodell, offen. Der Wisdom-Ansatz macht hier zwar deutlich, dass die verwendeten Modelle über das Wisdom-Rahmenwerk in einem direkten Bezug zueinander stehen, schweigt sich über die konkrete Umsetzung dieses Bezuges jedoch aus, wodurch ein zielgerichteter Einsatz der Fachentwickler nur bedingt ermöglicht wird und die Nachvollziehbarkeit der einzelnen Entwicklungsschritte nicht zwingend gegeben ist (A1.5, A1.1). Aufgrund dieses fehlenden Bezugs wird eine automatisierte Abbildung der verwendeten Modelle entlang eines modellgetriebenen Vorgehens durch den Wisdom-Ansatz nicht betrachtet (A1.6).

3.2.1.6 Sukaviriya et al.: Model-Driven Approach for Managing Human Interface Design Life Cycle

Sukaviriya et al. beschreiben in [SS+07a, SS+07b] einen modellgetriebenen Entwicklungsansatz für webbasierte Portale, der auf den Ideen von [SK+05, SF+93] beruht. Ausgangspunkt des Ansatzes ist ein Geschäftsmodell, das neben dem eigentlichen Kontrollfluss des Geschäftsprozesses detaillierte Informationen bezüglich der Datenperspektive im Sinne der zum Einsatz kommenden Geschäftsobjekte enthält und somit nach [SS+07a] eine optimale Grundlage zur Entwicklung einer Benutzerschnittstelle bietet. Die Autoren betonen, dass ein Modell eines Geschäftsprozesses dieselben Informationen umfasst wie die aus der traditionellen benutzerzentrischen Entwicklung (engl. *User-Centered Design*) bekannten Aufgabenmodelle (vgl. dazu Abschnitt 2.3.1). Daher kann nach [SS+07b] sowohl ein Geschäftsprozessmodell als auch ein traditionelles Aufgabenmodell als Ausgangspunkt der Entwicklung der Benutzerschnittstelle verwendet werden, obwohl das Geschäftsprozessmodell einen direkte-

ren Bezug zur Geschäftsdomäne herstellt. Zur Spezifikation des Geschäftsmodells setzen die Autoren die plattformspezifische Modellierungssprache des Websphere Business Modeler von IBM [IBM-WBM] ein. Das Geschäftsmodell wird in einem nächsten Entwicklungsschritt auf ein *Core Model* abgebildet, das den aktuellen Zustand des Geschäftsmodells einfriert und alle im Geschäftsmodell vorhandenen Elemente eindeutig festhält. Das *Core Model* wiederum wird in ein als *UI Extension Model* bezeichnetes UML-Modell überführt, das als Grundlage zur Entwicklung der Benutzerschnittstelle dient. Änderungen zwischen *Core* und *UI Extension Model* werden durch die Werkzeugunterstützung immer dann überprüft, wenn der Entwickler eine der vier angebotenen Perspektiven auf das *UI Extension Model* öffnet. Abbildung 23 zeigt die Modelle und Perspektiven auf das *UI Extension Model*, die durch das als MDHI (*Model-Driven Human Interaction*) bezeichnete, prototypisch implementierte Werkzeug auf Basis von Eclipse unterstützt werden.

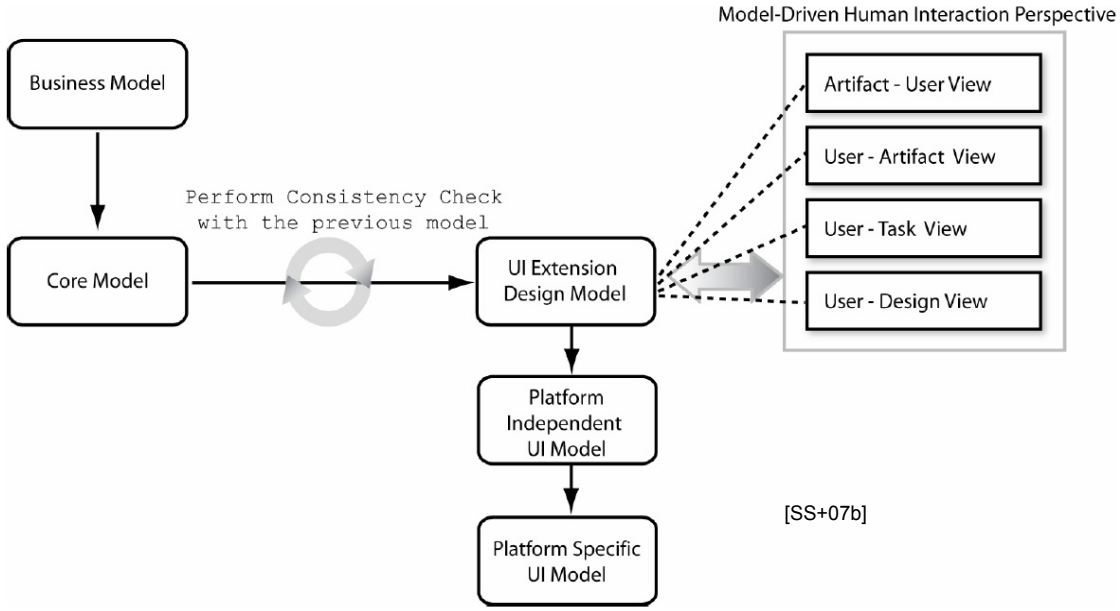

Abbildung 23: Unterstützte Modelle der Benutzerinteraktions-Entwicklung

Die als *User-Artifact View* bezeichnete Perspektive beispielsweise ordnet den einzelnen am Geschäftsprozess beteiligten Rollen die zu bearbeitenden Geschäftsobjekte zu und ermöglicht damit für jede Rolle eine genaue Spezifikation des Zugriffs auf die Geschäftsobjekte (schreibend und/oder lesend). Je nachdem ob ein Geschäftsobjekt nur innerhalb einer Aufgabe durch einen einzelnen Benutzer oder über mehrere Aufgaben hinweg durch mehrere Benutzer bearbeitet werden muss, unterscheidet der Ansatz zwischen kontextuellen Artefakten (*Contextual Artifacts*) und Geschäftsinformationsartefakten (*Business Info Artifacts*). In einer zweiten als *User-Design View* bezeichneten Perspektive des MDHI-Werkzeugs werden diejenigen Attribute, die für die Benutzerschnittstelle von Belang sind, zusätzlich als Benutzerschnittstellenattribute (*Human Interface Attribute*) modelliert und in unterschiedlichen Datengruppen zusammengefasst. Die einzelnen Datengruppen wiederum werden auf einer Seite, die aus mehreren Fragmenten und Elementen der Benutzerschnittstelle bestehen kann, dem Benutzer präsentiert. Das dem MDHI-Werkzeug zugrunde liegende Modell basiert auf dem Metamodell der UML. Folglich werden die zusätzlichen Stereotypen im MDHI-Werkzeug durch ein als *Human Interaction Design Profile* bezeichnetes UML-Profil bereitgestellt.

Bewertung des Ansatzes

Durch das in Abbildung 23 dargestellte Vorgehensmodell werden ausgehend von einem Geschäftsmodell alle Entwicklungsaktivitäten der Benutzerinteraktion integriert betrachtet (A1.1). Obwohl die

Konsistenz der einzelnen Modelle durch die Entwicklungsumgebung explizit sichergestellt wird, beantworten die Autoren die Frage nach einer nachvollziehbaren Entwicklung nicht vollständig, da insbesondere die Abbildungen zwischen den einzelnen Modellen nicht allgemein beleuchtet werden, sondern durch das Werkzeug anhand nicht näher spezifizierten Abbildungsregeln durchgeführt werden (A1.1, A1.6). Die zum Einsatz kommenden Modelle folgen den Prinzipien der modellgetriebenen Architektur und stellen daher zunächst keinen Bezug zu plattformspezifischen Details her. Eine plattformunabhängige Spezifikation der Modelle wird somit ermöglicht (A1.2). Im Hinblick auf die Spezifikation der Geschäftsanforderungen stehen durch die Wahl eines Geschäftsprozessmodells als Ausgangspunkt des modellgetriebenen Entwicklungsvorgehens die Kontroll- und Datenflussperspektive eines Workflows im Mittelpunkt des Ansatzes. Eine detaillierte Spezifikation der Ressourcenperspektive wird – außer durch die Spezifikation einzelner Benutzerrollen – durch den Ansatz jedoch nicht ermöglicht (A1.3). Die verschiedenen Aspekte einer Benutzerschnittstelle werden durch den Ansatz im sogenannten *UI Extension Model* vereint, auf welches das MDHI-Werkzeug unterschiedliche Perspektiven bereitstellt. Eine getrennte Betrachtung der Aspekte einer Benutzerschnittstelle in jeweils eigenständigen Modellen kann daher nicht erreicht werden. Wird in einer Perspektive eine Änderung an einem Modellelement durchgeführt, so wirkt sich diese Änderung folglich auch auf alle anderen Perspektiven des MDHI-Werkzeugs aus, die dieses Modellelement umfassen. Die Integration verschiedener Fachentwickler, welche die Entwicklung unterschiedlicher Aspekte der Benutzerschnittstelle unabhängig voneinander vorantreiben, ist somit nur bedingt möglich (A1.5).

Zur Spezifikation der Benutzerschnittstelle stellt das MDHI-Werkzeug verschiedene Stereotypen für die Inhalts- und Präsentationsperspektive bereit. Die Betrachtung der Navigationsperspektive wird durch die Autoren auf der Modellebene bewusst nicht berücksichtigt, sondern für eine spätere Untersuchung ausgeblendet (A1.4) [SS+07b]. Das *UI Extension Model* wird in einem nächsten Entwicklungsschritt auf eine als plattformunabhängiges Benutzerschnittstellenmodell bezeichnete XML-Repräsentation der Benutzerschnittstelle abgebildet, die dann in unterschiedliche plattformspezifische Modelle wie beispielsweise HTML überführt werden kann. Die hierzu verwendeten automatisierten Abbildungen zwischen den einzelnen Modellen sind rein werkzeugspezifisch implementiert und eignen sich daher nicht für einen allgemeinen Einsatz in einem modellgetriebenen Entwicklungsvorgehen (A1.6). Hinsichtlich einer Umsetzung des Aspektes des Kontrollflusses der erfassten Geschäftsprozesse zeigen die Autoren zwar auf, dass eine Verknüpfung der entwickelten Modelle mit unterschiedlichen Schnittstellentechnologien wie beispielsweise Webservices möglich ist, führen diese Aspekte jedoch nicht weiter aus, weshalb keine zusätzliche Bewertung des Ansatzes nach Kategorie A2 vorgenommen wird.

3.2.2 Forschungsansätze aus dem Bereich des Web Engineering

Der Forschungsbereich des *Web Engineering* befasst sich intensiv mit der Entwicklung qualitativ hochwertiger Anwendungen für das World Wide Web auf Basis der Prinzipien der Softwaretechnik [MD+01, BC+03]. Hierbei wird der Entwicklung einer adäquaten Benutzerschnittstelle für den Menschen ein zentraler Stellenwert beigemessen, weshalb die vorgestellten Forschungsansätze eine hohe Relevanz für die Zielsetzung dieser Arbeit aufweisen.

3.2.2.1 Web Modeling Language (WebML)

Die Web Modeling Language (WebML) stellt einen Ansatz zur plattformunabhängigen Spezifikation von Webanwendungen dar [CF+02, CF+00] zu dessen Umsetzung WebML ein eigenes, iteratives Entwicklungsvorgehen bereitstellt. Der in einer ersten Phase dieses Entwicklungsvorgehens durchge-

führten Anforderungsanalyse folgt in einer zweiten als Datenentwurf bezeichneten Phase die Entwicklung eines durch WebML als Strukturmodell[1] (engl. *Structural Model*) bezeichneten Domänenmodells, das alle Entitäten der Webanwendung umfasst und durch einen als Datenexperten bezeichneten Fachentwickler erstellt wird. Das Domänenmodell bildet die Grundlage für den im nächsten Entwicklungsschritt durchgeführten Entwurf des Hypertextes. Diese Phase unterteilt WebML in die beiden Unterphasen *Hypertext Design in the large* und *Hypertext Design in the small* [CF+00]. In der ersten Unterphase entwickelt der als *Web Application Architect* bezeichnete Fachentwickler ein Hypertextmodell, das zusammengesetzt aus einem Kompositions- und Navigationsmodell die statische und dynamische Struktur der gesamten Webanwendung spezifiziert. In der zweiten Unterphase wird das Hypertextmodell durch den gleichen Fachentwickler verfeinert und für jede einzelne Seite der Webanwendung ausgeprägt. Während des Hypertextentwurfs kommen die von WebML definierten Inhaltseinheiten zum Einsatz, die den Zugriff auf das darunterliegende Domänenmodell festlegen. Mit einer Dateneinheit beispielsweise kann modelliert werden, dass die Daten einer einzelnen Entität publiziert werden sollen, eine Indexeinheit hingegen verweist auf eine Auflistung mehrerer Entitäten. Die Darstellung mehrerer Inhaltseinheiten in einer Folge oder als eine Gruppe geschieht dann durch die Seiten, auf denen die einzelnen Inhaltseinheiten miteinander verbunden werden. Eben diese Seiten werden in der zweiten Unterphase des Hypertextentwurfs verfeinert. Ein weiteres Ergebnis des Hypertextentwurfs ist das Navigationsmodell, das die dynamischen Beziehungen zwischen Seiten und Inhaltseinheiten herstellt. WebML unterscheidet hier zwei Arten von Verweisen, die zur Modellierung dynamischer Bezüge verwendet werden können. *Contextual Links* verbinden je zwei Einheiten durch einen Kontext, der über diesen Bezug transportiert wird. Als Kontext bezeichnet WebML hier diejenigen Objekte, die durch die Ausgangseinheit angezeigt und zur Zieleinheit übermittelt werden sollen. *Non-Contextual Links* verbinden analog zwei Einheiten ohne einen Kontextbezug in freier Form miteinander. Abbildung 24 zeigt ein Beispiel für ein mögliches Ergebnis des Hypertextentwurfs in Form eines Kompositions- und Navigationsmodells.

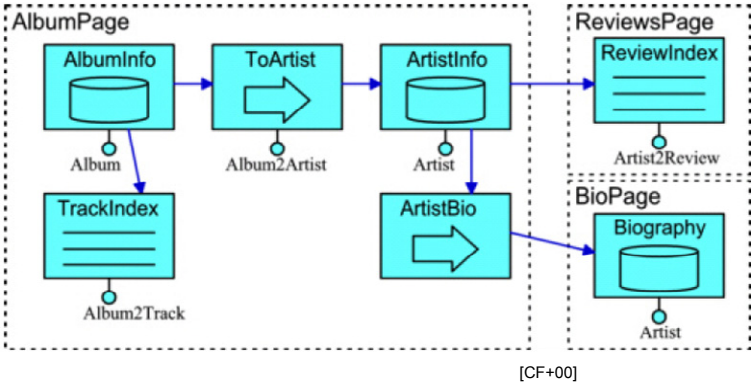

Abbildung 24: WebML Kompositions- und Navigationsmodellierung

Sobald alle im Hypertextentwurf entwickelten Modelle hinreichend stabil sind, tritt in der darauf folgenden Phase des Präsentationsentwurfs der als *Web Style Architect* bezeichnete Fachentwickler in das Entwicklungsvorgehen ein und erstellt für jede Seite eine Präsentationsvorlage, die das eigentliche Layout der Seite festlegt. Durch die Entkopplung des Domänenmodells und des Hypertextmodells ermöglicht WebML die Spezifikation mehrerer plattformunabhängiger Hypertextmodelle für je ein

[1] In späteren Versionen bezeichnet WebML das Strukturmodell analog zur in Abschnitt 2.3.2 wiedergegebenen allgemeinen Auffassung der Literatur als Daten- bzw. Domänenmodell (vgl. [Br06]).

Domänenmodell. Auf diese Weise können unterschiedliche Sichten auf ein Domänenmodell erzeugt werden, die für das jeweilige Anwendungsszenario adäquat sind.

In [BC+03] und [BC+06] erfolgt die Erweiterung des WebML-Ansatzes um den Aspekt der Workflow-Modellierung. Hierzu nimmt WebML neben den bisher betrachteten Kernaspekten der Daten und des Hypertextes zusätzlich den Aspekt des Prozesses mit in die Spezifikationsmöglichkeiten auf. Entsprechend fokussiert WebML auf konzeptioneller Ebene nun sowohl ein Daten- und Hypertextmodell als auch ein Prozessmodell. Der zusätzliche Prozessaspekt kann nach [BC+03] durch entsprechende Erweiterungen des Daten- und Hypertextaspektes berücksichtigt werden. So wird beispielsweise das Datenmodell von WebML um zusätzliche Benutzer- und Workflow-bezogene Entitäten wie *Process* oder *Activity Instance* erweitert. Im Hypertextmodell werden zusätzliche Prozesskomponenten eingeführt, welche die vorhandenen Seiten im Sinne eines Kontrollflusses miteinander verbinden. In [Br06] wird schließlich die Abbildung von Prozessbeschreibungen, die in der *Business Process Modeling Notation* (BPMN) [OASIS-BPMN] verfasst sind, auf WebML vorgestellt. Hierzu stellen die Autoren einen prototypisch implementierten Editor vor, der die Modellierung einfacher Workflows in BPMN ermöglicht. Im Prototyp spezifizierte Workflow-Modelle werden durch eine musterbasierte Abbildung in die WebML-Syntax überführt, die wiederum durch die WebML-Entwicklungsumgebung WebRatio interpretiert und ausgeführt werden kann. In WebRatio können durch die in [BC+06] vorgestellten *Web Service Units* Webservice-Aufrufe modelliert und WebML-Implementierungen in bestehende Softwarearchitekturen integriert werden. Um den bis dahin vorhandenen immanenten Bezug von WebML zur eigenen Entwicklungsumgebung WebRatio und zu Java zu lösen, präsentiert [MF+07] ein UML-Profil für WebML, um so eine breitere Anwendung des WebML-Ansatzes zu ermöglichen und dessen Konformität zu modellgetriebenen Entwicklungsansätzen herzustellen.

Bewertung des Ansatzes

Das von WebML vorgeschlagene Entwicklungsvorgehen ist klar strukturiert und verankert alle notwendigen Entwicklungsschritte nachvollziehbar in einem gemeinsamen Vorgehensmodell. Das vorgestellte Metamodell für WebML ermöglicht ferner eine Integration der WebML-Modelle in existierende Softwareentwicklungsprozesse, wodurch ein breiter Einsatz von WebML ermöglicht wird (A1.1). Die mit WebML spezifizierbaren Anforderungen können zusätzlich unabhängig von plattformspezifischen Details modelliert werden, weswegen eine Plattformunabhängigkeit der Modelle insbesondere auch durch das in [MF+07] zur Verfügung gestellte UML-Profil für WebML gegeben ist (A1.2). Durch die kontinuierliche Weiterentwicklung von WebML ist auch eine geschäftsprozessorientierte Herangehensweise ermöglicht worden, die in neueren Versionen von WebML am Beginn des Entwicklungsvorgehens steht. Die Abbildung eines Geschäftsprozesses auf Workflows und die WebML-Architektur wird durch WebML jedoch bewusst nicht in der vollen Komplexität unterstützt. Bedingt durch den Fokus auf leichtgewichtige Webanwendungen blendet WebML komplexe Workflows im Sinne der Unterstützung von komponierten Workflows, zeitlichen Abhängigkeiten oder auch Anforderungen der Ressourcenperspektive explizit aus [BC+03]. Zusätzlich betrachtet WebML die Datenperspektive eines Workflows nicht, wodurch die Frage nach dem Bezug des Prozessmodells zum Datenmodell unbeantwortet bleibt. Eine vollständige Betrachtung der unterschiedlichen Geschäftsanforderungen im Sinne der Workflow-Perspektiven wird somit nicht erreicht, zumal die bereitgestellte Entwicklungsumgebung WebRatio, wie [Br06] anführt, zwar die Spezifikation von Workflows durch die grafische Notation der BPMN ermöglicht, den Entwickler bei der Erstellung syntaktisch korrekter Modelle jedoch nicht unterstützt (A1.3). Das ursprünglich datengetriebene Vorgehensmodell des WebML-Ansatzes sieht für die Entwicklung der Benutzerschnittstelle zwei eigenständige Phasen vor, in denen WebML die Aspekte Inhalt und Hypertext (Struktur und Navigati-

on) der Benutzerschnittstelle betrachtet (A1.4). Bedingt durch den Fokus des Ansatzes auf hypertextbasierte Anwendungen sind die zur Modellierung des Hypertextmodells vorhandenen Elemente jedoch stark eingeschränkt und lassen nur eine sehr abstrakte Modellierung des Strukturmodells der Benutzerschnittstelle zu (A1.4). Die Aspekte Inhalt und Hypertext werden in ersten Versionen des WebML-Ansatzes in jeweils eigenständigen Modellen erfasst, wodurch die Integration einzelner Fachentwickler zunächst explizit unterstützt wird (A1.5). Durch den in neueren Versionen von WebML hinzugekommenen Prozessaspekt und dessen Berücksichtigung sowohl im Inhalts- als auch im Hypertextmodell kann durch die Vermischung der Modelle eine strikte Trennung der Zuständigkeiten jedoch nicht aufrechterhalten werden (A1.5).

Die plattformunabhängigen Modelle von WebML werden durch die Entwicklungsumgebung WebRatio auf Quellcode abgebildet, welcher auf der in [BC+06] beschriebenen Laufzeitumgebung ausgeführt werden kann. Die zur Abbildung der Modelle notwendige Komponente des Codegenerators überführt die in einer internen XML-Repräsentation erfassten WebML-Modelle durch Abbildungsregeln, die in XSL (*Extensible Stylesheet Language*) spezifiziert sind, in ausführbare Implementierungen für J2EE, Struts oder .NET. Da diese Abbildungsregeln durch den Entwickler adaptiert werden können, wird somit eine automatisierte Abbildung ermöglicht, die jedoch nicht auf den bestehenden Modellen operiert und daher nicht ohne Weiteres innerhalb eines modellgetriebenen Entwicklungsvorgehens eingesetzt werden kann (A1.6). Da WebML wie oben angeführt die Ausführung komplexer Workflows per se nicht unterstützt, wird eine entsprechende Unterstützung von der Laufzeitumgebung ebenfalls nicht erbracht (A2.1). Diese in [BC+06] näher beschriebene Architektur der Laufzeitumgebung von WebML sieht die Integration von Webservices über eigenständige Komponenten wie beispielsweise den SOAP *sender* und SOAP *listener* zwar vor, eine interne Dienstorientierung wird durch die Laufzeitumgebung jedoch nicht realisiert (A2.2). Aus diesem Grund wird eine Integration der zur Unterstützung der Benutzerinteraktion zusätzlich benötigten Fachfunktionalität in Form bedarfsgerechter fachfunktionaler Dienste nicht ermöglicht (A2.3).

3.2.2.2 UML-based Web Engineering (UWE)

In [Ko01] und [KK+01] konzipieren die Autoren mit dem als UML-based Web Engineering (UWE) bezeichneten Ansatz ein objektorientiertes Entwicklungsvorgehen für Webanwendungen auf Grundlage der UML. Hierbei bedient sich UWE bewusst bestehender Ansätze wie beispielsweise der *Relationship Management Methodology* (RMM) [IS+95] oder der *Object-Oriented Hypermedia Design Method* (OOHDM) [RS+00] und strebt an, deren Vorteile in einem gemeinsamen Ansatz zu vereinen. Als zugrunde liegendes Vorgehensmodell nutzt UWE den *Unified Software Development Process* [JB+99] und leitet aus diesem die vier zentralen Entwicklungsphasen der Anforderungsanalyse, des konzeptionellen Entwurfs, des Navigations- und des Präsentationsentwurfs ab. In der Analysephase werden die Anforderungen der zu entwickelnden Webanwendung in Form eines UML-Anwendungsfallmodells erfasst, das den zentralen Ausgangspunkt aller weiteren Entwicklungsschritte von UWE darstellt. In der zweiten Phase des konzeptionellen Entwurfs greift UWE das Anwendungsfallmodell auf und leitet aus diesem ein sogenanntes konzeptionelles Modell ab, das alle Entitäten samt deren Assoziationen und somit die Domäne der Anwendung in Form eines UML-Klassendiagramms erfasst. Die modellierten Klassen und Assoziationen werden in der folgenden Navigationsentwurfsphase zur Spezifikation von Navigationsknoten im Navigationsmodell herangezogen. Das Navigationsmodell legt fest, welche Daten des konzeptionellen Modells zur Verfügung stehen und gibt ferner an, wie dieser Zugriff erfolgen kann. Dazu wird das Navigationsmodell in ein Navigationsstruktur- und ein Navigationsraummodell unterteilt. Letzteres hält fest, welche Klassen des konzeptionellen Modells in einer Navigation verfügbar sein müssen. UWE verwendet zur Kennzeich-

nung dieser Klassen den Stereotyp *Navigational Class* und zur Spezifikation der Navigationsbeziehungen innerhalb dieser Klassen den Stereotyp *Direct Navigability*. Der Zugriff auf diese Navigationsklassen des Navigationsraummodells erfolgt dann wiederum mittels spezieller Zugriffsstrukturen, die im Navigationsstrukturmodell erfasst und als UML-Klassen visualisiert werden. Als Beispiele für Zugriffsstrukturen gibt [KK+01] den Stereotyp *Index* (Auflistung aller Navigationsklassen) oder *Guided Tour* (Durchblättern mehrerer Objekte von Navigationsklassen) an.

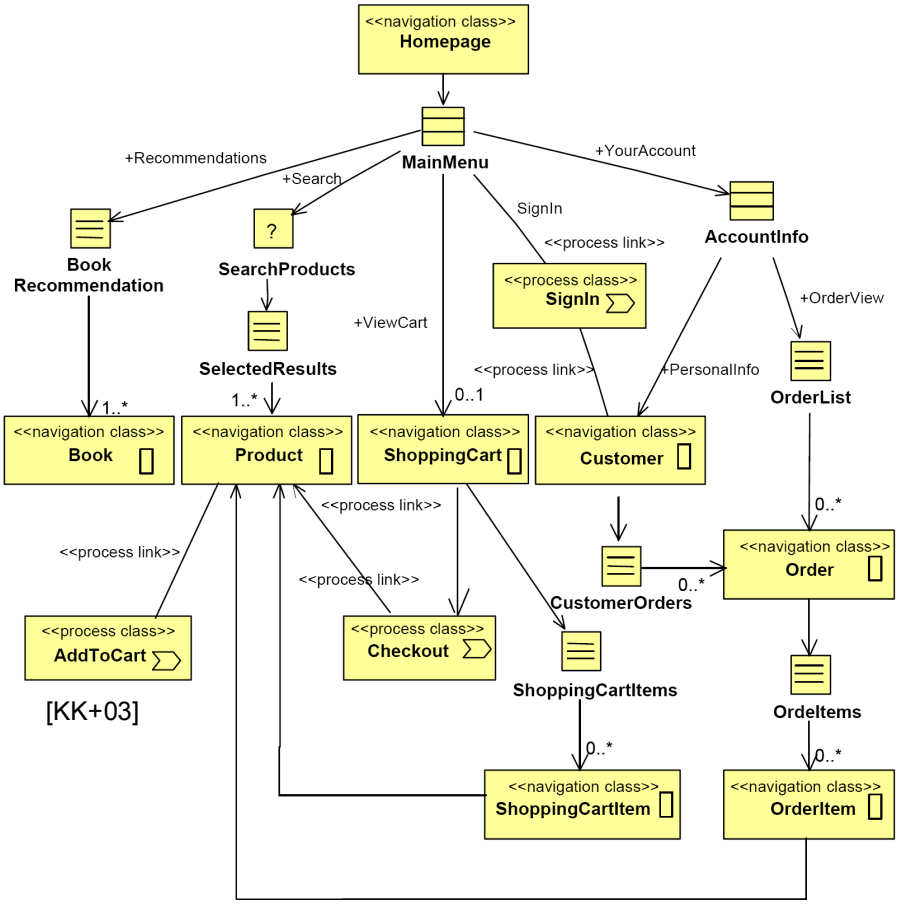

Abbildung 25: UWE – Erweitertes Navigationsmodell

Aufbauend auf dem Navigationsstrukturmodell und den in der Anforderungsanalyse gesammelten Informationen wird das Präsentationsmodell entwickelt, zu dessen Spezifikation UWE die Entwicklung von *Sketches*, *Storyboards* und Präsentationsflussmodellen vorschlägt. Unter *Sketches* versteht UWE die abstrakte Repräsentation einer Benutzerschnittstelle im Sinne einer Sicht auf einen Navigationsknoten. Zu deren Modellierung stellt UWE unterschiedliche Stereotypen wie beispielsweise ein *User Interface View* bereit, die als Container für Präsentationsklassen dienen. Präsentationsklassen wiederum dienen der Unterteilung einer Benutzerschnittstelle und können durch Benutzerschnittstellenelemente wie beispielsweise Form, *Image* oder Text befüllt werden. Wurden auf diese Weise alle abstrakten Benutzerschnittstellen zu den Navigationsknoten des Navigationsstrukturmodells erfasst, können diese optional über eine als *Storyboarding* bezeichnete Methode in Bezug zueinander gestellt werden. Letztendlich wird das Ergebnis des *Storyboarding* in ein Präsentationsflussmodell überführt, das die Aufteilung der Präsentation in Fenster und Rahmen beschreibt und damit das Strukturmodell spezifiziert.

Die Entwicklung einer konkreten Benutzerschnittstelle erfolgt nachgelagert in der Implementierungsphase. In [KK02] ist die Erweiterung von UWE um den Aspekt zur Benutzeraufgabenmodellierung zu

finden. Die Weiterentwicklung von UWE wird in [KK+03] mit der Erweiterung des Ansatzes um eine Prozessperspektive fortgesetzt. Nach wie vor steht das Anwendungsfallmodell am Beginn des Entwicklungsvorgehens, das dann jedoch in ein zweigeteiltes konzeptionelles Modell überführt wird, das neben der ursprünglichen gesamtheitlichen Sicht auf die Entitäten eine geteilte Sicht in allgemeine Entitäten und Entitäten, die direkt den Benutzer betreffen, in Form eines Benutzermodells (engl. *User Model*) ermöglicht. Im nächsten Entwicklungsschritt wird das Navigationsstrukturmodell um die Stereotypen *Process Class* und *Process Link* erweitert. Ein *Process Link* definiert den Einstiegspunkt zu einem Geschäftsprozess, der dann durch eine *Process Class* modelliert und durch ein UML-Aktivitätsdiagramm konkret spezifiziert wird. Abbildung 25 zeigt ein Beispiel für ein UWE-Navigationsmodell.

Aufgrund der durchgängigen Verwendung von UML liegt die in [KK+07a] und [Ko06] präsentierte Integration der Prinzipien der modellgetriebenen Architektur in UWE nahe. Hierbei sieht UWE die Erfassung von Anforderungen auf der Ebene eines berechnungsunabhängigen Modells durch das Anwendungsfallmodell und das konzeptionelle Modell vor und überführt diese entlang der Modellhierarchie der modellgetriebenen Architektur auf plattformunabhängige funktionale Modelle wie etwa das Navigations-, Prozess- oder Präsentationsmodell sowie ein architekturelles Modell. Alle funktionalen Modelle werden anschließend zusammen in ein einziges, sogenanntes *Big Picture*-Modell überführt und zusammen mit den Architekturmodellen auf ein *Integration Model* abgebildet, das wiederum für verschiedene Plattformen wie JEE oder .NET ausgeprägt werden kann.

Bewertung des Ansatzes

UWE nutzt den *Unified Software Development Process* als grundlegendes Vorgehensmodell, das alle Entwicklungsaktivitäten integriert (A1.1). Zusätzlich wird die Anforderung nach einer plattformunabhängigen Spezifikation der Anforderungen der Benutzerinteraktion im Rahmen der durch UWE betrachteten Aspekte insbesondere auch durch die Ausrichtung des Ansatzes an den Prinzipien der modellgetriebenen Architektur vollständig erfüllt (A1.2). Bedingt durch die Historie von UWE steht dabei immer ein Modell der zu unterstützenden Anwendungsfälle am Beginn des Vorgehens, wodurch das in späteren Versionen von UWE hinzugenommene Prozessmodell nicht in den Mittelpunkt des Ansatzes rückt. Demnach bleibt auch die Frage offen, wie die in UWE mittels UML-Aktivitätsdiagramme modellierten Geschäftsprozesse auf die IT-Unterstützung abgebildet werden. Ebenso stellt UWE zur Spezifikation der menschlichen Ressourcen, die an der Ausführung eines Geschäftsprozesses beteiligt sind, keine Konzepte zur Verfügung (A1.3). Zur Modellierung der Aspekte der Benutzerschnittstelle nutzt UWE im Vergleich zu anderen Ansätzen eine hohe Anzahl an unterschiedlichen Modelltypen, wodurch prinzipiell eine umfangreiche und detaillierte Spezifikation der Benutzerschnittstelle ermöglicht wird. Jedoch ist hierbei insbesondere das funktionale Verhalten der Benutzerschnittstelle auf die bereitgestellten Zugriffselemente wie *Index* oder *Guided Tour* beschränkt. Mit diesen Elementen wird der grundsätzliche Bedarf von Webanwendungen zwar abgedeckt, eine individuelle Anpassung an anwendungsspezifische Bedürfnisse wird von UWE jedoch nicht vorgesehen (A1.4). Die Integration von Fachentwicklern gestaltet sich für die betrachteten Aspekte von UWE durch die klar getrennten Modelle und die durchgängige Ausrichtung an einem Vorgehensmodell problemlos. Insbesondere wird auch durch den klaren Zusammenhang der einzelnen Modelle und deren Einsatz entlang des Vorgehens eine nachvollziehbare und konsistente Entwicklung ermöglicht (A1.5, A1.1). Zur Unterstützung des Entwicklungsvorgehens kommt eine als AgroUWE bezeichnete Erweiterung von AgroUML zum Einsatz, die ebenfalls zur automatisierten Abbildung der Modelle eingesetzt werden kann [KK+07a]. Wie in [KK+07b] beschrieben, kann durch den Einsatz von Metamodell-basierten Transformationen eine beliebige Zielplattform adressiert werden (A1.6).

3.2.2.3 Freudenstein et al.: Model-driven Construction of Workflow-based Web Applications with Domain-specific Languages

In [FB+07] beschreiben Freudenstein et al. einen modellgetriebenen Entwicklungsansatz für Workflow-basierte Webanwendungen. Anhand eines Geschäftsprozesses verdeutlichen die Autoren, dass bei der Umsetzung eines Geschäftsprozesses technische und organisatorische Herausforderungen zu beachten sind. Sowohl die technische Heterogenität bestehender Anwendungen und deren räumliche und organisatorische Verteilung als auch die involvierten Prozessbeteiligten und deren unterschiedliche fachliche Ausrichtung stellen Anforderungen an die Umsetzung von Geschäftsprozessen auf die vorhandene IT-Unterstützung. Aus diesen Herausforderungen leitet der Ansatz unterschiedliche Anforderungen an ein benötigtes Entwicklungsvorgehen und an die IT-Unterstützung in Form von Workflow-basierten Webanwendungen ab. Neben einer Flexibilität und Evolutionsfähigkeit des Vorgehens fordern die Autoren eine systematische Wiederverwendung einmal entwickelter Softwareartefakte, um so die Qualität der entwickelten Anwendungen steigern zu können. Eine weitere Anforderung an das Entwicklungsvorgehen liegt in der starken Einbindung der Prozessbeteiligten, um eine korrekte Abbildung der Anforderungen auf die Anwendungen frühzeitig sicherstellen zu können. Die Bereitstellung funktionaler Prototypen wird als eine geeignete Möglichkeit zur frühzeitigen Überprüfung der Anforderungen zwischen Prozessbeteiligten und Entwicklern angeführt.

Zur Umsetzung dieser Anforderungen müssen Workflow-basierte Webanwendungen nach [FB+07] neben föderativen Workflows und der multimodalen Partizipation der Benutzer auch Benutzerschnittstellen bereitstellen, die über eine hohe Dynamik verfügen und den Benutzer in der Bewältigung seiner Aufgabe geeignet unterstützen. Hierzu schlagen die Autoren ein Entwicklungsvorgehen vor, das auf dem in [NF+06a] und [NF+06b] vorgestellten Konzept der Anwendung domänenspezifischer Sprachen (engl. *Domain-Specific Language,* DSL) im *Web Engineering* beruht (vgl. Abschnitt 2.1.1). Entsprechend diesem Konzept führen die Autoren eine Workflow-DSL ein, die aus drei Kernelementen aufgebaut ist. Das domänenspezifische Modell (engl. *Domain-Specific Model*, DSM), das mithilfe der durch die WfMC standardisierten XML *Process Definition Language* (XPDL) [WfMC-XPDL2.1] spezifiziert wird, bildet die formale Grundlage aller Softwareartefakte der domänenspezifischen Sprache. Die Domäneninteraktionsmodelle (engl. *Domain Interaction Model*, DIM), unter denen der Ansatz eine konkrete (grafische) Syntax zu einem domänenspezifischen Modell versteht, stellen das zweite Kernelement einer domänenspezifischen Sprache dar. Entsprechend können unterschiedliche Domäneninteraktionsmodelle wie beispielsweise BPMN oder Petri-Netze zur Modellierung verwendet werden. Die als Lösungsbausteine (engl. *Solution Building Blocks*, SBB) bezeichneten Softwarekomponenten bilden das dritte Kernelement einer domänenspezifischen Sprache und können zur Ausführung von Softwareartefakten, die in einer domänenspezifischen Sprache spezifiziert wurden, genutzt werden.

Auf dem Konzept der DSL-basierten Entwicklung aufbauend stellt der Ansatz das in Abbildung 26 dargestellte Entwicklungsvorgehen für Workflow-basierte Webanwendungen vor. In der ersten als Geschäftsprozessmodellierung bezeichneten Phase wird der durch die Workflow-basierte Webanwendung zu realisierende Geschäftsprozess mithilfe eines geeigneten Domäneninteraktionsmodells abstrakt modelliert. Zum Fortschritt in dieser Phase tragen die Prozessbeteiligten, der Prozessanalyst und eine als Wiederverwendungsexperte (engl. *Reuse Librarian*) bezeichnete Rolle bei. Die zweite Phase der Workflow-Modellierung erweitert die Prozessmodelle um die als Zielkonfiguration (engl. *Concern Configuration*) bezeichneten ausführungsrelevanten Konfigurationsparameter. Hierbei wird jeder Aktivität des Prozessmodells ein Aktivitätsbaustein (engl. *Activity Building Block*), der über eine DSL konfiguriert werden kann, zugeordnet.

Die zweite Phase hat schließlich ein Softwareartefakt zum Ergebnis, das in der dritten Phase des physikalischen Entwurfs und der Ausführung (engl. *Physical Design & Execution*) an einen Lösungsbaustein weitergegeben wird, der einen Prototyp der Workflow-basierten Webanwendung konfiguriert. Hierbei setzen die Autoren mit dem WebComposition Service Linking System (WSLS) auf einem weiteren Forschungsergebnis auf, das in [GN+04] publiziert wurde. Die dritte und letzte Phase der Entwicklung wird durch die Rolle des *Application Configurator* begleitet, der ein Experte für das WSLS-Rahmenwerk und für die domänenspezifischen Sprachen der vorhandenen Lösungsbausteine ist. Um die Spezifikationen, die in den jeweiligen domänenspezifischen Sprachen erfasst sind, auf das domänenspezifische Modell abbilden zu können, erweitern die Autoren die im XPDL-Schema vorhandenen Anwendungstypen Form, Webservice und XSLT um je einen Untertyp, um die benötigten Konfigurationsparameter der *Activity Building Blocks* erfassen zu können.

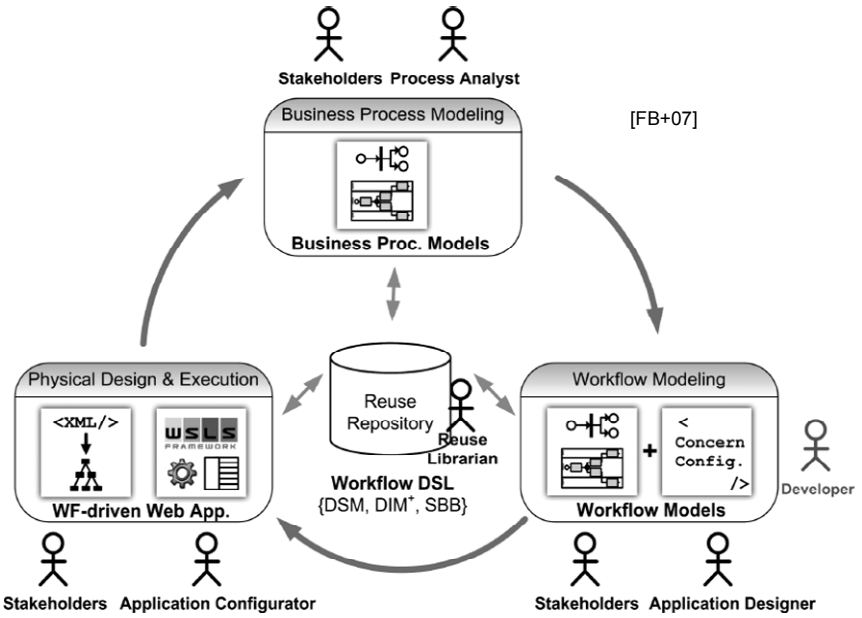

Abbildung 26: Phasen des DSL-basierten Entwicklungsansatzes

Zur Modellierung der Benutzerschnittstelle wird in [FN+08] ein eigenständiges vierstufiges Entwicklungsvorgehen eingeführt, dem eine als Dialog-DSL bezeichnete domänenspezifische Sprache zugrunde liegt. In der ersten Phase des Datenentwurfs steht das Datenmodell der zu entwickelnden Benutzerschnittstelle im Mittelpunkt. Die hier identifizierten Datenelemente werden in einer zweiten als Partitionsentwurf bezeichneten Entwicklungsphase mithilfe eines webbasierten Editors einem Struktur- und Verhaltensmodell zugeordnet. Die Repräsentation der einzelnen Elemente durch verschiedene Interaktionselemente, wie beispielsweise Ein- oder Ausgabefelder, folgt in der dritten Entwicklungsphase des Designentwurfs, die erneut durch den webbasierten Editor unterstützt wird. Eine vierte als Evolution bezeichnete Entwicklungsphase adressiert Änderungen des Datenmodells. Diese Änderungen werden durch das genutzte technische Rahmenwerk dahin gehend unterstützt, dass Änderungen am Datenmodell nur dann eine Neugenerierung der zugeordneten Interaktionselemente hervorrufen, wenn diese von der Änderung betroffen sind.

Bewertung des Ansatzes

Der vorgestellte Ansatz verfolgt zur Entwicklung Workflow-basierter Anwendungen ein klares Entwicklungsvorgehen, das alle benötigten Entwicklungsaktivitäten integriert und daher die Nachvollziehbarkeit und Konsistenz der entwickelten Softwareartefakte sicherstellt (A1.1). Im Entwicklungsvorgehen nutzt der Ansatz verschiedene domänenspezifische Sprachen wie beispielsweise eine

Workflow-DSL und eine Dialog-DSL, die wiederum unterschiedliche Domäneninteraktionsmodelle umfassen und eine Abstraktion von technischen Details der Laufzeitumgebung ermöglichen. Die Forderung nach einer plattformunabhängigen Spezifikation ist somit vollständig erfüllt (A1.2). Bei der Umsetzung der Geschäftsanforderungen fokussieren die Autoren insbesondere die Kontrollfluss- und Datenperspektive eines Workflows, indem unterschiedliche Spezifikationssprachen wie beispielsweise BPMN oder Petri-Netze zur Erfassung von Workflows herangezogen werden. Eine detaillierte Betrachtung der Ressourcenperspektive wird durch die Autoren nicht weiter verfolgt. Die Erweiterung der Ausdrucksmächtigkeit dieser Spezifikationssprachen durch geeignete Metamodellerweiterungen steht ebenfalls nicht im Fokus des Ansatzes (A1.3). Die Aspekte einer Benutzerschnittstelle werden durch die Dialog-DSL auf Basis von XForms und Petri-Netzen modelliert, wobei die für diese Arbeit wesentlichen Aspekte Inhalt, Präsentation und Navigation vollständig berücksichtigt und die Entwicklung der benötigten Modelle zusätzlich durch einen eigenständigen webbasierten Editor unterstützt werden. Der Einsatz und der Bezug der neuen Modellelemente wird durch die Autoren durch ein Klassendiagramm aufgezeigt. Die Frage nach der für diese Arbeit wichtigen Integration der entworfenen Modellelemente in ein bestehendes Metamodell und der daraus resultierenden Unterstützung des Entwicklers bei der Erstellung syntaktisch korrekter Modelle bleibt durch die Autoren jedoch unbeantwortet (A1.4). Die Integration von Fachentwicklern nach dem Prinzip der getrennten Zuständigkeiten wird durch das verwendete Entwicklungsvorgehen explizit adressiert und durch die verwendeten Modelle unterstützt (A1.5). Zur Abbildung der Modelle der Benutzerschnittstelle nutzen die Autoren ein mehrstufiges Verfahren, das ein Muster des domänenspezifischen Modells in unterschiedliche Auszeichnungssprachen wie beispielsweise XForms überführen kann und somit eine automatisierte Abbildung ermöglicht. Eine Umsetzung der Abbildungen durch Metamodell-basierte Transformationen, die in ihrer Ausführung direkt auf den Modellen operieren, wird durch das präsentierte Vorgehen nicht ermöglicht (A1.6).

3.3 Dienstorientierte Architektur zur Unterstützung der Benutzerinteraktion

Zur IT-gestützten Ausführung von Geschäftsprozessen mit Benutzerinteraktion wird eine dienstorientierte Architektur benötigt, die einerseits eine Benutzerschnittstelle zur Verfügung stellt und andererseits die Ausführung von Benutzeraufgaben im Kontext eines Workflows unterstützt. Gleichzeitig muss die Interoperabilität zwischen allen hierzu benötigten fachfunktionalen Komponenten gegeben sein, um eine flexible IT-Unterstützung der Geschäftsprozesse gewährleisten zu können. Es muss daher untersucht werden, welche Forschungsansätze für derartige Erweiterungen bestehender dienstorientierter Referenzarchitekturen vorhanden sind. Dementsprechend werden die im Folgenden diskutierten Forschungsansätze nach Kategorie A2 des Anforderungskatalogs bewertet.

3.3.1 Thomas et al.: User Tasks and Access Control over Web Services

In [TP+07] stellen die Autoren einen Ansatz zur Unterstützung von Benutzerinteraktion in dienstorientierten Architekturen vor, der entsprechend dem in Abbildung 27 gezeigten schematischen Aufbau konzipiert ist. Die dargestellte BPEL-*Engine* führt Webservice-Orchestrierungen aus, die durch standardkonformen BPEL-Code spezifiziert sind. Sie bedarf demnach keiner Anpassung hinsichtlich der Unterstützung von Benutzeraufgaben, worin ein wesentlicher Vorteil dieses Ansatzes liegt. Als neue zentrale Komponente zur Unterstützung der Benutzerinteraktion wird der sogenannte *People Activity Manager* eingeführt. Er wird als Mediator für Interaktionen zwischen Geschäftsprozessen und dem Menschen konzipiert und kann daher von der BPEL-*Engine* über eine WSDL-Schnittstelle

[W3C-WSDL1.1] aufgerufen werden. Gleichzeitig stellt die Komponente eine weitere WSDL-Schnittstelle bereit, die von einer Benutzerschnittstelle genutzt werden kann. Der *People Activity Manager* hält Definitionen sowie laufende und abgeschlossene Instanzen von Benutzeraufgaben vor. Über die Benutzerschnittstelle kann der Benutzer die Funktionalität des *People Activity Manager* nutzen. Hierzu stellt der *People Activity Manager* eine WSDL-Schnittstelle bereit, welche die Abfrage und Steuerung von Instanzen der Benutzeraufgaben gestattet. Ferner ist der *People Activity Manager* zusätzlich an ein System des Identitätsmanagements angebunden. Die Integration eines Identitätsmanagementsystems ist notwendig, da die zur Bearbeitung einer Benutzeraufgabe berechtigten Menschen häufig in Form von abstrakten Rollen definiert werden (vgl. dazu Abschnitt 2.2.1). Diese indirekten Benutzerbeschreibungen müssen in konkrete Benutzernamen aufgelöst werden (Abbildung 27: *resolve people attributes*), um anschließend den Benutzer authentifizieren und entsprechend seiner Rechte autorisieren zu können. Hierzu ist das Identitätsmanagementsystem an ein Benutzerverzeichnis angebunden. Der *People Activity Manager* kann somit über das Identitätsmanagementsystem die Berechtigungen der Benutzer zur Bearbeitung einer Benutzeraufgabeninstanz überprüfen.

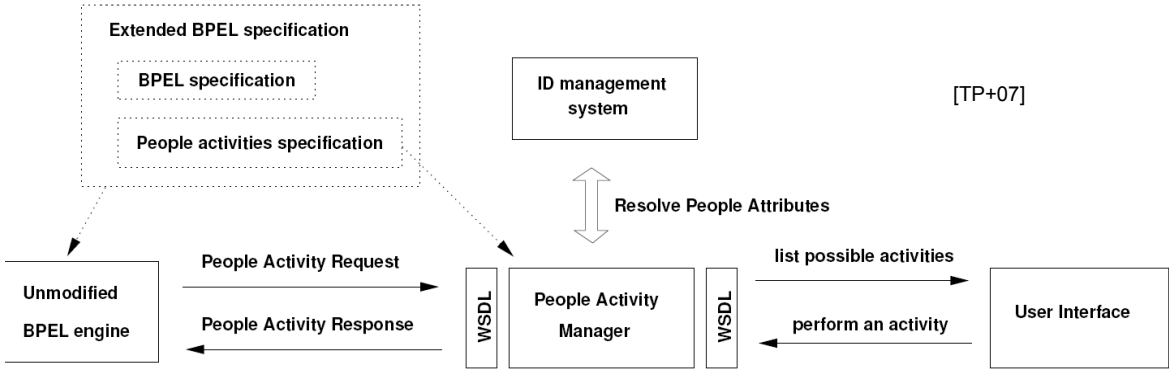

Abbildung 27: Erweiterte dienstorientierte Architektur

Das dynamische Verhalten der einzelnen Komponenten untereinander wird durch die Autoren dahin gehend beschrieben, dass zur Erzeugung einer neuen Benutzeraufgabeninstanz der *People Activity Manager* von einer BPEL-Orchestrierung über eine Operation *newActivity* asynchron aufgerufen wird. Die Ausführung des Workflows wird bis zur Antwort des *People Activity Manager* unterbrochen. Die weitere Verwaltung der Benutzeraufgabeninstanz erfolgt nun im *People Activity Manager*. Der Benutzer kann über die Benutzerschnittstelle die Operationen *listActivites*, *claimActivity*, *completeActivity* und *revokeActivity* aufrufen (vgl. dazu Abschnitt 2.2.2.4). Nach der Bearbeitung der Benutzeraufgabeninstanz durch einen Benutzer ruft der *People Activity Manager* die BPEL-*Engine* über einen Rückruf auf, die dann mit der Abarbeitung des Workflows fortfährt. Während der Ausführung einer Benutzeraufgabe kann die BPEL-*Engine* eine bereits instanziierte Benutzeraufgabe durch Aufruf der Operation *cancelActivity* des *People Activity Manager* für ungültig erklären.

Bewertung des Ansatzes

Mit ihrem Ansatz fokussieren die Autoren eine Unterstützung von Workflows mit Benutzerinteraktion in dienstorientierten Architekturen. Neben der eigentlichen Benutzerschnittstelle und der zur Ausführung von Workflows verwendeten BPEL-*Engine* konzipieren die Autoren mit dem *People Activity Manager* die im betrachteten Szenario dieser Arbeit (vgl. Abbildung 1) benötigte dritte Komponente der Benutzeraufgabenverwaltung zur vollständigen Unterstützung von Benutzerinteraktion (A2.1). Obwohl die Integration aller Komponenten prinzipiell über Webservice-Schnittstellen erfolgt, lassen die Autoren die Frage nach der konkreten Verknüpfung zu einer Benutzerschnittstelle weitestgehend unbeantwortet und deuten diese lediglich an (A2.2). Aufgrund der Mächtigkeit des *People Activity*

Manager, der unter anderem auch einen zentralen *Policy Decision Point* enthält (vgl. dazu [Em08]), ist die Anforderung nach der Integrationsfähigkeit der konzipierten Architektur ebenso nur bedingt gegeben (A2.3).

3.3.2 Rodriguez et al.: Exploring Human Workflow Architectures

In [RM06] skizzieren die Autoren ähnlich zu [TP+07] den schematischen Aufbau einer *Human Workflow Architecture*. Unter dem Begriff *Human Workflow Services* führen die Autoren vier Dienste zusammen, welche die Ausführung von Workflows mit Benutzerinteraktion unterstützen. Für diese Unterstützung stellen die *Human Workflow Services* eine Schnittstelle zu Managementsystemen für Geschäftsprozesse (*BPM Systems*) und eine weitere Schnittstelle zu *Client*-Applikationen (*Client Applications*) bereit. Die Autoren sehen weiter ein externes Benutzerverzeichnis und ein sogenanntes *Tracking Repository* vor. Letzteres dient der Persistenzhaltung und speichert die Zustände der laufenden Benutzeraufgabeninstanzen. Abbildung 28 zeigt den schematischen Aufbau der beschriebenen *Human Workflow Architecture*.

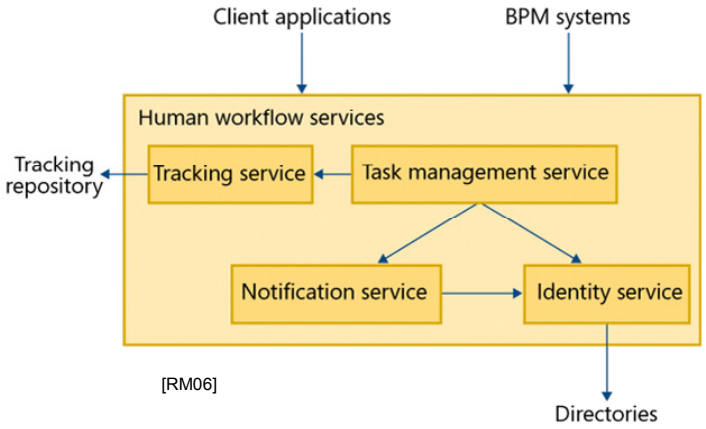

Abbildung 28: Schematischer Aufbau der Human Workflow Architecture

Zentraler Dienst der *Human Workflow Services* ist der *Task Management Service*. Dieser nimmt Anfragen des *BPM Systems* zur Erzeugung von Benutzeraufgabeninstanzen sowie Anfragen der Client-Applikation entgegen und liefert Daten oder ändert Zustände der Benutzeraufgabeninstanzen. Die Zustände der laufenden Benutzeraufgabeninstanzen werden durch ein *Tracking Service* gespeichert, der das *Tracking Repository* als externe Persistenzhaltung nutzt. Zur Benachrichtigung von Benutzern über die Erzeugung oder Zustandsänderung einer Benutzeraufgabeninstanz nutzt der *Task Management Service* einen Benachrichtigungsdienst (*Notification Service)*. Der *Task Management Service* und der *Notification Service* wiederum nutzen zur Auflösung von Benutzergruppen sowie zur Authentifizierung und Autorisierung der Benutzer einen vierten als *Identity Service* bezeichneten Dienst. Hierzu fragt der *Identity Service* ein angebundenes Benutzerverzeichnis über eine entsprechende Abfragesprache für Organisationsverzeichnisse wie beispielsweise das *Lightweight Directory Access Protocol* (LDAP) ab [RFC-2251].

Bewertung des Ansatzes

Durch die Konzeption unterschiedlicher Dienste legt der Ansatz die Grundlage zu einer durchgängigen Unterstützung von Benutzerinteraktion. Trotz der Andeutung der beiden Schnittstellen zu den *Client Applications* und *BPM Systems* bleibt die Frage nach der Art und Weise der Integration von Benutzerschnittstellen und Ausführungsumgebungen für Workflows offen (A2.1). Ferner trifft der Ansatz keine Aussagen zu der verwendeten Schnittstellentechnologie, wodurch deren Interoperabilität nicht zwin-

gend gegeben ist (A2.2). Durch die feingranulare Spezifikation einzelner Dienste, die je einen Aspekt der Unterstützung von Benutzerinteraktion fokussieren, zeigt der Ansatz eine flexible Erweiterung einer dienstorientierten Architektur, deren Konzepte problemlos in bestehende dienstorientierte Architekturen integriert werden können (A2.3).

3.4 Zusammenfassung und Handlungsbedarf

Das Ergebnis der Bewertung der untersuchten Ansätze wird in Tabelle 2 und Tabelle 3 zusammengefasst. Für alle Anforderungen bedeutet ein „+“ in der Tabelle, dass die Anforderung vollständig erfüllt wird. Ein „o“ zeigt eine nicht vollständige oder nur teilweise Erfüllung einer Anforderung an. Wird eine Anforderung durch einen Forschungsansatz nicht erfüllt, so wird diese mit „-“ bewertet. Eine grau schattierte Zelle weist darauf hin, dass eine Anforderung durch einen Forschungsansatz nicht oder nur sehr entfernt betrachtet wird und daher nicht fundiert bewertet werden kann.

		Softwaretechnik						Web Eng.		
		Almendros et al.	Pinheiro et al.	Lorenz	Scogins et al.	Nunes et al.	Sukaviriya et al.	WebML	UWE	Freudenstein et al.
A1	1.1 Integration in gemeinsames Vorgehensmodell		o	+	+	o	o	+	+	+
	1.2 Plattformunabhängigkeit	-	+	+	+	+	+	+	+	+
	1.3 Spezifikation der Geschäftsanforderungen			-			o	-	-	o
	1.4 Aspekte einer Benutzerschnittstelle	o	o	+	o	+	o	o	o	o
	1.5 Integration der Fachentwickler	+	+	-	-	o	o	o	+	+
	1.6 Automatisierte Ausführung der Abbildung			-			-	-	+	o

Tabelle 2: Ergebnis der Bewertung der Forschungsansätze nach Kategorie A1

Eine Betrachtung des in Tabelle 2 dargestellten Ergebnisses der Bewertung zeigt, dass keiner der in Abschnitt 3.2 betrachteten Forschungsansätze in der Lage ist, die Anforderungen der Kategorie A1 zufriedenstellend zu erfüllen. Defizite verwandter Forschungsansätze bestehen insbesondere bei einer durchgängigen Spezifikation der Geschäftsanforderungen, die sich auf die verschiedenen Perspektiven eines Workflows auswirken. So betrachten die untersuchten Forschungsansätze zwar häufig die Perspektive des Kontrollflusses und teilweise auch die Perspektive der Daten, die für die Benutzerinteraktion wichtigen Perspektiven der Ressourcen und der Ausnahmebehandlung findet jedoch keine ausreichende Berücksichtigung. Eine umfassende Unterstützung von Workflows mit Benutzerinteraktion kann somit nur durch einen hohen manuellen Entwicklungs- und Konfigurationsaufwand erreicht werden. Dieser Aufwand steht einer flexiblen IT-Unterstützung im Wege. Ferner ist zu bemerken, dass nur wenige Forschungsansätze die Abbildung der konzipierten Modelle bis auf Quellcode verfolgen, sodass eine durchgängige Umsetzbarkeit der präsentierten Ansätze nicht beurteilt werden kann.

Das Ergebnis der Bewertung der in Abschnitt 3.3 untersuchten Forschungsansätze zur Erweiterung einer dienstorientierten Architektur zeigt, dass kein Ansatz die Anforderungen der Kategorie A2

vollständig erfüllt. Obwohl alle Ansätze die Erweiterung einer dienstorientierten Architektur zur Unterstützung von Benutzerinteraktion als notwendig erachten, werden diese insbesondere der Anforderung nach interoperablen Dienstschnittstellen, die für eine flexible IT-Unterstützung essenziell sind, nicht gerecht.

		SOA		
		WebML	Thomas et al.	Rodriguez et al.
A2	2.1 Unterstützung von Benutzerinteraktion	o	+	o
	2.2 Interoperable Dienstschnittstellen	-	o	o
	2.3 Integrationsfähigkeit und Flexibilität	-	o	+

Tabelle 3: Ergebnis der Bewertung der Forschungsansätze nach Kategorie A2

Da kein Forschungsansatz alle Anforderungen der Kategorien A1 und A2 zu erfüllen vermag, ist es das Ziel dieser Arbeit, einen modellgetriebenen Ansatz zu entwickeln, der den gestellten Anforderungen gerecht wird. Dieser Ansatz muss demnach auf einem klar definierten Vorgehensmodell zur Softwareentwicklung beruhen, das alle zusätzlichen Entwicklungsaktivitäten zur Berücksichtigung der Benutzerinteraktion integriert. Gleichzeitig muss der Ansatz durch die Anwendung der Prinzipien der modellgetriebenen Softwareentwicklung der Heterogenität heutiger IT-Landschaften begegnen und Modelle zur Spezifikation der Benutzerinteraktion bereitstellen, die von den technischen Details der Zielplattformen abstrahieren. Die untersuchten Forschungsansätze aus dem Bereich der Softwaretechnik liefern hierfür wesentliche Grundlagen, auf denen die Beiträge dieser Arbeit aufbauen. Gelingt es ferner, alle in dieser Arbeit als relevant erachteten Aspekte der Benutzerinteraktion durch Modelle zu erfassen, so kann eine durchgängige, modellgetriebene Entwicklung von Geschäftsprozessen mit Benutzerinteraktion erreicht werden. Zur Konzeption der hierzu benötigten Modelle liefern insbesondere die untersuchten Forschungsansätze des *Web Engineering* wertvolle Beiträge, die dementsprechend in dieser Arbeit aufgegriffen und weiterentwickelt werden. Die aus dem Bereich der dienstorientierten Architekturen diskutierten Forschungsansätze hingegen zeigen vielversprechende Konzepte zur Erweiterung einer dienstorientierten Architektur zur Unterstützung von Benutzerinteraktion auf, die in dieser Arbeit entsprechend den gegebenen Anforderungen angepasst und erweitert werden müssen. Dementsprechend muss die benötigte dienstorientierte Architektur im Gegensatz zu den bestehenden Ansätzen zum einen die Interaktion des Menschen in einem Geschäftsprozess vollumfänglich unterstützen und zum anderen den Anforderungen einer heute geforderten flexiblen IT-Unterstützung gerecht werden.

Im Hinblick auf diese Herausforderungen führen die folgenden Kapitel die Beiträge dieser Arbeit ein. Ziel dieser Beiträge ist die Konzeption eines Lösungsansatzes, der den zuvor erörterten Anforderungen gerecht wird und damit den Defiziten bestehender Ansätze begegnet. In Kapitel 4 stehen zunächst die zur Spezifikation der Benutzerinteraktion benötigten Metamodelle im Mittelpunkt, die schrittweise eingeführt und im Detail diskutiert werden. Kapitel 5 greift die entwickelten Metamodelle auf und zeigt, wie entlang der Modellhierarchie der modellgetriebenen Architektur eine automatische Abbildung einer Modellinstanz auf die jeweils nächste Modellinstanz durch Modell-zu-Modell-

Transformationen erreicht werden kann. Diese mehrstufigen Transformationen resultieren schließlich in plattformspezifischen Softwareartefakten, die durch die Dienste einer dienstorientierten Architektur ausgeführt werden können. Kapitel 6 führt abschließend alle Beiträge der Arbeit zusammen und zeigt die Tragfähigkeit der entwickelten Konzepte anhand einer Umsetzung in zwei konkreten Szenarien aus der Forschung und Lehre sowie aus der Industrie.

4 Metamodelle zur Spezifikation der Aspekte der Benutzerinteraktion

Gemäß der Zielsetzung dieser Arbeit soll ein Geschäftsprozess unter der Berücksichtigung der Aspekte der Benutzerinteraktion durch ein modellgetriebenes Entwicklungsvorgehen auf eine dienstorientierte Architektur abgebildet werden. Um hierbei die betrachteten Aspekte der Benutzerinteraktion in einem modellgetriebenen Entwicklungsvorgehen berücksichtigen zu können, werden geeignete Metamodelle benötigt, die den in Abschnitt 3.1.1 erfassten Anforderungen genügen müssen. Dieses Kapitel befasst sich daher mit der Konzeption dieser Metamodelle. Für jedes entwickelte konzeptionelle Metamodell wird eine beispielhafte Umsetzung beschrieben und zur Veranschaulichung die Anwendung der Metamodellelemente anhand eines allgemeingültigen Beispiels demonstriert.

4.1 Beiträge dieses Kapitels im Überblick

Dieser Abschnitt gibt zunächst einen Überblick über die betrachteten plattformunabhängigen Modelle der Benutzerinteraktion (Anforderung A1.2, Seite 52) und zeigt deren Zusammenhänge über ein integriertes Vorgehensmodell auf (Anforderung A1.1). In Anlehnung an die Konzepte der geschäftsgetriebenen Entwicklung dient ein Modell eines zu unterstützenden Geschäftsprozesses als Ausgangspunkt des vorgestellten Ansatzes. Dieses Modell wird entlang der Modellhierarchie der modellgetriebenen Architektur von einer berechnungsunabhängigen auf eine plattformunabhängige Ebene überführt und anschließend durch mehrere horizontale Transformationen auf weitere plattformunabhängige Modelle, die zur Spezifikation der verschiedenen Aspekte der Benutzerinteraktion benötigt werden (Anforderung A1.3, A1.4, A1.5), abgebildet.

Unterschiedlichste Vorgehensmodelle sehen zu Beginn eines Softwareentwicklungsprojektes eine Phase zur Analyse und Spezifikation der Anforderungen des Kunden vor. Im Kontext der geschäftsgetriebenen Entwicklung wird diese Phase als ein Teil der sogenannten Modellierungsphase aufgefasst, in der die Geschäftsziele des Kunden sowie die daraus resultierenden fachlichen Anforderungen identifiziert und in Form von Geschäftsprozessen ohne den Bezug zu einer möglichen IT-Unterstützung festgehalten werden (vgl. Abschnitt 2.2.4). Aus Sicht der modellgetriebenen Architektur sind diese Spezifikationen daher als berechnungsunabhängige Modelle (engl. *Computation Independent Model*, CIM) anzusehen, die auch dann noch Gültigkeit haben, falls keine Umsetzung auf eine IT-Unterstützung erfolgen soll. [KW+03]. Das berechnungsunabhängige Modell eines Geschäftsprozesses kann je nach Grad der Formalisierung in einem nächsten Modellierungsschritt zu großen Teilen für den Entwurf der Systemmodelle wiederverwendet werden [BJ05]. Aufgrund des bewusst fehlenden Bezugs eines berechnungsunabhängigen Modells zur IT-Unterstützung erfolgt der Übergang zu einem plattformunabhängigen Modell, das dann den Bezug zur IT herstellt, vorwiegend manuell. Da diese Arbeit eine automatisierte Abbildung der Modelle im Sinne von Transformationen fokussiert (A1.6), soll der Übergang von einem berechnungsunabhängigen **Geschäftsprozessmodell** auf ein plattformunabhäniges **Workflow-Modell** im Sinne einer manuell vorgenommenen Abbildung betrachtet werden und das Workflow-Modell als Ausgangspunkt der Entwicklung dienen. Durch den Bezug zur IT-Unterstützung blendet das Workflow-Modell manuelle Aktivitäten aus, da diese im späteren Betrieb nicht durch eine Ausführungsumgebung unterstützt werden können (vgl. Abschnitt 2.2.1). Die Abbildung des Workflow-Modells auf eine maschinenverarbeitbare Ausführungssprache wie beispielsweise BPEL wurde bereits von vielen Arbeiten wie etwa [EW+06, Ri07, BM+04] intensiv untersucht und soll daher durch diese Arbeit nicht weiter verfolgt werden. Abbildung 29 illustriert das

gesamte Entwicklungsvorgehen und hebt die Modelle heraus, deren Metamodelle im weiteren Verlauf eingeführt werden.

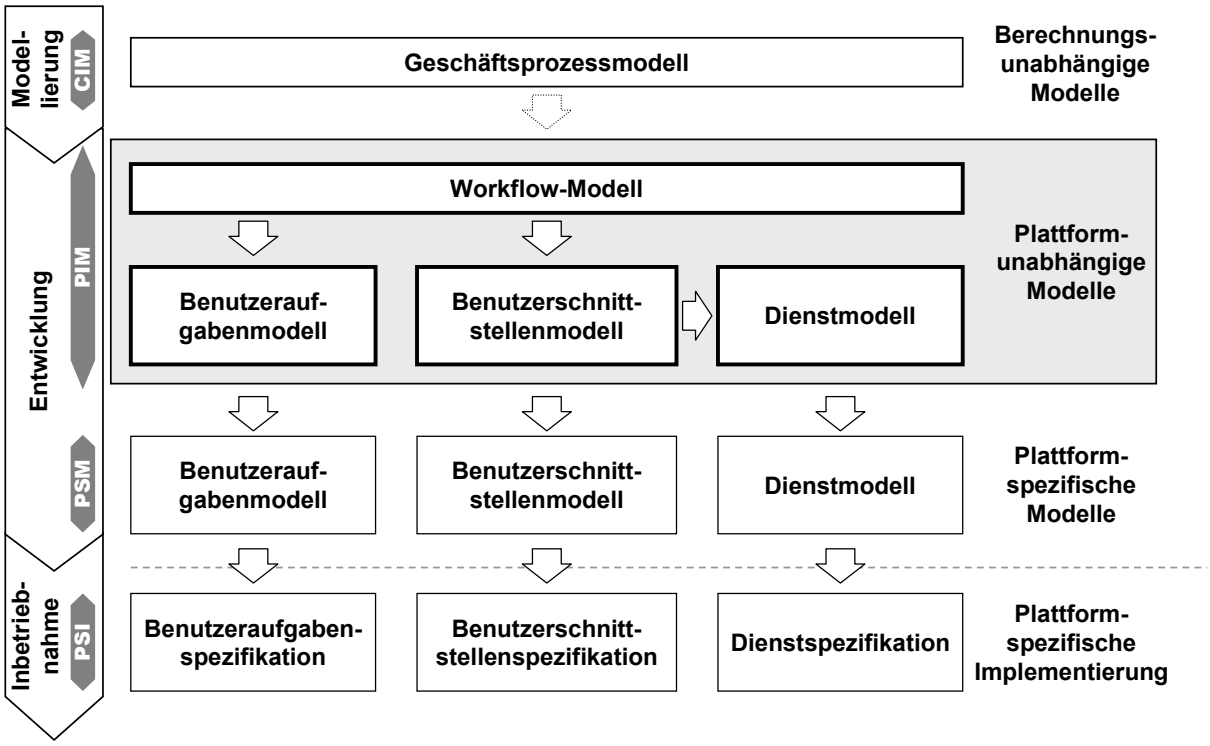

Abbildung 29: Modelle und verfolgtes Entwicklungsvorgehen im Überblick

Wie Abbildung 29 darstellt, dient das Workflow-Modell als Ausgangspunkt der Ableitung von drei Modellen, die zur Spezifikation der folgenden Aspekte der Benutzerinteraktion benötigt werden:

Das in Abschnitt 4.3 eingeführte **Benutzeraufgabenmodell** greift die durch einen Menschen in einem Workflow auszuführenden Arbeitseinheiten auf und ermöglicht deren detaillierte Spezifikation in Form von Benutzeraufgaben. Der Fokus dieses Modells liegt auf dem Menschen als Ressource, der zur Ausführung eines Workflows benötigt wird. Im Gegensatz zu technischen Ressourcen sind die Handlungen und Aktionen menschlicher Ressourcen nur sehr ungenau im Vorhinein bestimmbar und müssen daher etwa in Bezug auf die Einhaltung von Terminen und Fristen überwacht werden.

Um im Rahmen der Ausführung eines Workflows mit der IT-Unterstützung interagieren zu können, benötigt der Mensch eine geeignete Benutzerschnittstelle. Diese wird durch das in Abschnitt 4.4 beleuchtete **Benutzerschnittstellenmodell** spezifiziert. Das Benutzerschnittstellenmodell setzt sich wiederum aus unterschiedlichen Teilmodellen zusammen, die den verschiedenen Aspekten einer Benutzerschnittstelle gerecht werden (vgl. Abschnitt 2.3).

Das in Abschnitt 4.5 präsentierte **Dienstmodell** spezifiziert die im Kontext eines Workflows benötigten Dienste näher. Wie in Abschnitt 4.4 gezeigt wird, kann durch das Benutzerschnittstellenmodell eine detaillierte Spezifikation der Ein- und Ausgaben eines Benutzers in Rahmen einer Benutzeraktion vorgenommen werden. Diese Ein- und Ausgaben des Benutzers müssen von der Benutzerschnittstelle an die Benutzeraufgabenverwaltung und weiter an die Workflow-*Engine* übermittelt werden. Da beide Komponenten im Kontext dieser Arbeit als eigenständige Dienste betrachtet werden, können die zur Kommunikation zwischen diesen Diensten benötigten Dienstnachrichten und Dienstparameter aus dem Benutzerschnittstellenmodell abgeleitet werden.

Es sei an dieser Stelle darauf hingewiesen, dass die in dieser Arbeit konzipierten Metamodelle der eben angeführten Modelle jeweils auf Basis der *Meta Object Facility* (MOF) spezifiziert und mittels der durch die OMG vorgesehenen grafischen Notation dargestellt werden (siehe dazu [OMG-MOF2]). Sofern die Ausdrucksmächtigkeit der verwendeten MOF-Elemente nicht hinreichend ist, werden die einzelnen Metamodellelemente zusätzlichen Einschränkungen in Form von OCL-Ausdrücken unterworfen (vgl. Abschnitt 2.1.3.1). Um die Lesbarkeit der grafischen Darstellung der Metamodelle gewährleisten zu können, werden zusätzlich benötigte Einschränkungen nicht direkt in der grafischen Darstellung des Metamodells erfasst, sondern separat spezifiziert. Um die konzipierten Metamodelle in einem Entwicklungsvorgehen einsetzen zu können, zeigt diese Arbeit deren exemplarische Umsetzung anhand verschiedener UML-Profile auf (vgl. dazu Abschnitt 2.1.4). Zu deren grafischer Darstellung nutzt diese Arbeit ebenfalls die durch die OMG vorgesehene Notation.

Um eine bessere Einordnung der im Folgenden eingeführten Metamodelle und deren Modellelemente ermöglichen zu können, zeigt Abbildung 30 das Modell eines einfachen Geschäftsprozesses, das im weiteren Verlauf der Arbeit als durchgängiges Beispiel Verwendung findet. Die verwendete konkrete Syntax entspricht der grafischen Notation eines UML-Aktivitätsdiagramms [OMG-UML2-Super]. Für eine detaillierte Einführung der konkreten Syntax der UML sei auf die Literatur (u. a. [Ke06, Oe06, Fo04]) verwiesen.

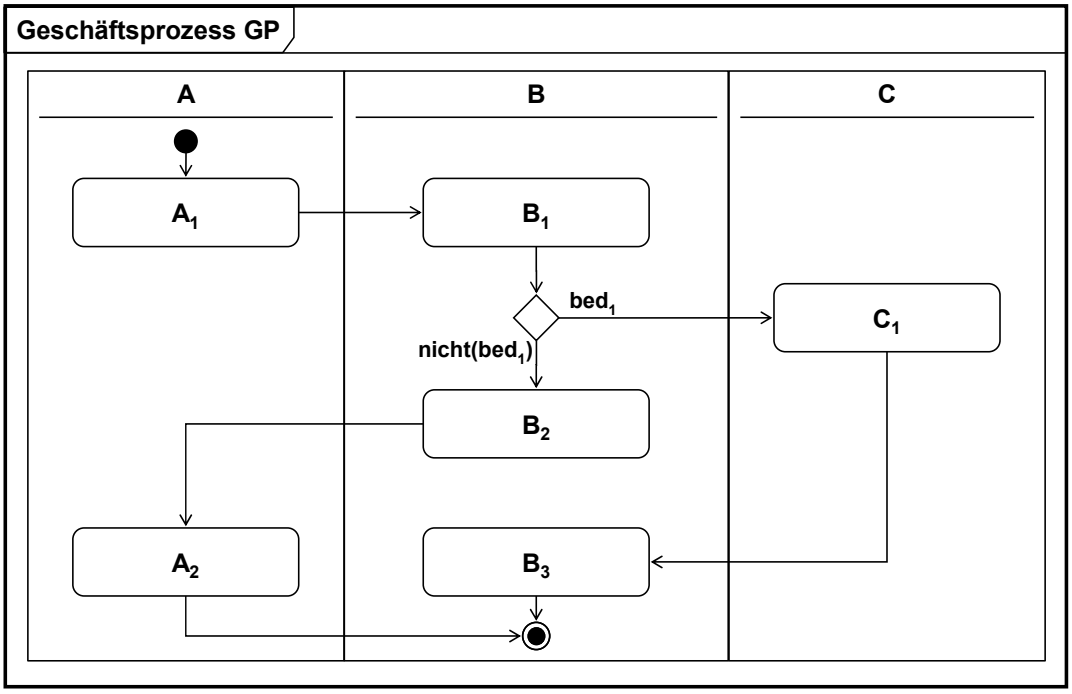

Abbildung 30: Abstraktes Beispiel eines Geschäftsprozesses

Das in Abbildung 30 gezeigte abstrakte Beispiel eines Geschäftsprozessmodells stellt keinerlei Bezug zu einer möglichen IT-Unterstützung her und kann daher als berechnungsunabhängiges Modell betrachtet werden. Damit dieses Modell auf ein plattformunabhängiges Workflow-Modell abgebildet werden kann, müssen die erfassten Aktivitäten konkretisiert und dadurch festgelegt werden, welche Aktivitäten des Geschäftsprozesses durch IT unterstützt werden können. Um diese Konkretisierung des Geschäftsprozessmodells durchführen zu können, muss das verwendete Metamodell die dazu benötigten Modellelemente zur Verfügung stellen. Abschnitt 4.2 motiviert ein entsprechendes Metamodell für Geschäftsprozesse, das auf den in diesem Kontext bestehenden Metamodellen der WfMC [WfMC-T&G3] sowie der OMG [OMG-BPDM1.0] beruht.

4.2 Integration der Benutzerinteraktion in Geschäftsprozesse

Um einen Geschäftsprozess durch IT unterstützen zu können, muss dieser zunächst auf einen Workflow abgebildet werden. Kann ein Geschäftsprozess vollständig durch IT unterstützt werden, so besteht das dazu gehörende Workflow-Modell nach der Definition der *Workflow Management Coalition* folglich nur aus automatisierten Aktivitäten, die durch ein entsprechendes Workflow-Management-System unterstützt werden können (vgl. Abschnitt 2.2.1). Können diese automatisierten Aktivitäten zusätzlich vollständig durch rein technische Ressourcen abgearbeitet werden, so kann der Workflow ohne das Zutun eines Menschen ausgeführt werden. Ein Geschäftsprozess kann jedoch, wie in Kapitel 2.2.1 erörtert, auch rein manuelle Aktivitäten sowie Aktivitäten, die der Interaktion eines Benutzers bedürfen, umfassen. Wie der Autor in [LS+07a] publiziert hat, wird eine Unterscheidung dieser verschiedenen Aktivitätstypen von bestehenden Modellierungssprachen nur bedingt unterstützt. Da die verschiedenen Aktivitätstypen für die im weiteren Verlauf dieser Arbeit konzipierten Transformationen jedoch wesentlich sind, muss deren präzise Spezifikation bereits zur Modellierungszeit eines Geschäftsprozesses durch geeignete Modellelemente ermöglicht werden.

Das durch die WfMC spezifizierte Metamodell für Geschäftsprozesse [WfMC-T&G3] sowie die Konzepte des *Business Process Definition Metamodel* (BPDM) der OMG [OMG-BPDM1.0] dienen als Grundlage des für die Konzepte dieser Arbeit adaptierten Metamodells für Geschäftsprozesse. Durch die Bestrebung, den Empfehlungen der modellgetriebenen Architektur zu folgen, wird das konzeptionelle Metamodell anschließend auf das Metamodell der UML ausgeprägt und die Anwendung der neuen Modellelemente am Beispiel demonstriert. Es sei an dieser Stelle angemerkt, dass die Spezifikationen der WfMC und OMG einige Begriffe unterschiedlich definieren. Während die OMG im Kontext der UML den Begriff „Aktion“ für eine nicht weiter zu verfeinernde Arbeitseinheit und „Aktivität“ für eine Folge von Aktionen nutzt [OMG-UML2-Super], kennt die WfMC diese Unterscheidung nicht und verwendet durchgängig den Begriff „Aktivität“ [WfMC-T&G3]. Um eine Verwirrung zwischen diesen beiden unterschiedlichen Definitionen zu vermeiden, baut diese Arbeit im Folgenden auf der UML-Nomenklatur der OMG auf und unterscheidet daher zwischen Aktivität und Aktion. Eine durch die WfMC bezeichnete Aktivität eines Geschäftsprozesses wird demnach durch eine UML-Aktion dargestellt, sofern eine weitere Verfeinerung einer Aktion aus der aktuellen Perspektive heraus nicht betrachtet wird.

4.2.1 Konzeptionelles Metamodell für Geschäftsprozesse

Die Frage, ob eine Aktion von einem System, von einem Menschen alleine oder durch eine Interaktion von Mensch und System abzuarbeiten ist, muss durch ein Modell des Geschäftsprozesses beantwortet werden können. Abbildung 31 zeigt das in dieser Arbeit verwendete Metamodell eines Geschäftsprozesses als Erweiterung der erwähnten Metamodelle der WfMC und der OMG. Die einzelnen Elemente des Metamodells werden im Folgenden detailliert eingeführt und geben die durch den Autor in [LS+07b] und [LH+09] publizierten Konzepte wieder. Da der Fokus dieser Arbeit auf den in einem Geschäftsprozess durchzuführenden Aktionen und deren Bezug zu menschlichen Akteuren liegt, blendet das konzeptionelle Metamodell bewusst die vielfältigen Spezifikationsmöglichkeiten für Kontrollflüsse zwischen mehreren Aktionen, wie beispielsweise eine Sequenz oder Parallelität, aus und greift auf die etablierten Konzepte bestehender Metamodelle zurück. Für eine detaillierte Betrachtung der verschiedenen Konzepte dieser Workflow-Perspektive sei daher auf [RH+06] verwiesen.

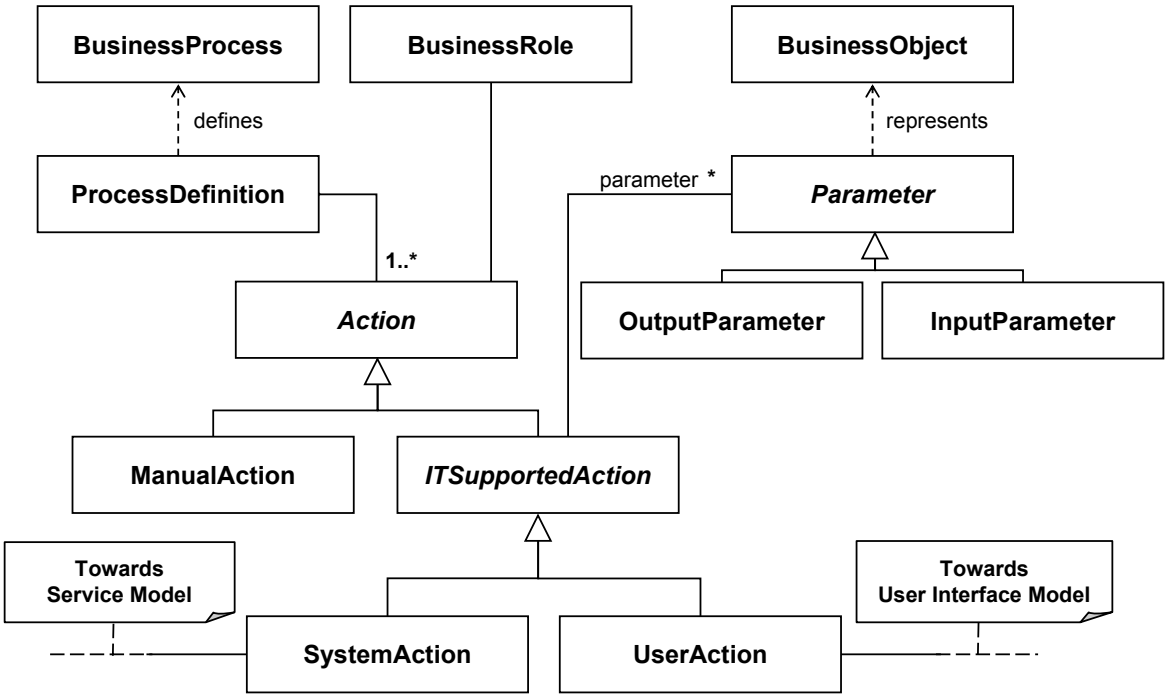

Abbildung 31: Konzeptionelles Metamodell für Geschäftsprozesse

4.2.1.1 Prozessdefinition

Eine Prozessdefinition (`ProcessDefinition`) definiert einen Geschäftsprozess und umfasst analog zu [WfMC-T&G3] mindestens eine Aktion. Weitere Aspekte einer Prozessdefinition, wie beispielsweise Kontroll- oder Datenflüsse, werden von bestehenden Metamodellen übernommen und daher im konzeptionellen Metamodell in Abbildung 31 nicht betrachtet.

4.2.1.2 Aktion

Eine Aktion (`Action`) wird durch das Metamodell zunächst abstrakt als eine Arbeitseinheit innerhalb eines Geschäftsprozesses betrachtet, zu deren Ausführung eine Ressource benötigt wird. Anhand der zur Ausführung der Arbeitseinheit benötigten Ressourcen kann eine Aktion durch die folgenden Spezialisierungen präzise spezifiziert werden.

IT-unterstützte Aktion

Kann eine Aktion durch eine technische Ressource, wie beispielsweise eine Anwendung, unterstützt werden, so stellt diese eine abstrakte IT-unterstützte Aktion (`ITSupportedAction`) dar, die durch zwei Spezialisierungen ausgeprägt wird. Wird eine Aktion aus Sicht des Menschen ohne sein Zutun vollautomatisch durch die IT-Unterstützung ausgeführt, so handelt es sich hierbei um eine **Systemaktion** (`SystemAction`). Ein Beispiel für eine Systemaktion ist die Berechnung eines Ergebnisses nach einem vorgegebenen Algorithmus. Im Kontext einer dienstorientierten Architektur wird die Arbeitseinheit einer derartigen Systemaktion durch die Fachfunktionalität eines Dienstes erbracht, weshalb Abbildung 31 an dieser Stelle die Verbindung zu dem in Abschnitt 4.5 eingeführten Dienstmodell aufzeigt.

Kann eine Aktion nicht ohne das Zutun eines Menschen ausgeführt werden, so muss der Mensch mit den IT-Systemen der IT-Unterstützung in Interaktion treten und beispielsweise benötigte Daten eingeben oder eine Entscheidung fällen. Da in dieser Interaktion der Mensch zur Ausführung einer

Arbeitseinheit die Unterstützung eines Anwendungssystems nutzt, wird diese Arbeitseinheit als **Benutzeraktion** (`UserAction`) bezeichnet. Diese Spezialisierung einer Aktion stellt auf Modellebene die Verbindung zu weiteren in dieser Arbeit entwickelten Metamodellen her, wie Abbildung 31 illustriert. Eine Benutzeraktion benötigt einerseits immer eine Benutzerschnittstelle, über die der Benutzer mit den IT-Systemen der IT-Unterstützung interagieren kann. Abschnitt 4.4 führt die zur Spezifikation einer Benutzerschnittstelle benötigten Modelle ein. Andererseits bedingt eine Benutzeraktion immer eine Benutzeraufgabe, welche in dieser Arbeit detailliert durch das in Abschnitt 4.3 eingeführte Benutzeraufgabenmodell spezifiziert wird.

Eine IT-unterstützte Aktion kann zusätzlich über Parameter verfügen. Diese Parameter legen den Zustandswechsel, der durch eine IT-unterstützte Aktion herbeigeführt wird, detailliert fest. Hierbei ist zu beachten, dass das Verhalten eines Benutzers in einer Benutzeraktion im Gegensatz zu einer Systemaktion nicht im Voraus präzise spezifiziert werden kann. Kann bei einer Systemaktion das zugrunde liegende Verhalten zum Beispiel durch einen Algorithmus verfeinert und durch ein weiteres Modell oder eine verfeinerte Sicht auf das bestehende Modell konkret spezifiziert werden, so ist dies bei einer Benutzeraktion im Sinne dieser Arbeit nicht möglich. Zu einer Benutzeraktion können lediglich der Anfangszustand und der angestrebte Endzustand durch entsprechende Ein- und Ausgabeparameter festgehalten werden, nicht aber das genaue Verhalten, das diesen Zustandsübergang bewirkt.

Sobald ein Benutzer die von ihm in einer Benutzeraktion geforderte Arbeitseinheit bewältigt hat, muss er durch eine Rückmeldung die Benutzeraktion beenden. Diese Rückmeldung durch den Benutzer kann dabei in Form einer einfachen Bestätigung aber auch in Form eines fertig ausgefüllten Formulars erfolgen. Als Beispiel sei an dieser Stelle eine Lesebestätigung eines Benutzers genannt. Bekommt der Benutzer in einem Portal eine neue Nachricht angezeigt, so kann er durch das Setzen eines Häkchens bestätigen, dass er die Nachricht gelesen hat und sie nicht erneut angezeigt bekommen möchte. Dementsprechend ändert sich durch diese einfache Rückmeldung des Benutzers der Zustand der Nachricht von „nicht gelesen“ oder „neu“ auf „gelesen“. Hieraus kann abgeleitet werden, dass eine Benutzeraktion immer eine Zustandsänderung eines Geschäftsobjekts bewirken und daher mindestens einen Ausgabeparameter haben muss. Abschnitt 4.2.1.4 geht auf diese Anforderung im Detail ein.

Der folgende OCL-Ausdruck fordert für das in Abbildung 31 dargestellte Metamodell ein, dass in der Menge der Parameter, welche einer Benutzeraktion über die gerichtete Assoziation `parameter` zugeordnet sind, die Untermenge der Parameter, die vom Typ Ausgabeparameter (`OutputParameter`) sind, immer größer null und damit nicht leer ist. Dieser OCL-Ausdruck hält somit fest, dass eine Benutzeraktion immer mindestens einen Ausgabeparameter haben muss.

```
context UserAction inv:
self.parameter -> select(oclIsTypeOf(OutputParameter)) -> size() > 0
```

Manuelle Aktion

Neben der Benutzer- und Systemaktion existieren weitere Aktionen in einem Geschäftsprozess, die durch einen Menschen ausgeführt werden müssen, ohne dass dieser dazu eine der in dieser Arbeit betrachteten grafischen Benutzerschnittstellen in Anspruch nehmen kann. Eine solche als manuelle Aktion (`ManualAction`) bezeichnete Arbeitseinheit steht zwar immer im direkten Zusammenhang mit einem Geschäftsprozess, findet sich in einem Workflow-Modell jedoch nicht wieder, da diese Arbeitseinheit durch die Ausführungsumgebung nicht überwacht werden kann. Das Führen eines Beratungsgespräches mit einem Studierenden sei als Beispiel für eine manuelle Aktion genannt.

4.2.1.3 Geschäftsrolle

Die in einem Geschäftsprozess durch Aktionen spezifizierten Arbeitseinheiten müssen durch technische oder menschliche Ressourcen erbracht werden [Aa98]. Da die konkreten technischen oder menschlichen Ressourcen, welche die Arbeitseinheiten eines Geschäftsprozesses zur Ausführungszeit übernehmen, zur Entwurfszeit eines Geschäftsprozesses in der Regel nicht festgelegt werden können, müssen diese zunächst abstrakt spezifiziert werden. Aufgrund der Tatsache, dass ein Geschäftsprozess in einem Unternehmen ausgeführt wird, stehen zu dieser abstrakten Spezifikation der technischen und menschlichen Ressourcen die in Abschnitt 2.2.1 eingeführten Ressourcenklassen „Rolle“ (Gruppierung nach Fähigkeit) und „organisatorische Einheit“ (Gruppierung nach Zugehörigkeit) zur Verfügung. Beide Ressourcenklassen stellen zur Entwurfszeit in Bezug auf einen Geschäftsprozess eine Geschäftsrolle (`BusinessRole`) dar, die zur Ausführungszeit durch eine konkrete technische oder menschliche Ressource ausgeprägt werden muss.

4.2.1.4 Parameter

Eine IT-unterstützte Aktion eines Geschäftsprozesses spezifiziert eine Arbeitseinheit, durch deren Erbringung das Anwendungssystem in einen neuen Zustand überführt wird. Muss beispielsweise eine Entscheidung getroffen werden, so ändert das Objekt der Entscheidung seinen Zustand, etwa von „nicht bewilligt“ auf „bewilligt“. Es ist daher vorteilhaft, wenn die entsprechenden Objekte, deren Zustand durch eine Aktion verändert wird, bereits in der Modellierungsphase rudimentär angegeben werden können. Durch die Modellierung von Parametern, die in einer Aktion ihren Zustand ändern, kann eine solche Spezifikation erfolgen. An dieser Stelle unterscheidet sich der Modellierungsansatz dieser Arbeit wesentlich von bestehenden Arbeiten (vgl. Abschnitt 3.2). Zum Zeitpunkt der Modellierung des Geschäftsprozesses können die beteiligten Entwicklungsrollen bereits technologieunabhängige Aussagen zu den Objekten treffen, die in einer Aktion bearbeitet werden sollen. Beispielsweise ist einem Hochschullehrer[2] (Geschäftsrolle) durchaus zur Entwurfszeit der Benutzeraktion „Prüfungstermin vereinbaren“ bewusst, dass er in dieser Benutzeraktion die Daten eines Studierenden benötigt, um mit diesem einen Prüfungstermin vereinbaren zu können. Somit kann zur Entwurfszeit spezifiziert werden, dass in der Benutzeraktion „Prüfungstermin vereinbaren“ das Objekt „Studierender“ benötigt wird. Da diese Objekte den Geschäftsprozess maßgeblich beeinflussen, werden sie im Folgenden in Anlehnung an [Oe06] als **Geschäftsobjekte** bezeichnet. Die genauen Interna dieser Geschäftsobjekte sind auf der abstrakten Modellierungsebene eines Geschäftsprozesses jedoch noch nicht von Belang. Dementsprechend kann auf der Ebene eines Geschäftsprozesses ein Parameter ein Geschäftsobjekt repräsentieren. Werden einer IT-unterstützten Aktion beispielsweise die Parameter P_1 und P_2 zugeordnet, so spezifizieren diese, dass die Geschäftsobjekte G_1 und G_2 in dieser Aktion bearbeitet werden und gegebenenfalls ihren Zustand ändern. Der Bezug zwischen Parametern, Geschäftsobjekten und IT-unterstützten Aktionen wird an einem verfeinerten Modell des als Beispiel verwendeten Geschäftsprozesses in Abbildung 34 (Seite 89) weiter verdeutlicht.

Eingabeparameter

Als Eingabeparameter (`InputParameter`) werden in dieser Arbeit analog zu [OMG-UML2-Super] Parameter verstanden, die einer Aktion zur Verfügung gestellt werden müssen, bevor deren Bearbeitung beginnen kann. Gehen Eingabeparameter in eine Systemaktion ein, so werden diese dort nach

[2] Die im weiteren Verlauf verwendeten Termini zur Bezeichnung von menschlichen Ressourcen wie etwa „Hochschullehrer“ sind geschlechtsneutral zu verstehen.

einem entsprechenden Algorithmus als Eingabeparameter verwendet. Gehen sie in eine Benutzeraktion ein, so müssen diese Parameter für den Benutzer in einer geeigneten Benutzerschnittstelle visualisiert werden.

Ausgabeparameter

Ausgabeparameter (`OutputParameter`) stehen nach der Abarbeitung einer IT-unterstützten Aktion zur Verfügung und können erneut als Eingabeparameter für eine folgende Aktion dienen. Eine Benutzeraktion muss immer über mindestens einen Ausgabeparameter verfügen. Diese Anforderung liegt in der Tatsache begründet, dass Benutzeraktionen Arbeitseinheiten spezifizieren, die durch einen Menschen erbracht werden. Dementsprechend muss durch den Menschen nach Beendigung seiner Arbeitseinheit eine Rückmeldung erfolgen, damit die zugeordnete Geschäftsprozessinstanz weiter ausgeführt werden kann. Benutzeraktionen, die ohne einen Ausgabeparameter spezifiziert und damit ohne Rückmeldung des Benutzers abgearbeitet werden können, werden in dieser Arbeit nicht betrachtet.

4.2.2 Das Benutzeraktionsprofil

Das in Abbildung 31 gezeigte konzeptionelle Metamodell kann in unterschiedliche konkrete Metamodelle überführt werden. Da sich diese Arbeit an den Empfehlungen der modellgetriebenen Architektur der OMG orientiert, wird das konzeptionelle Metamodell auf das Metamodell der UML umgesetzt. Zu dieser Umsetzung ist ein UML-Profil notwendig, da das Metamodell der UML die benötigte feingranulare Unterscheidung der einzelnen Aktionstypen per se nicht unterstützt. Abbildung 32 zeigt das im weiteren Verlauf der Arbeit als **Benutzeraktionsprofil** bezeichnete UML-Profil, das die zusätzlich benötigten Modellelemente `SystemAction`, `UserAction` und `ManualAction` entsprechend den in Abschnitt 4.2.1.2 gegebenen Definitionen als Stereotypen zur Verfügung stellt.

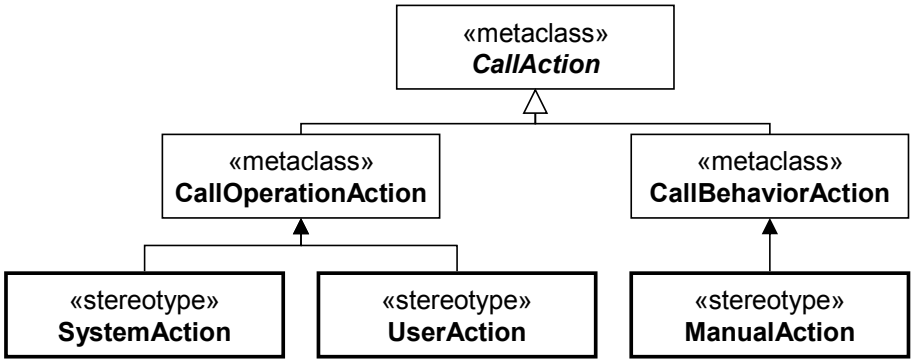

Abbildung 32: Stereotypen des Benutzeraktionsprofils

Alle weiteren Modellelemente des konzeptionellen Metamodells können ohne zusätzliche Erweiterungen auf das UML-Metamodell abgebildet werden, da dieses bereits geeignete Modellelemente mit der benötigten Semantik zur Verfügung stellt. Tabelle 4 zeigt die entsprechende Zuordnung der konzeptionellen Metamodellelemente zu den Modellelementen des UML-Metamodells auf.

Konzeptionelles Metamodell	UML-Metamodell
OutputParameter	OutputPin
InputParameter	InputPin
BusinessRole	ActivityPartition

Tabelle 4: Zuordnung der konzeptionellen Metamodellelemente

Für eine genaue Spezifikation der Semantik der verwendeten UML-Metamodellelemente sei auf [OMG-UML2-Super] verwiesen. Die zur Abbildung der Kontroll- und Datenflüsse benötigten Modellelemente werden, wie bereits erwähnt, im konzeptionellen Metamodell nicht berücksichtigt. Zu deren Erfassung bedient sich diese Arbeit daher ebenfalls der Spezifikationsmöglichkeiten der UML. Die im Benutzeraktionsprofil spezifizierten Stereotypen werden in den folgenden Abschnitten eingeführt.

4.2.2.1 Erweiterung der Metaklasse CallOperationAction

Eine **Systemaktion** als Erweiterung der UML-Metaklasse `CallOperationAction` repräsentiert eine Arbeitseinheit eines Geschäftsprozesses, die eigenständig durch ein Anwendungssystem erbracht werden kann. Die OMG definiert eine `CallOperationAction` in der UML-*Superstructure* [OMG-UML2-Super] wie folgt:

> *"CallOperationAction is an action that transmits an operation call request to the target object, where it may cause the invocation of associated behavior. The argument values of the action are available to the execution of the invoked behavior."*

Wird bei der Ausführung einer Workflow-Instanz eine Systemaktion erreicht, so ruft diese eine Operation auf einem Zielobjekt auf. Im Kontext der dienstorientierten Architekturen kann eine Systemaktion daher beispielsweise eine Operation eines Webservice aufrufen, welche die benötigte Funktionalität zur Abarbeitung der Arbeitseinheit der Systemaktion erbringt.

Eine **Benutzeraktion** ist im Benutzeraktionsprofil analog zur Systemaktion als Erweiterung der UML-Metaklasse `CallOperationAction` definiert. Dementsprechend ruft eine Benutzeraktion ebenfalls eine Operation auf einem Zielobjekt auf. Wie Abschnitt 5.3 näher ausführt, hat eine Benutzeraktion im Kontext dieser Arbeit immer die Benutzeraufgabenverwaltung als Zielobjekt (vgl. Abschnitt 2.2.2), deren Funktionalität in einer entsprechend erweiterten dienstorientierten Architektur in Form eines Webservice zur Verfügung gestellt wird. Erreicht die Workflow-Ausführungsumgebung bei der Abarbeitung einer Workflow-Instanz eine Benutzeraktion, so erzeugt diese durch einen Dienstoperationsaufruf, wie etwa *createUserTask* auf der Benutzeraufgabenverwaltung eine neue Benutzeraufgabe (vgl. Abbildung 33). Die Definition einer `CallOperationAction` sieht dabei in [OMG-UML2-Super] eine für diese Arbeit wesentliche Unterscheidung der Art des Aufrufs vor.

Wird eine `CallOperationAction` als synchron gekennzeichnet, so wartet die Workflow-Ausführungsumgebung mit der Ausführung der Workflow-Instanz, bis ein Ergebnis der aufgerufenen Dienstoperation vorliegt. Im asynchronen Fall wird nicht auf das Ergebnis gewartet, die Abarbeitung der Workflow-Instanz kann sofort weitergeführt werden. Wird demnach eine Benutzeraktion als asynchron gekennzeichnet, so kann die Workflow-Instanz sofort weiter ausgeführt werden, da die aufgerufene Dienstoperation kein Ergebnis liefert bzw. dieses Ergebnis nicht ausgewertet wird. In diesem Fall ist die im folgenden Kapitel betrachtete Überwachung einer Benutzeraktion in Form einer Benutzeraufgabe unnötig, da die Benutzeraktion beendet wird, ohne die Rückmeldung der Benutzeraufgabenverwaltung abzuwarten und somit nicht ermittelt werden kann, wann die Bearbeitung durch den Benutzer abgeschlossen ist. Wird eine Benutzeraktion hingegen als synchron gekennzeichnet, so muss die Ausführung der zugeordneten Benutzeraufgabe unter anderem hinsichtlich zeitlicher Restriktionen überwacht werden, da die Workflow-Instanz auf das Ergebnis der durch die Benutzeraktion aufgerufenen Operation wartet. Abbildung 33 illustriert diesen wesentlichen Unterschied anhand zweier einfacher UML-Sequenzdiagramme.

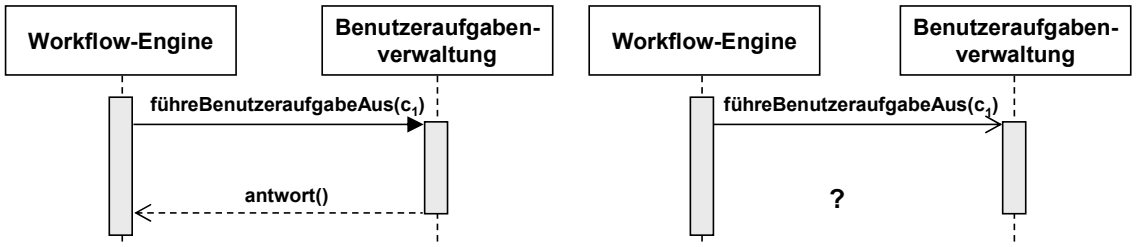

Abbildung 33: Synchrone und asynchrone Operationsaufrufe einer Benutzeraktion

Da die Überwachung einer Benutzeraufgabe für diese Arbeit eine zentrale Herausforderung darstellt, werden im weiteren Verlauf nur synchrone Benutzeraktionen betrachtet, die dementsprechend – wie im konzeptionellen Metamodell bereits festgehalten – immer einen Ausgabeparameter haben müssen, um die eine Rückmeldung des Benutzers registrieren zu können. Daher repräsentiert in einem Workflow-Modell ein Ausgabepin einen Ausgabeparameter und dieser wiederum ein Geschäftsobjekt, das durch die Interaktion des Benutzers seinen Zustand ändert. Eine genauere Betrachtung dieser Zustandsänderung anhand der Eigenschaften eines Geschäftsobjektes ist auf der Ebene des Workflow-Modells noch nicht von Belang und wird daher erst im Kontext des in Abschnitt 4.4 eingeführten benutzerzentrischen Domänenmodells näher beleuchtet.

4.2.2.2 Erweiterung der Metaklasse CallBehaviorAction

Eine **manuelle Aktion** wird im Benutzeraktionsprofil durch die Erweiterung der Metaklasse `CallBeaviorAction` spezifiziert. Die OMG definiert diese Metaklasse als Aufrufaktion, die ein Verhalten anstatt über eine Operation direkt aufrufen kann. Da manuelle Aktionen aufgrund der fehlenden IT-Unterstützung durch diese Arbeit nicht weiter betrachtet werden, wird eine detaillierte Spezifikation der manuellen Aktion nicht weiter verfolgt.

4.2.3 Anwendung des Benutzeraktionsprofils

Mithilfe des Benutzeraktionsprofils kann das in Abbildung 30 (Seite 81) gezeigte Beispiel eines Geschäftsprozessmodells hinsichtlich der verwendeten Aktionstypen verfeinert werden. Abbildung 34 zeigt den durch die spezifizierten Stereotypen verfeinerten und um Ein- und Ausgabeparameter erweiterten Geschäftsprozess. G sei der Bezeichner für ein Geschäftsobjekt, das durch die verschiedenen Aktionen entlang der modellierten Datenflüsse bearbeitet wird. Durch die Präzisierung des jeweiligen Typs der einzelnen Aktionen wird gleichzeitig festgelegt, welche Teile des Geschäftsprozesses durch einen Workflow unterstützt werden sollen. Da manuelle Aktionen zum Zeitpunkt des Entwurfs des Geschäftsprozessmodells nicht durch ein Anwendungssystem unterstützt werden können oder sollen, müssen diese in einem Workflow-Modell nicht mehr betrachtet werden. Die verbleibenden zusammenhängenden Benutzer- und Systemaktionen bilden somit Ausschnitte des ursprünglichen Geschäftsprozesses, die durch einen jeweils eigenständigen Workflow unterstützt werden können. Der aus Abbildung 34 abgeleitete Workflow besteht somit aus den Aktionen A_1, B_1 sowie C_1, B_3 und B_2. Zusätzlich wird eine weitere Kontrollflusskante von B_2 zum Endknoten benötigt. Die manuelle Aktion A_2 wird somit im Folgenden nicht weiter betrachtet. Eine automatische Transformation eines Geschäftsprozessmodells zu einem oder mehreren Workflow-Modellen ist aufgrund der bekannten Metamodelle nach dem eben beschriebenen Vorgehen theoretisch möglich. In der Praxis sind jedoch im Vergleich zu dem zur Veranschaulichung verwendeten Beispiel dieser Arbeit weitaus komplexere Geschäftsprozesse mit verschiedensten Kontrollflüssen zwischen den einzelnen Aktionen vorhanden. Eine automatische Transformation muss somit durch ein entsprechend mächtiges Transformations-

Regelwerk unterstützt werden, dessen Entwicklung nicht im Fokus dieser Arbeit liegt. Der Übergang von einem Geschäftsprozessmodell zu einem Workflow-Modell erfolgt daher manuell nach dem beschriebenen Vorgehen.

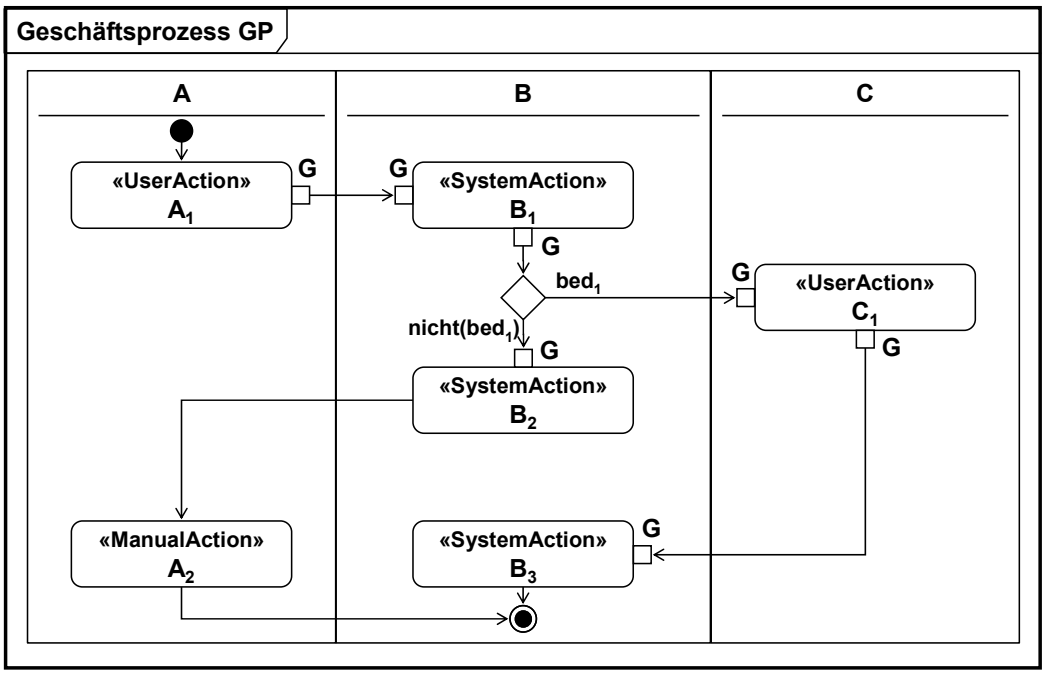

Abbildung 34: Erweiterung der Spezifikation des Geschäftsprozessmodells

Das Workflow-Modell, das vollständig auf die IT-Unterstützung abgebildet werden kann, dient nun im weiteren Verlauf als Ausgangspunkt der Ableitung weiterer plattformunabhängiger Modelle. Der folgende Abschnitt führt als erstes Folgemodell das plattformunabhängige Benutzeraufgabenmodell ein, welches eine detaillierte Spezifikation der aus den Benutzeraktionen heraus entstehenden Benutzeraufgaben ermöglicht.

4.3 Spezifikation der Benutzeraufgaben

Die im vorangegangenen Abschnitt eingeführten Benutzeraktionen spezifizieren Arbeitseinheiten in einem Workflow, die durch einen Menschen in Interaktion mit einem Anwendungssystem zu erbringen sind. Nach der in Abschnitt 2.2 vorgenommenen Definition resultiert daher jede Benutzeraktion in einer Benutzeraufgabe, die durch einen Menschen mit geeigneten Fähigkeiten erledigt werden muss. Die Interaktion eines Menschen im Rahmen einer Benutzeraufgabe kann im Gegensatz zu einem Systemverhalten zur Entwurfszeit nicht präzise spezifiziert werden, da das Verhalten eines Menschen von einer Vielzahl von Faktoren abhängig ist. Die Motivation, die Fähigkeit und die Verfügbarkeit eines Menschen seien als Beispiele für diese Faktoren genannt. Da Benutzeraktionen in der Ausführung eines Workflows synchron abgearbeitet werden und daher den Workflow blockieren, ist eine zielgerichtete und möglichst rasche Abarbeitung der Instanzen einer Benutzeraktion anzustreben. Daher muss eine Benutzeraufgabeninstanz, die bei der Abarbeitung einer Instanz einer Benutzeraktion erzeugt wird, überwacht werden, um so die Ausführung der Benutzeraufgabeninstanz bei Bedarf bis zu einem gewissen Grad beeinflussen zu können. Die Ressourcenperspektive eines Workflows rückt daher in den Mittelpunkt dieses Abschnitts (vgl. dazu auch [RA+05]).

Immer wiederkehrende Anforderungen an die Spezifikation der Ressourcenperspektive eines Workflows haben sich heute in Form von verschiedenen Workflow-Mustern etabliert (vgl. Abschnitt 2.2.3).

Beispielsweise kommt es immer wieder vor, dass eine Benutzeraufgabeninstanz einer menschlichen Ressource R_1 zugeordnet wird, diese aber beispielsweise aufgrund ihrer aktuellen Belastung die Benutzeraufgabe nicht innerhalb gegebener zeitlicher Rahmenbedingungen abarbeiten kann. Um in solch einer Situation geeignet reagieren zu können, sieht das in Abschnitt 2.2.3.1 beschriebene Delegationsmuster bereits zur Entwurfszeit die Zuordnung einer zweiten menschlichen Ressource R_2 zu einer Benutzeraufgabe vor, die dann zur Laufzeit die Abarbeitung der Benutzeraufgabeninstanz übernehmen muss. Um die Ressourcenperspektive in diesem Detaillierungsgrad in einem modellgetriebenen Entwicklungsvorgehen berücksichtigen und deren Anforderungen durch eine Modellierungssprache spezifizieren zu können, bedarf es eines geeigneten Metamodells.

Eine weitere Anforderung an die Ressourcenperspektive resultiert aus der bereits angeführten Tatsache, dass ein Benutzerverhalten nicht im Voraus präzise festgelegt werden kann und somit insbesondere die Dauer der Ausführung einer Benutzeraufgabeninstanz nicht vorhersagbar ist. Da eine Benutzeraufgabeninstanz durch die aufrufende synchrone Benutzeraktion die Fortführung einer Workflow-Instanz blockiert, müssen durch die verwendete Modellierungssprache geeignete Indikatoren spezifiziert werden können, anhand der die Ausführung einer Benutzeraufgabeninstanz überwacht und gegebenenfalls steuernd in die Ausführung eingegriffen werden kann. Als einfaches Beispiel eines solchen Indikators sei die Zeitspanne genannt, binnen der eine Benutzeraufgabeninstanz abgearbeitet werden muss. Erfolgt die vollständige Abarbeitung einer Benutzeraufgabeninstanz nicht innerhalb dieser Zeitspanne, dann müssen entsprechende Eskalationsmechanismen greifen, um die Ausführung der blockierten Workflow-Instanz fortsetzen zu können. Ein möglicher Eskalationsmechanismus ist beispielsweise der Entzug der Benutzeraufgabeninstanz von ihrem aktuellen Bearbeiter und die anschließende Neuvergabe an eine weitere menschliche Ressource.

Es lässt sich festhalten, dass jede Benutzeraktion eines Workflows zur Entwurfszeit in Bezug auf die Ressourcenperspektive im Rahmen einer Benutzeraufgabe genauer spezifiziert und zur Ausführungszeit des Workflows überwacht werden muss. Die Spezifikation von Benutzeraufgaben wird von bestehenden Modellierungssprachen jedoch nur ungenügend adressiert (vgl. dazu Kapitel 3.2). Daher müssen viele Details einer Benutzeraufgabe, obwohl diese bereits zur Modellierungsphase eines geschäftsgetriebenen Softwareentwicklungsprozesses vorhanden sind, während der Inbetriebnahmephase eines neuen Workflows manuell konfiguriert werden. Sofern die bestehende IT-Infrastruktur die Ausführung und Überwachung von Benutzeraufgabeninstanzen überhaupt zulässt, entsteht folglich für jede einzelne Benutzeraufgabe ein hoher manueller Konfigurationsaufwand. Gleichzeitig müssen diese Konfigurationen plattformspezifisch für die jeweils zu verwendende Ausführungsumgebung für Benutzeraufgaben entwickelt werden, wodurch eine Portabilität dieser Konfigurationen nicht gegeben ist. Es muss daher das Ziel sein, Benutzeraufgaben bereits in der Modellierungsphase plattformunabhängig spezifizieren zu können, um so eine frühzeitige und portable Erfassung der bei den Entwicklungsrollen vorhandenen Informationen über die Benutzeraufgaben zu ermöglichen.

4.3.1 Entwurf des Benutzeraufgabenmetamodells

Die vorangegangene Motivation und die daraus abgeleiteten Anforderungen gehen nun in den Entwurf eines Metamodells für Benutzeraufgaben ein. Ziel dieses Metamodells ist die Bereitstellung ausdrucksmächtiger Modellelemente zur plattformunabhängigen Spezifikation einer Benutzeraufgabe. Es werden zunächst in einem konzeptionellen Entwurf grobgranular die Elemente beschrieben, die im Rahmen dieser Arbeit zur Spezifikation einer Benutzeraufgabe benötigt werden.

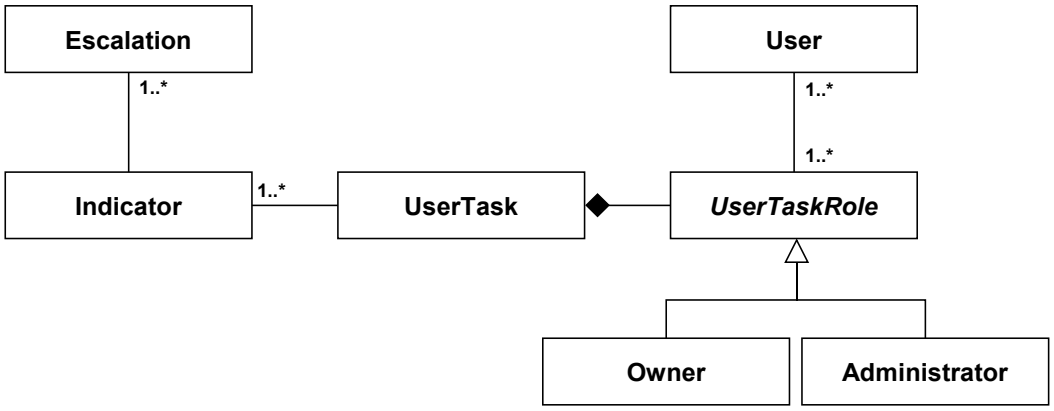

Abbildung 35: Entwurf des konzeptionellen Metamodells für Benutzeraufgaben

Zentrales Element des in Abbildung 35 gezeigten konzeptionellen Entwurfs des Metamodells für Benutzeraufgaben ist die Benutzeraufgabe (`UserTask`) selbst. Jede Benutzeraktion eines Workflows instanziiert eine Benutzeraufgabe (vgl. Abschnitt 4.2). Eine Benutzeraufgabe unterscheidet sich von einer Benutzeraktion insofern, dass bei der Spezifikation einer Benutzeraufgabe ein Perspektivenwechsel vollzogen wird und nicht mehr der Kontrollfluss eines Workflows, sondern die zu dessen Ausführung benötigten menschlichen Ressourcen im Sinne von Benutzern (`User`) fokussiert werden. Ein Benutzer nimmt dabei im Kontext einer Benutzeraufgabe immer mindestens eine Benutzeraufgabenrolle (`UserTaskRole`) ein. Beispielsweise muss ein Benutzer für die Ausführung einer Benutzeraufgabe zuständig sein und steht damit zu dieser in der Benutzeraufgabenrolle des Eigentümers (`Owner`).

Da die Überwachung der Ausführung einer Benutzeraufgabe eine zentrale Anforderung darstellt, wird neben der Benutzeraufgabenrolle des Ausführenden zusätzlich die Benutzeraufgabenrolle eines Administrators (`Administrator`) benötigt. Ferner spezifiziert eine Benutzeraufgabenrolle eine Verantwortlichkeit gegenüber einer Benutzeraufgabe, die durch eine menschliche Ressource übernommen werden muss. Dementsprechend muss eine Benutzeraufgabenrolle immer durch mindestens einen Benutzer belegt werden. Wird eine Benutzeraufgabe nicht ihm Rahmen der spezifizierten Parameter ausgeführt, so müssen entsprechende Eskalationsmechanismen (`Escalation`) greifen und beispielsweise die Benutzeraufgabenrolle des Administrators mit der Lösung der Problemsituation beauftragt werden. Um die Auslösung einer Eskalation veranlassen zu können, sind entsprechende Indikatoren (`Indicator`) notwendig, welche Aufschluss über den Abarbeitungszustand einer Benutzeraufgabe geben. Eine Eskalation muss somit durch mindestens einen Indikator näher spezifiziert werden.

Auf diesem konzeptionellen Entwurf aufbauend kann das Metamodell zur Spezifikation von Benutzeraufgaben konkretisiert werden. Als Erweiterung des konzeptionellen Entwurfs sieht das Metamodell einige feinere Unterscheidungen wie beispielsweise die getrennte Betrachtung einer Benutzeraufgabe und einer Eskalationsaufgabe vor. Ferner greift das Metamodell die in Abschnitt 2.2.1 eingeführte Klassifikation menschlicher Ressourcen auf und gestattet deren detaillierte Spezifikation. Abbildung 36 illustriert die einzelnen Modellelemente des Metamodells, die im Folgenden detailliert eingeführt und beschrieben werden.

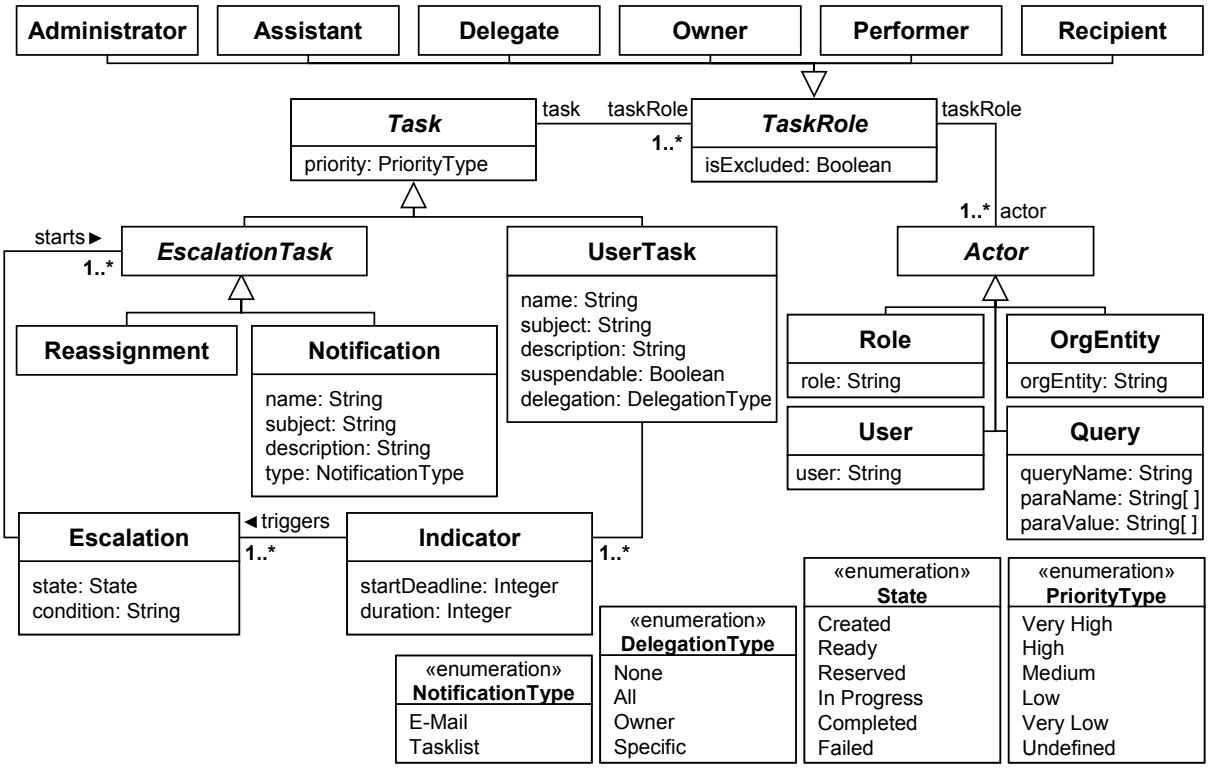

Abbildung 36: Metamodell zur Spezifikation von Benutzeraufgaben

4.3.1.1 Aufgabe

Eine Aufgabe (`Task`) spezifiziert eine zu erledigende Arbeitseinheit in Bezug auf eine konkrete Ressource. Jede Aufgabe kann mit einer Priorität (`priority`) versehen werden, welche genau dann von Relevanz ist, falls einer Ressource unterschiedliche Aufgaben zugeordnet werden können. Durch ein einheitliches Vorgehen bei der Priorisierung von Aufgaben kann in diesem Fall die einzuhaltende Abarbeitungsreihenfolge bestimmt werden. Die im Metamodell vorhandene Enumeration `PriorityType` sieht unterschiedliche Prioritäten von sehr hoch (`Very High`) bis sehr gering (`Very Low`) vor, lässt aber auch unbestimmte Prioritäten zu (`Undefined`), falls einer Aufgabe keine Priorität zugeordnet werden soll oder nicht zugeordnet werden kann.

Benutzeraufgabe

Eine Benutzeraufgabe (`UserTask`) beschreibt eine durch eine menschliche Ressource durchzuführende Arbeitseinheit näher. Sie benötigt einen Namen (`name`), einen Betreff (`subject`) sowie eine Beschreibung (`description`). Durch diese drei Eigenschaften kann die Benutzeraufgabe inhaltlich detailliert beschrieben und ein Benutzer über die durch die Benutzeraufgabe geforderte Arbeitseinheit informiert werden. Wie alle Aufgaben verfügt eine Benutzeraufgabe über eine Priorität (`priority`). Über diese Eigenschaft kann bereits zur Entwurfszeit die Prioritätsstufe einer Benutzeraufgabe festgelegt werden, die allen Instanzen dieser Benutzeraufgabe zum Erzeugungszeitpunkt zugewiesen werden soll. Zur Ausführungszeit einer Benutzeraufgabeninstanz ist die zugewiesene Prioritätsstufe jedoch kein statisch zugewiesener Eigenschaftswert. Wird eine Benutzeraufgabeninstanz beispielsweise durch eine Eskalation einem neuen Benutzer zugeteilt, so ist in diesem Zuge die Zuweisung einer höheren Priorität sinnvoll, sofern die Benutzeraufgabe nicht bereits die höchstmögliche Prioritätsstufe zugewiesen bekommen hat.

Eine Benutzeraufgabe verfügt ferner über eine Delegationseigenschaft (`delegation`). Die Bearbeitungszeit, die ein menschlicher Benutzer zur Abarbeitung einer Benutzeraufgabe benötigt, ist in den seltensten Fällen genau berechenbar. Tritt daher der Fall ein, dass ein Benutzer selbstständig vorhersehen kann, dass er eine Benutzeraufgabe bis zu einer gesetzten Frist nicht erfüllen kann oder er aus anderweitigen Gründen die Benutzeraufgabe nicht eigenständig bearbeiten will, so kann er sie, sofern die Eigenschaft `delegation` der Benutzeraufgabe dies gestattet, an einen dazu bevollmächtigten Benutzer delegieren. Dabei kann durch die in der Enumeration `DelegationType` erfassten möglichen Eigenschaftswerte unterschieden werden, ob eine Benutzeraufgabe nur an einen weiteren Eigentümer (`Owner`), an alle weiteren Aufgabenrollen (`All`), nur an den eigens in der Benutzerrolle des Delegierten spezifizierten Akteur (`Specific`) oder gar nicht (`None`) delegierbar ist.

Die Spezifikation einer Benutzeraufgabe ist in Bezug auf die weiteren Elemente des in Abbildung 39 gezeigten Metamodells einigen Einschränkungen unterworfen, die nicht (oder nicht eindeutig) durch die verwendete MOF-Syntax beschrieben werden können und daher durch zusätzliche OCL-Ausdrücke formalisiert werden müssen. Die in den folgenden Einschränkungen erwähnten Metamodellelemente werden in den folgenden Abschnitten detailliert beleuchtet. Zunächst muss einer Benutzeraufgabe immer mindestens ein Akteur (`Actor`) zugeordnet werden, der die Aufgabenrolle (`TaskRole`) eines Eigentümers (`Owner`) innehat. Da die Ausführung einer Benutzeraufgabe ferner immer überwacht werden muss, bedarf es eines weiteren Akteurs, der als Administrator (`Administrator`) der Benutzeraufgabe fungiert. Um eine eindeutige Zuständigkeit eines Administrators zu einer Benutzeraufgabe sicherstellen zu können, darf immer nur genau ein Administrator einer Benutzeraufgabe zugeordnet werden. Der folgende OCL-Ausdruck stellt die Einhaltung dieser Bedingungen sicher.

```
context UserTask
inv: self.taskRole -> select(oclIsTypeOf(Owner)) -> size() > 0
inv: self.taskRole -> select(oclIsTypeOf(Administrator)) -> size() = 1
```

Ferner kann eine Benutzeraufgabe delegiert werden, sofern der eigentlich zur Ausführung beauftragte Benutzer sich außerstande sieht, diese innerhalb gewisser Randbedingungen ausführen zu können. Falls die Delegationseigenschaft der Benutzeraufgabe (`delegation`) den Wert `specific` trägt, ist die Zuordnung der Aufgabenrolle des Delegierten zur Benutzeraufgabe verpflichtend. Der folgende OCL-Ausdruck formalisiert diese Bedingung.

```
context UserTask inv:
  (self.delegation = DelegationType::specific) implies
    self.taskRole -> select(oclIsTypeOf(Delegate)) -> size() > 0
```

Eine weitere Einschränkung betrifft die Aufgabenrolle des Empfängers (`Recipient`). Diese Aufgabenrolle darf einer Benutzeraufgabe nicht zugeordnet werden, da der Empfänger – wie im Folgenden erläutert – nur zwischen einem Akteur und einer Benachrichtigung eingesetzt werden darf.

```
context UserTask inv:
  self.taskRole -> select(oclIsTypeOf(Recipient)) -> size() = 0
```

Neben der Spezifikation der statischen Semantik des Metamodells für Benutzeraufgaben muss insbesondere die dynamische Semantik und damit das Verhalten einer Benutzeraufgabeninstanz zur Laufzeit als zentrales Element des Metamodells näher betrachtet werden (vgl. Abschnitt 2.1.1). Während der Ausführung einer solchen Benutzeraufgabeninstanz ändert diese ihren Zustand und durchläuft

dabei einen Lebenszyklus. Dieser soll im Folgenden in Anlehnung an [AA+07b] und [RH+04] zunächst textuell beschrieben und anschließend auf die formale Semantik eines in Abbildung 37 in Form eines UML-Zustandsdiagramms dargestellten endlichen Automaten abgebildet werden.

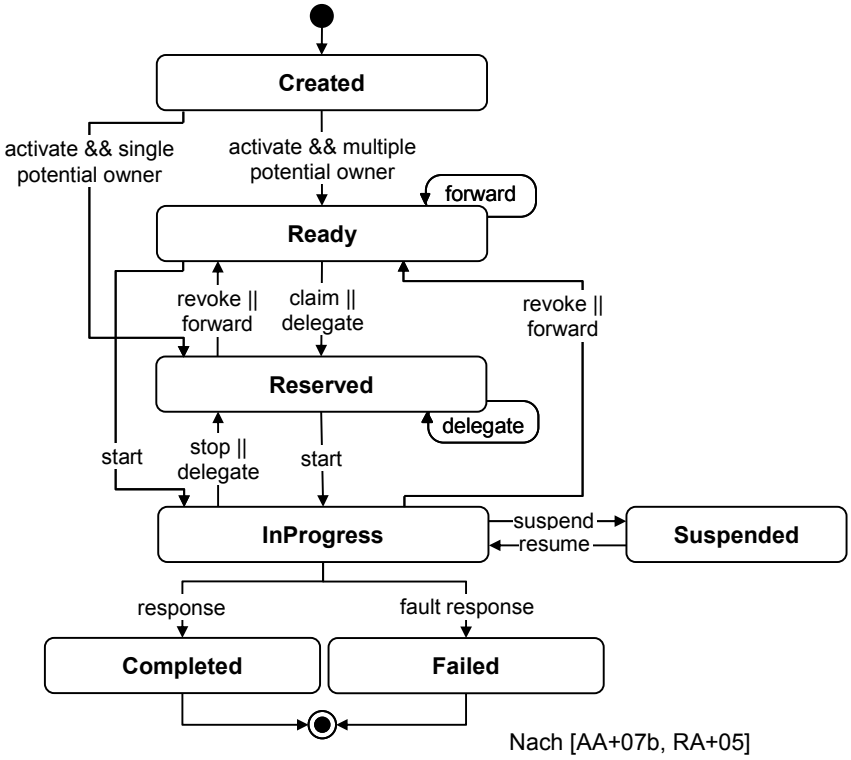

Abbildung 37: Verhalten einer Benutzeraufgabeninstanz

Wird während der Ausführung einer Workflow-Instanz eine Benutzeraktion erreicht, so wird eine Instanz derjenigen Benutzeraufgabe erzeugt (`Created`), die der Benutzeraktion zugeordnet ist. Daraufhin muss beispielsweise unter Zuhilfenahme eines Organisationsverzeichnisses ermittelt werden, welche Benutzer einer Gruppe oder welche Benutzer mit einer spezifischen Rolle zur Bearbeitung dieser Benutzeraufgabeninstanz qualifiziert sind. Wurde der Benutzeraufgabe schon zur Entwurfszeit im Modell ein konkreter Benutzer zugeordnet oder kann nur ein einziger geeigneter Benutzer über das Organisationsverzeichnis ermittelt werden, so wird die Instanz der Benutzeraufgabe sofort für diesen Benutzer reserviert (`Reserved`). Können hingegen mehrere geeignete Benutzer über eine Gruppe oder eine Rolle ermittelt werden, so wird die Instanz für all diese Benutzer zur Bearbeitung freigegeben (`Ready`). In diesem Zustand kann die Benutzeraufgabeninstanz von allen ermittelten Benutzern zur Bearbeitung angefordert werden. Erkennt ein administrativ handelnder Benutzer in dieser Situation, dass ein weiterer Benutzer zu der Menge der potenziellen Bearbeiter hinzugenommen werden sollte, so kann er die Benutzeraufgabeninstanz an weitere Benutzer weiterleiten (`forward`). Erklärt sich einer dieser potenziellen Bearbeiter bereit, eine Benutzeraufgabeninstanz abzuarbeiten (`claim`), so wird die Instanz für ihn reserviert (`Reserved`). Anschließend kann der Benutzer die Bearbeitung der Instanz beginnen (`start`) und überführt diese damit in den Zustand der Bearbeitung (`InProgress`). Die Bearbeitung der Benutzeraufgabeninstanz wird entweder erfolgreich beendet (`response, Completed`) oder aufgrund eines Fehlers abgebrochen (`fault response, Failed`). Im Zustand der Bearbeitung (`InProgress`) kann die Bearbeitung einer Benutzeraufgabeninstanz durch den Benutzer oder einen Administrator einerseits vollständig gestoppt (`stop`) werden, wodurch die Instanz in den Zustand `Ready` zurückversetzt wird. Andererseits kann die

Bearbeitung durch den Benutzer ausgesetzt (`Suspend`) im Sinne von pausiert und anschließend wieder fortgesetzt werden (`Resume`), sofern die Spezifikation der Benutzeraufgabe eine Aussetzung zulässt. Um eine Aussetzung zu erlauben oder zu verweigern, wird die Eigenschaft `suspendable` der Benutzeraufgabe herangezogen, welche die booleschen Werte `true` oder `false` annehmen kann. Das Aussetzen einer Benutzeraufgabeninstanz ist genau dann sinnvoll, wenn ein Benutzer zwar über längere Zeit nicht an der Instanz arbeiten, deren Abarbeitung im Rahmen der gesetzten zeitlichen Restriktionen jedoch sicherstellen kann. Durch diesen Mechanismus wird verhindert, dass sich die Gesamtbearbeitungsdauer einer Benutzeraufgabe unnötig erhöht.

Die textuell beschriebene dynamische Semantik einer Benutzeraufgabe wird durch den in Abbildung 37 in Form eines UML-Zustandsdiagramms dargestellten endlichen Automaten formalisiert. Die spezifizierten Zustandsübergänge werden zur Laufzeit durch Akteure in unterschiedlichen Aufgabenrollen (siehe Abschnitt 4.3.1.4), aber auch durch das verwendete System zur Benutzeraufgabenverwaltung herbeigeführt. Die mit `delegate` beschrifteten Zustandsübergänge können aus der zuvor eingeführten Delegationseigenschaft einer Benutzeraufgabe abgeleitet werden.

Eskalationsaufgabe

Eine Eskalationsaufgabe (`EscalationTask`) spezifiziert eine durch ein System zu erbringende Arbeitseinheit im Rahmen einer Eskalation. Eine Eskalationsaufgabe wird dann instanziiert, wenn durch einen spezifizierten Indikator eine Eskalation ausgelöst werden soll und die Bedingungen der Eskalation erfüllt sind. Das Metamodell sieht konkret die im Folgenden beschriebenen zwei Eskalationsaufgaben vor.

Benachrichtigung

Wird eine Eskalationsinstanz durch eine Indikatorinstanz ausgelöst, so kann zur Umsetzung dieser Eskalation eine einfache Benachrichtigung (`Notification`) eines entsprechend zugeordneten Empfängers erfolgen, die diesen über Grund und Ursache der Eskalation informiert. Folglich muss sichergestellt werden, dass bei der Spezifikation einer Benachrichtigung immer mindestens einmal die Aufgabenrolle eines Empfängers zugeordnet werden muss. Weitere Aufgabenrollen dürfen der Benachrichtigung nicht zugeordnet werden. Der folgende OCL-Ausdruck nimmt diese Einschränkung vor.

```
context Notification
inv: self.taskRole -> select(oclIsTypeOf(Recipient)) -> size() > 0
inv: self.taskRole -> forAll(oclIsTypeOf(Recipient))
```

Eine Benachrichtigung kann in Abhängigkeit der Typeigenschaft (`type`) durch eine E-Mail oder über die Aufgabenliste des Empfängers erfolgen. Eine Benachrichtigung besteht, vergleichbar zur Benutzeraufgabe, aus einem Namen (`name`), einem Betreff (`subject`) und einer Beschreibung (`description`). Da bei einem Empfänger jederzeit mehrere Benachrichtigungen eingehen können, ermöglicht die vererbte Eigenschaft Priorität (`priority`) eine Sortierung der Benachrichtigungen nach ihrer Wichtigkeit. Ferner muss eine Benachrichtigung im Gegensatz zu einer Benutzeraufgabe asynchron betrachtet werden, da sie rein informativen Charakter, aber kein konkretes Arbeitsergebnis hat. Es ist daher wichtig, dass einer der möglichen Empfänger einer Benachrichtigung über eine Eskalation derjenige Akteur ist, der die Aufgabenrolle des Administrators erfüllt, da dieser entsprechende Maßnahmen ergreifen und in die Ausführung der Aufgabenrolle steuernd eingreifen kann. Diese Anforderung wird durch den folgenden OCL-Ausdruck formalisiert.

```
context Notification inv:
  self.taskRole -> select(oclIsTypeOf(Recipient)).actor ->
       select(oclIsTypeOf(User)) ->
     exists(recipientUser |
         self.escalation.indicator.userTask.taskRole ->
         select(oclIsTypeOf(Administrator)).actor ->
         select(oclIsTypeOf(User)) ->
       exists(administratorUser | recipientUser == administratorUser))
```

Neuzuweisung

Eine weitere Form der Umsetzung einer Eskalation ist eine Neuzuweisung (`Reassignment`) einer Benutzeraufgabe. Eine Neuzuweisung ist beispielsweise dann sinnvoll, wenn ein Benutzer eine Benutzeraufgabe als Eigentümer für sich reserviert hat, dann aber aus irgendwelchen Gründen bis zu einem gewissen Zeitpunkt nicht mit der Bearbeitung der Benutzeraufgabe beginnt. Dann muss die Benutzeraufgabe an einen neuen Eigentümer zugewiesen werden. Folglich muss eine Neuzuweisung immer mit der Aufgabenrolle eines Eigentümers assoziiert sein. Dabei muss darauf geachtet werden, dass, sofern der Neuzuweisung als Eigentümer ein konkreter Benutzer als Akteur zugewiesen wird, dieser Eigentümer nicht gleichzeitig der ursprüngliche Eigentümer der Benutzeraufgabe ist.

```
context Reassignment
-- Reassignments may have only one owner
inv: self.taskRole -> select(oclIsTypeOf(Owner)) -> size() = 1

-- One user may not be ass. with Reassignment and Owner of same UserTask
inv: self.taskRole -> select(oclIsTypeOf(Owner)).actor ->
         select(oclIsTypeOf(User)) ->
       forAll(reassignmentUser |
         self.escalation.indicator.userTask.taskRole ->
         select(oclIsTypeOf(Owner)).actor -> select(oclIsTypeOf(User)) ->
         forAll(ownerUser | reassignmentUser != ownerUser)))
```

Die dynamische Semantik einer Eskalationsaufgabe kann durch den in Abbildung 38 in Form eines UML-Zustandsdiagramms dargestellten endlichen Automaten formalisiert werden.

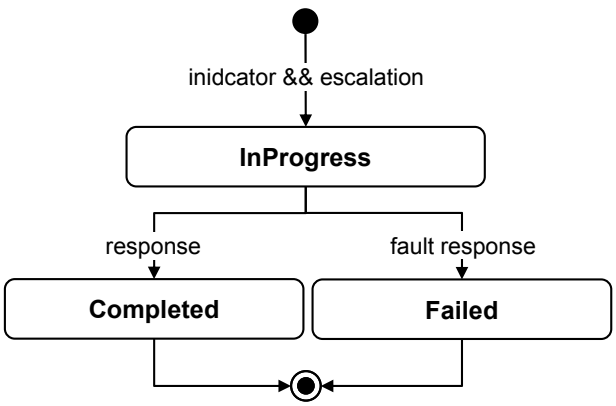

Abbildung 38: Verhalten einer Systemaufgabe

Die Instanz einer Eskalationsaufgabe wird im Gegensatz zu einer Benutzeraufgabeninstanz erst dann erzeugt, wenn sowohl ein Indikator einer Benutzeraufgabe als auch die dem Indikator zugeordnete Eskalation eine Eskalationsaufgabe auslösen (`indicator && escalation`). Dementsprechend

wird eine Eskalationsaufgabeninstanz aus dem Initialzustand direkt in den Zustand der Bearbeitung (`inProgress`) überführt. Die Ausführung kann erfolgreich beendet (`response`, `Complete`) oder aufgrund eines Fehlers abgebrochen werden (`fault response`, `Failed`).

4.3.1.2 Indikator

Ein Indikator dient zur Spezifikation von Parametern, die der Überwachung einer Benutzeraufgabe dienen. Hierbei können theoretisch unterschiedlichste Arten von Indikatoren, wie beispielsweise Performanz- oder Kostenindikatoren zum Einsatz kommen (vgl. hierzu auch [Ho08a]). Für die Zwecke dieser Arbeit ist jedoch ein sehr einfacher Performanzindikator (kurz: Indikator), der über die beiden Eigenschaften „letzter Starttermin“ (`startDeadline`) und Dauer (`duration`) verfügt, ausreichend. Über den letzten Starttermin kann angegeben werden, zu welchem Zeitpunkt mit der Bearbeitung einer Benutzeraufgabe spätestens begonnen werden muss. Dieser Zeitpunkt wird dabei immer relativ zur Erzeugung der Benutzeraufgabeninstanz festgelegt. Ein möglicher letzter Starttermin könnte somit den Wert 86 400 als die Anzahl der Sekunden eines Tages haben und damit den letzten Starttermin genau einen Tag nach Erzeugung der Benutzeraufgabeninstanz festsetzen. Die zweite Eigenschaft Dauer dient zur Spezifikation der Zeitspanne, welche die Bearbeitung einer Benutzeraufgabeninstanz maximal in Anspruch nehmen darf. Durch eine geeignete Kombination der beiden Eigenschaften eines Indikators lässt sich ebenfalls der Zeitpunkt festlegen, zu welchem die Bearbeitung der Benutzeraufgabeninstanz spätestens abgeschlossen sein muss.

Weitere Ausprägungen von Indikatoren werden bewusst nicht betrachtet, da der angeführte Performanzindikator für die Realisierung der Ziele dieser Arbeit ausreichend ist und das Metamodell jederzeit eine Erweiterung um zusätzliche Indikatoren zulässt. Dennoch liegt in der Auswahl und Spezifikation von Indikatoren, die für die Überwachung der Interaktion eines Menschen relevant sind, eine weitreichende Fragestellung (siehe dazu auch [FH+09]). Aus diesem Grund wird diese Thematik im Ausblick zu dieser Arbeit in Abschnitt 7.3 erneut aufgegriffen.

4.3.1.3 Eskalation

Eine Eskalation (`Escalation`) wird durch einen Indikator ausgelöst (`triggers`), der einer Benutzeraufgabe zugeordnet ist. Durch die beiden Eigenschaften Bedingung (`condition`) und Status (`state`) einer Eskalation kann gesteuert werden, wann eine Eskalation eine Eskalationsaufgabe auslöst (`starts`), die dann wiederum die Ausführung einer Benutzeraufgabeninstanz beeinflusst. Über die Statuseigenschaft einer Eskalation kann spezifiziert werden, dass Eskalationen nur dann eine Eskalationsaufgabe auslösen, wenn sich die zugeordnete Benutzeraufgabe im angegebenen Zustand wie beispielsweise `Reserved` befindet. Auf diese Weise können Eskalationen präzise für die verschiedenen Zustände der Benutzeraufgabeninstanzen spezifiziert werden. Über die Bedingungseigenschaft kann beispielsweise festgelegt werden, dass eine Eskalation nur dann eine Eskalationsaufgabe auslösen soll, wenn der aktuelle Tag ein Werktag ist. Wird die Bedingung der Eskalation zu dem booleschen Wert `true` ausgewertet und befindet sich die zugeordnete Benutzeraufgabe im gleichen Zustand, wie in der Eskalation spezifiziert wurde, so wird eine Eskalationsaufgabe wie etwa eine Benachrichtigung oder Neuzuweisung instanziiert. Daher muss einer Eskalation immer mindestens eine Eskalationsaufgabe zugeordnet werden.

4.3.1.4 Aufgabenrolle

Die Aufgabenrolle (`TaskRole`) wird im Metamodell zunächst abstrakt aufgefasst. Sie definiert, welche Verantwortung ein Akteur gegenüber einer Aufgabe übernimmt. Alle Aufgabenrollen besitzen die Eigenschaft `isExcluded`, über die spezifiziert werden kann, welche Aufgabenrolle ein Akteur explizit nicht einnehmen darf. So kann beispielsweise festgelegt werden, dass eine Gruppe von Personen zur Bearbeitung einer Benutzeraufgabe geeignet ist, ein konkreter Benutzer aus dieser Gruppe jedoch explizit nicht. Die einzelnen Aufgabenrollen werden nun jeweils kurz beleuchtet.

Eigentümer

Akteure mit der Aufgabenrolle des Eigentümers (`Owner`) sind für die Bearbeitung einer Benutzeraufgabe zuständig, weshalb diese immer wieder entsprechende Instanzen einer Benutzeraufgabe zur Abarbeitung angeboten bekommen. Die Eigentümer können zur Laufzeit den Zustand einer Benutzeraufgabeninstanz ändern und diese für sich reservieren, mit der Abarbeitung einer Instanz beginnen oder diese abschließen. Ebenso können sie die Priorität einer Benutzeraufgabeninstanz bearbeiten. Ist eine Benutzeraufgabe in einem für den Eigentümer reservierten Zustand, so darf der Eigentümer die Instanz der Benutzeraufgabe delegieren. Die Aufgabenrolle des Eigentümers kann nur durch einen Benutzer gegenüber einer Benutzeraufgabe oder einer Neuzuordnung eingenommen werden. Andere Zuweisungen sind nicht zulässig. Der folgende OCL-Ausdruck stellt die Einhaltung dieser Bedingungen sicher.

```
context Owner inv:
  self.task -> forAll(task | task.oclIsTypeOf(UserTask) or
                             task.oclIsTypeOf(Reassignment))
```

Administrator

Die Aufgabenrolle des Administrators (`Administrator`) überwacht die Ausführung der ihr zugeordneten Benutzeraufgaben. Der Administrator kann zugewiesene Instanzen einer Benutzeraufgabe einsehen, deren Status bearbeiten oder neue Prioritäten vergeben. Folglich hat ein Administrator mehr Rechte an einer Benutzeraufgabe als ein Eigentümer. Die Aufgabenrolle des Administrators kann nur einer Benutzeraufgabe oder einer Benachrichtigung zugewiesen werden. Der folgende OCL-Ausdruck ermöglicht eine Konsistenzprüfung der Modelle gegenüber dieser Einschränkung.

```
context Administrator inv:
  self.task -> forAll(task | task.oclIsTypeOf(UserTask) or
                             task.oclIsTypeOf(Notification))
```

Die Einschränkung, dass jeweils nur ein Administrator je Benutzeraufgabe oder Benachrichtigung definiert werden kann und dieser ein konkreter Benutzer sein muss, wird im Kontext der Metamodellelemente Benutzeraufgabe und Benachrichtigung spezifiziert.

Delegierter

Ein Delegierter (`Delegate`) hat per se keinen Einfluss auf die Bearbeitung einer Benutzeraufgabe. Er kann dazu nur durch einen Eigentümer beauftragt werden. Sofern die Delegationseigenschaft einer Benutzeraufgabe die Weitergabe einer Benutzeraufgabeninstanz an einen weiteren Eigentümer oder an einen spezifischen Akteur gestattet, so muss dieser mit der Benutzeraufgabe über die Aufgabenrolle Delegierter assoziiert sein. Als Beispiel sei ein Eigentümer „Dozent“ genannt, welcher eine Instanz

der Benutzeraufgabe „Prüfungstermin auswählen“ an einen Akteur „Sekretärin“ delegieren kann. Der Akteur „Sekretärin“ muss in diesem Beispiel somit die Aufgabenrolle „Delegierter“ einnehmen. Wird eine Benutzeraufgabeninstanz einem Delegierten zugewiesen, so erhält dieser die gleichen Rechte wie der Eigentümer. Die Aufgabenrolle Delegierter darf daher nur einer Benutzeraufgabe zugewiesen werden. Ferner muss darauf geachtet werden, dass diese Zuweisung eines Delegierten nur dann zulässig ist, wenn die Benutzeraufgabe eine spezifische Delegation erfordert (`specific`). Alle weiteren Eigenschaftswerte der Delegationseigenschaft einer Benutzeraufgabe benötigen die Zuordnung eines Delegierten nicht (vgl. Abschnitt 4.3.1.1).

```
context Delegate inv:
  self.task -> forAll(task | task.oclIsTypeOf(UserTask) and
    task.OclAsType(UserTask).delegation = DelegationType::specific)
```

Assistent

Die Aufgabenrolle des Assistenten (`Assistant`) kann den Fortschritt einer Benutzeraufgabeninstanz verfolgen und dem Eigentümer bei der Bearbeitung einer Benutzeraufgabeninstanz assistieren. Assistenten können den Zustand einer Benutzeraufgabeninstanz jedoch nicht ändern und daher von einem Assistenten weder gestartet, noch abschlossen werden. Die Aufgabenrolle Assistent steht immer in Bezug zu einer Benutzeraufgabe und darf mit keinem weiteren Modellelement assoziiert werden.

```
context Assistant inv:
  self.task -> forAll(oclIsTypeOf(UserTask))
```

Empfänger

Ein Empfänger (`Recipient`) ist im Rahmen einer Eskalationsaufgabe für den Empfang von Benachrichtigungen zuständig. Folglich darf diese Aufgabenrolle nur zwischen einer Benachrichtigung und einer Benutzeraufgabe bestehen.

```
Context Recipient inv:
  self.task -> forAll(oclIsTypeOf(Notification))
```

Alle Aufgabenrollen müssen, sofern sie in einem Benutzeraufgabenmodell spezifiziert wurden, durch einen Akteur belegt werden. Akteure als abstrakte Bezeichnung für menschliche Ressourcen werden im folgenden Abschnitt eingeführt.

Ausführender

Die Aufgabenrolle des Ausführenden bestimmt die menschliche Ressource, welche zur Ausführungszeit einer Workflow-Instanz mit der Abarbeitung einer Benutzeraufgabe betraut wird, näher. Da diese menschliche Ressource häufig erst zur Laufzeit bestimmt werden kann, muss die Aufgabenrolle des Ausführenden entweder zwischen zwei Benutzeraufgaben oder zwischen einer Benutzeraufgabe und einem konkreten menschlichen Benutzer modelliert werden.

4.3.1.5 Akteur

Das abstrakte Metamodellelement Akteur (`Actor`) dient zur Spezifikation menschlicher Ressourcen, die durch die drei im Metamodell vorgesehenen und im Folgenden beschriebenen Spezialisierungen eines Akteurs konkretisiert werden können.

Benutzer

In gewissen Fällen ist es sinnvoll, menschliche Ressourcen bereits zur Entwurfszeit in Form von konkreten Benutzern zu spezifizieren und diese über eine Aufgabenrolle an eine Benutzeraufgabe zu binden. Beispielsweise nimmt die Aufgabenrolle des Administrators eine hohe Verantwortung gegenüber der Ausführung aller Instanzen einer Benutzeraufgabe ein, weshalb eine präzise Bestimmung dieser menschlichen Ressource eine eindeutige Zuständigkeit zur Laufzeit sicherstellt. Soll festgehalten werden, dass ein ganz konkreter Benutzer eine Benutzeraufgabe nicht erfüllen darf, so kann dies ebenfalls unter Verwendung des Modellelements Benutzer (`User`) in Zusammenhang mit der entsprechenden Aufgabenrolle spezifiziert werden. In vielen weiteren Fällen muss zur Entwurfszeit einer Benutzeraufgabe kein konkreter Benutzer angegeben werden. Soll diese Zuordnung auf die Ausführungszeit einer Benutzeraufgabeninstanz verlagert werden, so stehen zur Entwurfszeit die beiden Modellelemente Rolle (`Role`) und Organisationseinheit (`OrgEntity`) zu einer abstrakten Spezifikation menschlicher Ressourcen zur Verfügung (vgl. auch Abschnitt 2.2.1).

Rolle

Eine Rolle beschreibt nach [Aa98] eine Gruppe von Ressourcen, die gleiche oder ähnliche Fähigkeiten aufweisen. Wird ein Akteur als Rolle konkretisiert, so bedeutet dies, dass beispielsweise eine Benutzeraufgabe „Prüfungstermin wählen“, die der Rolle „Dozent“ über die Aufgabenrolle „Eigentümer“ zugeordnet wird, durch alle menschlichen Ressourcen bearbeitet werden kann, die dieser Rolle angehören.

Organisationseinheit

Durch das Modellelement Organisationseinheit (`OrgEntity`) kann die organisatorische Struktur eines Unternehmens in einem Benutzeraufgabenmodell reflektiert werden. Wird ein Akteur durch eine Organisationseinheit konkretisiert, so bedeutet dies, dass alle menschlichen Ressourcen, die dieser Organisationseinheit zugehören, beispielsweise zur Ausführung einer Benutzeraufgabe herangezogen werden können. Die Organisationseinheit „Cooperation & Management“ sei als Beispiel angeführt.

Anfrage

Da die zur Erfüllung einer Benutzeraufgabe geeigneten menschlichen Ressourcen nicht immer über ihre Fähigkeiten oder ihre organisatorische Zugehörigkeit selektiert werden können, sieht das Metamodell zusätzlich das Modellelement Anfrage (`Query`) vor, mithilfe dessen freie Anfragen an ein Organisationsverzeichnis gestellt werden können. Dabei wird die Anfrage durch mindestens ein Name-Wert-Paar konkret bestimmt, sodass über eine Anfrage mit „Beschäftigungsdauer“ und „>10J“ beispielsweise alle Mitarbeiter ermittelt werden können, die mehr als 10 Jahre bei einer organisatorischen Einheit angestellt sind und dadurch als Ressourcen zur Abarbeitung von sensitiven Benutzeraufgaben besonders geeignet sind.

4.3.2 Das Benutzeraufgabenprofil

Das im vorangegangenen Abschnitt vorgestellte konzeptionelle Metamodell muss auf ein konkretes Metamodell abgebildet werden, damit die spezifizierten Modellelemente in einem modellgetriebenen Entwicklungsvorgehen verwendet werden können. Da diese Arbeit den Empfehlungen der modellgetriebenen Architektur der *Object Management Group* folgt [MM03], erfolgt die Umsetzung des konzeptionellen Metamodells auf das Metamodell der UML. In der UML-*Superstructure* [OMG-UML2-Super] findet sich das Paket der Anwendungsfälle, durch dessen Modellelemente Akteure und

deren Beziehungen zu einem System spezifiziert werden können. Dieser Modelltyp umfasst im Wesentlichen die drei Modellelemente Anwendungsfall (`UseCase`), Akteur (`Actor`) und Assoziation (`Association`). Bedingt durch die zentralen Modellelemente der Aufgabe (`Task`) und des Akteurs (`Actor`) des Benutzeraufgabenmetamodells liegt deren Umsetzung auf die Modellelemente eines UML-Anwendungsfalls nahe. Dementsprechend zeigt Abbildung 39 ein in [LH+09] publiziertes UML-Profil, das im Folgenden als **Benutzeraufgabenprofil** bezeichnet wird. Durch dieses UML-Profil werden die spezifizierten Metamodellelemente als Erweiterungen bestehender Metaklassen der UML in Form von Stereotypen umgesetzt. Diese Stereotypen können dann wiederum in modernen Entwicklungsumgebungen wie beispielsweise dem Rational Software Architect von IBM [IBM-RSA] zur Spezifikation von Benutzeraufgabenmodellen verwendet werden. Um die Verwendung der Stereotypen, die durch das Benutzeraufgabenprofil neu bereitgestellt werden, in einem modellgetriebenen Entwicklungsvorgehen zu vereinfachen, gestattet die UML eine Anpassung ihrer konkreten Syntax durch die Spezifikation eigener grafischer Symbole für Stereotypen [OMG-UML2-Super]. Das in Abbildung 39 dargestellte Benutzeraufgabenprofil deutet daher mögliche grafische Symbole[3] für die Stereotypen an, durch die eine vereinfachte Verwendung und Identifizierung der neuen Stereotypen in den entsprechenden Diagrammen möglich wird.

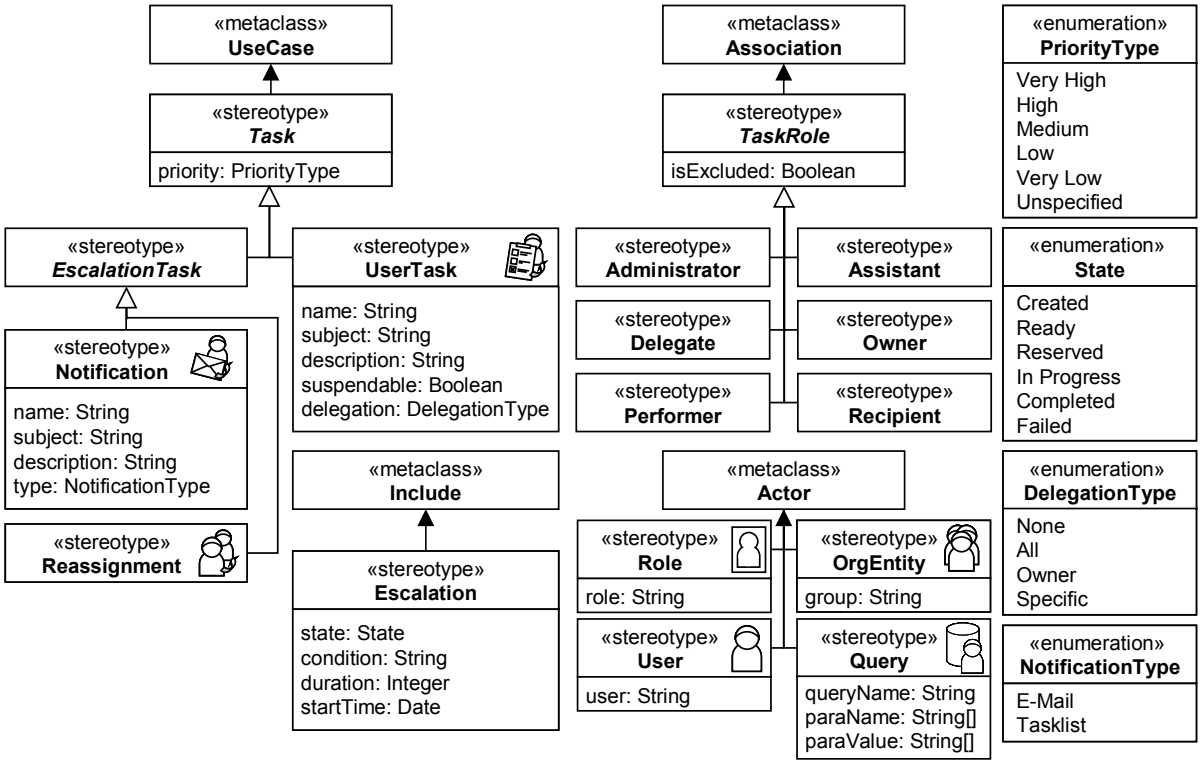

Abbildung 39: Stereotypen des Benutzeraufgabenprofils

Bei der Umsetzung des Metamodells auf das Benutzeraufgabenprofil wurde bewusst eine Vereinfachung der zur Verfügung stehenden Modellelemente auf Kosten der Wiederverwendbarkeit vorgenommen. Das Metamodell sieht zwischen einer Benutzeraufgabe, einem Indikator, einer Eskalation und einer Eskalationsaufgabe jeweils eine Kardinalität größer oder gleich eins vor. Demnach können zu einer Benutzeraufgabe n Indikatoren, zu einem Indikator m Eskalationen und zu einer Eskalation j

[3] Für eine zur UML konformen Spezifikation der grafischen Repräsentationen müsste jeder Stereotyp über eine Assoziation „icon“ mit einem Modellelement Image verbunden werden. Aus Gründen der Übersichtlichkeit wurde daher eine vereinfachte grafische Darstellung des Benutzeraufgabenprofils gewählt.

Eskalationsaufgaben spezifiziert werden (m, n, j seien aus der Menge der natürlichen Zahlen ohne 0 entnommen). Durch diese Mehrstufigkeit lassen sich beispielsweise Eskalationsketten spezifizieren, die in unterschiedlichsten Kontexten wiederverwendet werden können.

Im betrachteten Szenario kann jedoch davon ausgegangen werden, dass eine Benutzeraufgabe mit einer überschaubaren Anzahl an Indikatoren und Eskalationen versehen wird und die Mehrstufigkeit des Metamodells zur Verringerung der Komplexität des Benutzeraufgabenmodells reduziert werden kann. Dementsprechend fasst der als Erweiterung der UML-Metaklasse `Include` spezifizierte Stereotyp `Escalation` die beiden Metamodellelemente `Escalation` und `Indicator` im Benutzeraufgabenprofil zusammen.

Wird in einem weiteren Szenario eine flexiblere und somit entkoppelte Spezifikation von Eskalationen und Indikatoren benötigt, so müssen hierzu lediglich zwei getrennte Stereotypen für eine Eskalation und einen Indikator zur Verfügung gestellt werden. Abbildung 40 illustriert die eben beschriebene Vereinfachung. Bei der Umsetzung des Metamodells zur Spezifikation von Benutzeraufgaben als ein UML-Profil kommen vier Metaklassen des UML-Metamodells zum Einsatz, die in den folgenden Abschnitten eingeführt werden. Ferner müssen bei der Umsetzung die Einschränkungen der Metamodellelemente des Benutzeraufgabenmodells auf das Metamodell der UML angepasst werden. Die Anpassung der im gesamten Abschnitt 4.3.1 eingeführten Einschränkungen wird in den folgenden Abschnitten beispielhaft aufgezeigt.

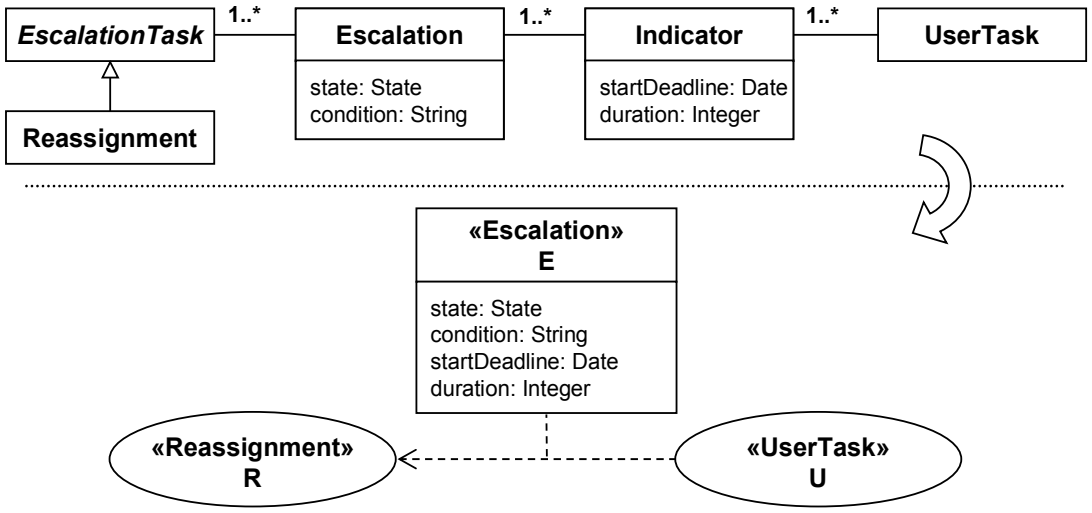

Abbildung 40: Vereinfachte Zuordnung von Indikatoren und Eskalationen

4.3.2.1 Erweiterung der Metaklasse Anwendungsfall

Nach [OMG-UML2-Super] ist ein Anwendungsfall (`Use Case`) definiert als:

> *"[…] a means for specifying required usages of a system. Typically, they are used to capture the requirements of a system, that is, what a system is supposed to do."*

Entsprechend verwendet das Benutzeraufgabenprofil diese Metaklasse, um als deren Erweiterungen die abstrakten Stereotypen Aufgabe (`Task`) und Eskalationsaufgabe (`EscalationTask`) sowie die konkreten Stereotypen Benutzeraufgabe (`UserTask`), Benachrichtigung (`Notification`) und Neuzuweisung (`Reassignment`) festzulegen, die gegenüber dem Benutzer alle ein sichtbares Ergebnis liefern. Die im Benutzeraufgabenmetamodell spezifizierte statische Semantik des Metamo-

dellelements Benutzeraufgabe wird durch die folgenden OCL-Ausdrücke für den Stereotyp Benutzeraufgabe als Erweiterung des UML-Anwendungsfalls formalisiert.

```
context UserTask
-- A UserTask has to have exactly one Administrator
inv: self.ownedAttribute -> select(a | (a.association -> notEmpty()
       and a.opposite.class.oclIsTypeOf(User)
       and a.association.OclIsTypeOf(Administrator)) -> size() = 1

-- A UserTask has to have at least one Owner
inv: self.ownedAttribute -> exists(a | (a.association -> notEmpty()
       and a.opposite.class.oclIsKindOf(Actor)
       and a.association.OclIsTypeOf(Owner))

-- A UserTask may not have a Recipient
inv: self.ownedAttribute -> not exists(a | (a.association -> notEmpty()
       and a.opposite.class.oclIsKindOf(Actor)
       and a.association.OclIsTypeOf(Recipient))

-- If delegation-property is set, UserTask has to have a Delegate
inv: self.delegation = DelegationType::specific implies
         self.ownedAttribute -> select(a | (a.association -> notEmpty()
           and a.opposite.class.oclIsTypeOf(User)
           and a.association.OclIsTypeOf(Delegate)) -> size() = 1
```

4.3.2.2 Erweiterung der Metaklasse Assoziation

Die UML-Metaklasse Assoziation (`Association`) kann dazu verwendet werden, Beziehungen zwischen unterschiedlichen Modellelementen herzustellen. Diese Metaklasse eignet sich daher bestens, um eine Aufgabenrolle (`TaskRole`), die ein Akteur gegenüber einer Aufgabe einnimmt, präzise spezifizieren zu können. Dementsprechend sieht das Benutzeraufgabenprofil die verschiedenen Aufgabenrollen des Benutzeraufgabenmetamodells als Spezialisierung des Stereotyps `TaskRole` vor, welcher selbst wiederum die UML-Metaklasse `Association` erweitert. Auch für diese Stereotypen muss die statische Semantik des Benutzeraufgabenmetamodells übertragen und an die statische Semantik des UML-Metamodells angepasst werden.

```
context Notification
-- Each Notification has to have at least one Recipient
inv: self.ownedAttribute -> select(a | (a.association -> notEmpty()
       and a.opposite.class.oclIsKindOf(Actor)
       and a.association.OclIsTypeOf(Recipient)) -> size() > 0

-- No other TaskRole but a Recipient is allowed with a Notification
inv: self.ownedAttribute -> forAll(a | (a.association -> notEmpty()
       and a.opposite.class.oclIsKindOf(Actor) implies
       a.association.OclIsTypeOf(Recipient))
```

4.3.2.3 Erweiterung der Metaklasse Akteur

Die UML-Metaklasse des Akteurs (`Actor`) wird zur Spezifikation einer Entität verwendet, die mit einem Gegenstand interagiert, selbst aber nicht Teil dieses Gegenstandes ist. Die UML-*Superstructure* sieht hier explizit menschliche Akteure vor. Dementsprechend verwendet das Benutzeraufgabenprofil

die Metaklasse `Actor` und erweitert diese entsprechend dem Benutzeraufgabenmetamodell um die drei benötigten Stereotypen Rolle (`Role`), Benutzer (`User`) und Organisationseinheit (`OrganizationalUnit`).

4.3.2.4 Erweiterung der Metaklasse „Include"

Abschließend müssen noch die beiden Metamodellelemente `Escalation` und `Indicator` auf das Benutzeraufgabenprofil abgebildet werden. Bei dieser Abbildung wird die in Abbildung 40 dargestellte vereinfachte Zuordnung von Eskalationen und Indikatoren zu Benutzeraufgaben verwendet. Für diesen Fall bietet sich die UML-Metaklasse `Include` an, die in der UML-*Superstructure* wie folgt definiert wird [OMG-UML2-Super]:

> *"An include relationship defines that a use case contains the behavior defined in another use case."*

Diese Beziehung trifft genau für das im Metamodell vorhandene Konzept der Eskalation (`Escalation`) zu. Eine Eskalation ruft ein weiteres Verhalten wie beispielsweise die Neuzuweisung auf und entspricht damit genau der Definition der *Include*-Beziehung der UML. Es existiert folglich nur ein Stereotyp, der die Metaklasse `Include` erweitert.

4.3.3 Anwendung des Benutzeraufgabenprofils

Mit dem eingeführten UML-Profil für Benutzeraufgaben kann eine Benutzeraufgabe nun im Detail spezifiziert werden. Hierzu wurde bewusst ein vom ursprünglichen Workflow-Modell separiertes und eigenständiges Modell gewählt, da die Spezifikation von Benutzeraufgaben eine konkrete Sichtweise auf einen Workflow verfeinert und daher nicht für alle an einem modellgetriebenen Entwicklungsvorgehen beteiligten Entwicklungsrollen relevant ist. Durch ein eigenständiges Modell können entsprechende Fachentwickler in das Entwicklungsvorgehen integriert werden, die sich ausschließlich mit der Spezifikation der durch das Benutzeraufgabenmodell adressierten Ressourcenperspektive eines Workflows befassen. Dementsprechend kann auf dem in Abbildung 34 (Seite 89) illustrierten Beispiel eines Workflow-Modells aufbauend für Benutzeraktion C_1 der Geschäftsrolle C ein entsprechendes Benutzeraufgabenmodell entwickelt werden.

Abbildung 41 stellt dieses beispielhaft als UML-Anwendungsfalldiagramm dar, das um die Stereotypen des in Abbildung 39 gezeigten Benutzeraufgabenprofils erweitert wurde. Die verwendete grafische Repräsentation der Modellelemente ist hierbei in modernen Entwicklungsumgebungen frei wählbar und kann daher an die Bedürfnisse der Entwickler, die zur Erstellung der Benutzeraufgabenmodelle herangezogen werden, und an die Gegebenheiten des übergeordneten modellgetriebenen Entwicklungsvorgehens angepasst werden.

Das Metamodell für Benutzeraufgaben gestattet eine plattformunabhängige Spezifikation von Benutzeraufgaben durch Modelle, die anschließend durch Modelltransformationen auf plattformspezifischen Quellcode für unterschiedlichste Plattformen abgebildet werden können. Die automatisierte Abbildung der Modelle auf ausführbaren Quellcode wird durch diese Arbeit ebenfalls verfolgt. Aus diesem Grund greift Kapitel 5 das Benutzeraufgabenprofil wieder auf und führt die in diesem Abschnitt nicht näher beleuchteten Modelltransformationen zwischen Workflow-Modell, plattformunabhängigem Benutzeraufgabenmodell und plattformspezifischem Benutzeraufgabenmodell ein.

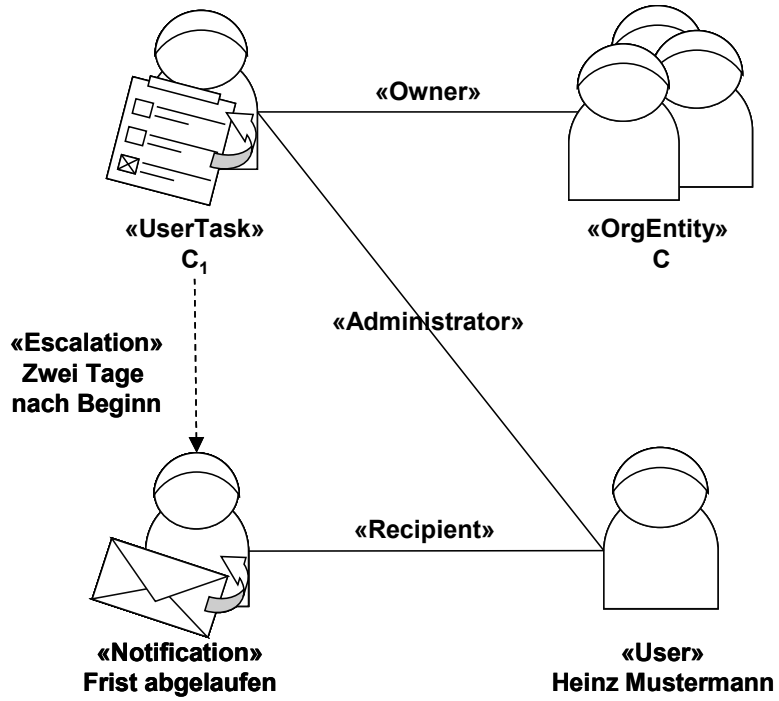

Abbildung 41: Spezifikation einer Benutzeraufgabe als UML-Anwendungsfalldiagramm

Neben der Anforderung, die Interaktion eines Menschen als Benutzer eines Anwendungssystems überwachen zu können, benötigt der Mensch eine Benutzerschnittstelle, über die er mit einem Anwendungssystem interagieren kann. Die Benutzerschnittstelle muss daher ebenfalls integriert in einem modellgetriebenen Entwicklungsvorgehen berücksichtigt werden. Der folgende Abschnitt befasst sich daher mit der Spezifikation einer Benutzerschnittstelle durch Modelle.

4.4 Spezifikation der Benutzerschnittstellen

Um die Interaktion eines Menschen mit einem Anwendungssystem zu ermöglichen, muss dieses eine geeignete Schnittstelle für den Menschen bereitstellen. Da ein Mensch das Anwendungssystem über diese Schnittstelle nutzt, wird eine derartige Schnittstelle als Benutzerschnittstelle bezeichnet. Aus der Vielfalt heute vorhandener Benutzerschnittstellen betrachtet diese Arbeit die Menge der grafischen Benutzerschnittstellen näher und bezeichnet diese der Einfachheit halber kurz als Benutzerschnittstelle (vgl. dazu Prämisse P1). Als primärer Berührungspunkt des Benutzers mit dem Anwendungssystem kann ein Modell oder auch ein Prototyp einer Benutzerschnittstelle neben den Workflow-Modellen wesentlich zum Gesamtverständnis des Anwendungssystems beitragen und damit als Diskussionsgrundlage für alle am Entwicklungsprozess beteiligten Entwicklungsrollen dienen [CN05a]. Ziel muss es daher sein, dem Kunden als späterem Benutzer des Anwendungssystems sehr früh im Softwareentwicklungsprozess eine erste funktionale Version der Benutzerschnittstelle präsentieren zu können, anhand derer die korrekte Umsetzung der Anforderungen des Kunden verifiziert werden kann. Gerade in frühen Stadien eines Projektes zur Softwareentwicklung ergeben sich anhand einer Diskussion auf Basis der Workflow-Modelle und der Benutzerschnittstelle immer wieder Änderungen, die an allen Softwareartefakten des zu entwickelnden Anwendungssystems gespiegelt werden müssen. Folglich müssen Modelle der Workflows, aber auch der Benutzerschnittstellen so spezifiziert werden, dass notwendige Änderungen nachvollziehbar und rasch durchführbar sind und somit die geforderte Flexibilität in der Abbildung von Geschäftsprozessen auf die IT-Unterstützung erreicht werden kann.

Bestehende Ansätze messen der Benutzerschnittstelle zwar einen zentralen Stellenwert bei, integrieren diese aber entweder gar nicht oder nur sehr lose in das übergeordnete Entwicklungsvorgehen (vgl. dazu Kapitel 3.2). Häufig kommen hier Methoden wie beispielsweise das *Rapid Prototyping* [BB+96], *Storyboarding* [KK+01] oder *Sketching* [CN05b, BJ04, Co03] zum Einsatz, die ebenfalls das Ziel verfolgen, dem Kunden als späterem Benutzer des Anwendungssystems möglichst rasch eine Benutzerschnittstelle zu präsentieren, um so dessen Anforderungen verifizieren zu können. Der in diese Methoden investierte Aufwand führt zwar zu einem schnellen Prototyp der Benutzerschnittstelle, jedoch ist dieser nicht funktional und dient alleine der Verifikation der Anforderungen des Kunden sowie als Dokumentation für die eigentliche, sich anschließende Entwicklung der Benutzerschnittstelle. Die insbesondere in frühen Entwicklungsphasen häufig auftretenden Änderungen der Anforderungen resultieren somit in einer Änderung des Prototyps, einer erneuten Abstimmung mit dem Kunden anhand des Prototyps und zusätzlich in der Umsetzung der Änderungen auf die funktionale Benutzerschnittstelle. Dieses Vorgehen ist folglich nicht nur aufwandsintensiv, es führt auch zu einer geringeren Softwarequalität, da der entwickelte Prototyp und die eigentliche Benutzerschnittstelle nur durch einen erhöhten Entwicklungsaufwand konsistent zueinander gehalten werden können [FR07]. Daher muss es das Ziel sein, die Entwicklung der Benutzerschnittstelle so in den Softwareentwicklungsprozess zu integrieren, dass der investierte Aufwand einerseits eine Grundlage für die Diskussion mit dem Kunden liefert und andererseits unmittelbar zur Erzeugung von Quellcode herangezogen werden kann [Bi06].

Ein weiterer Vorteil des verfolgten modellgetriebenen Vorgehens liegt in der Möglichkeit, der heute im Bereich der Benutzerschnittstellen besonders ausgeprägten Heterogenität der IT-Systeme begegnen zu können. Durch die Abstraktion von einer konkreten Zielplattform können im Rahmen der modellgetriebenen Entwicklung Anforderungen an eine Benutzerschnittstelle zunächst ohne technologische Details im Sinne eines Fachkonzeptes spezifiziert und erst in einem zweiten Schritt für eine konkrete Zielplattform, wie etwa ein Entwicklungsrahmenwerk für Benutzerschnittstellen, ausgeprägt werden. Die plattformunabhängig erfassten Fachkonzepte behalten dabei ihre Gültigkeit und können je nach Bedarf für unterschiedliche Zielplattformen als Ausgangsmodelle verwendet werden. Der benötigte Aufwand zur Erzeugung der Benutzerschnittstellen wird somit reduziert.

Wie im Überblick der Beiträge dieses Kapitels in Abbildung 29 (Seite 80) angedeutet und in Abschnitt 4.2 ausgeführt, kann aus einem Workflow-Modell bereits extrahiert werden, zu welchen Zeitpunkten ein Benutzer in einer Benutzeraktion tätig werden muss und dementsprechend eine Benutzerschnittstelle benötigt. Über die Ein- und Ausgabeparameter einer Benutzeraktion kann ferner bereits im Workflow-Modell festgehalten werden, welche Geschäftsobjekte durch einen Benutzer in einer Benutzeraktion bearbeitet werden sollen. Da der Kunde als späterer Nutzer des Anwendungssystems zusätzlich eine Vorstellung darüber besitzt, wie er mit dem Anwendungssystem interagieren und welche Daten er ändern oder neu eingeben möchte, müssen diese Angaben für die modellgetriebene Entwicklung der Benutzerschnittstelle verfügbar gemacht werden. Gelingt es daher, diese Informationen in einem plattformunabhängigen Modell der Benutzerschnittstelle integriert im modellgetriebenen Softwareentwicklungsvorgehen zu spezifizieren, so kann eine automatische Transformation dieses Modells bis auf Quellcode der Benutzerschnittstelle erreicht werden.

4.4.1 Entwurf der Metamodelle

Durch die in Abschnitt 4.2 eingeführten Auszeichnungen der einzelnen Aktionen in einem Workflow-Modell wird auf einfache Art und Weise ersichtlich, zu welchem Zeitpunkt der Ausführung einer Workflow-Instanz der Benutzer mit dem Anwendungssystem über eine Benutzerschnittstelle in

Interaktion treten muss. Jede dieser Benutzerinteraktionen lässt sich auf das einfache Prinzip der Manipulation von Daten zurückführen [SP03]. Gleich, ob diese Manipulation durch Neueingabe von Daten, das Editieren bestehender Daten oder durch die Bestätigung einer Entscheidung erfolgt, sie überführt das Anwendungssystem in einen neuen Zustand. Folglich ist für die Benutzerschnittstelle wesentlich, welche Menge von Daten durch den Benutzer manipuliert werden soll. Hieraus leitet sich auch der in Abschnitt 2.3.1 angeführte enge Bezug zwischen einem Modell der Benutzerschnittstelle und einem Domänenmodell des Anwendungssystems her.

Wie der Autor in [LS+08] und [LS+07b] publiziert hat, verfolgt diese Arbeit bei der Entwicklung der Benutzerschnittstelle ein Vorgehen, das den Kunden als Benutzer des Anwendungssystems in den Mittelpunkt rückt. Bestehende Entwicklungsvorgehen fokussieren häufig zunächst alle Entitäten, die im Kontext des Workflows, ja sogar im Kontext des gesamten Anwendungssystems auftreten, und streben an, diese in einem ersten Entwicklungsschritt vollständig in einem Domänenmodell zu erfassen. Für den Benutzer des Anwendungssystems sind viele dieser Entitäten und die damit verbundenen Details des Domänenmodells jedoch irrelevant. Beispielsweise ist die interne Repräsentation einer Entität „Studierender“ für den Benutzer nur von geringer Bedeutung. Viel eher interessiert diesen, welche Eigenschaften der Entität „Studierender“ er während der Ausführung einer Workflow-Instanz bearbeiten kann. Somit stehen für den Benutzer des Anwendungssystems lediglich diejenigen Eigenschaften der Geschäftsobjekte eines Workflows im Vordergrund, auf die er direkten Einfluss hat. Werden diese Eigenschaften der Geschäftsobjekte direkt in der Benutzerschnittstelle reflektiert, so kann eine korrekte Umsetzung der Anforderungen und Vorstellungen des Kunden durch das Anwendungssystem frühzeitig verifiziert werden.

Das in Abbildung 34 (Seite 89) illustrierte erweiterte Modell des abstrakten Beispiels eines Geschäftsprozesses dient auch als Ausgangsmodell der Entwicklung des Benutzerschnittstellenmodells. Es umfasst Ein- und Ausgabepins (engl. *Input / Output Pins*), welche die Ein- und Ausgabeparameter der einzelnen Aktion festhalten. Wie bereits in Abschnitt 4.2 diskutiert wurde, können auf diese Weise abstrakt die Geschäftsobjekte modelliert werden, die entlang des Kontrollflusses eines Workflows in den verschiedenen Aktionen bearbeitet werden müssen. Für Benutzeraktionen bilden diese abstrakt erfassten Geschäftsobjekte gleichzeitig die Grundlage für das zu entwickelnde Benutzerschnittstellenmodell. Durch die Verwendung der *Input* und *Output Pins* einer Aktion kann der Bezug dieser Geschäftsobjekte zu einer zugeordneten Benutzerschnittstelle konkretisiert werden. Abbildung 42 zeigt drei unterschiedliche Möglichkeiten zur Spezifikation der Verwendung eines Geschäftsobjektes in einer Benutzeraktion auf, die, wie im Anschluss erläutert, unterschiedliche Auswirkungen auf das korrelierte Benutzerschnittstellenmodell haben.

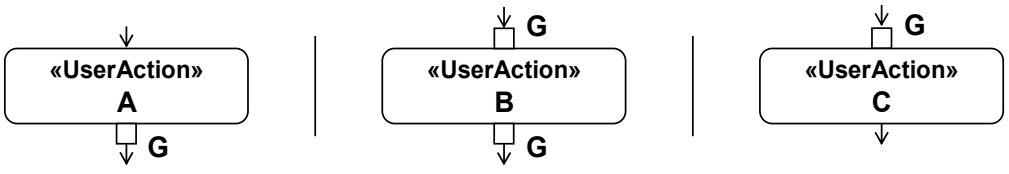

Abbildung 42: Geschäftsobjekte als Grundlage der Benutzerschnittstelle

Geht Geschäftsobjekt G, wie im linken Beispiel gezeigt, aus einer Benutzeraktion A aus, vorher aber nicht in diese ein, so wird das Geschäftsobjekt G in dieser Benutzeraktion neu instanziiert und durch den Benutzer mit Werten belegt. Dementsprechend muss dem Benutzer eine geeignete Benutzerschnittstelle zur Verfügung gestellt werden, in der er die Werte der Geschäftsobjektinstanz eingeben kann. Als ein typisches Beispiel sei das Anlegen eines neuen Studierenden für ein Studierendenver-

zeichnis durch eine Fachkraft genannt. Geht ein Geschäftsobjekt G, wie am mittleren Beispiel zu sehen, in eine Benutzeraktion B ein und auch wieder aus, so ist zum Ausführungszeitpunkt der Benutzeraktion eine Instanz dieses Geschäftsobjektes bereits vorhanden und muss für den Benutzer über eine geeignete Benutzerschnittstelle visualisiert werden. Da die Geschäftsobjektinstanz die Benutzeraktion auch wieder verlässt, muss es dem Benutzer zusätzlich ermöglicht werden, die Werte dieser Instanz zu bearbeiten. Die Aktualisierung der Daten eines Studierenden ist eine typische Ausprägung für das mittlere Beispiel. Das rechte Beispiel in Abbildung 42 zeigt die dritte Spezifikationsmöglichkeit auf. Hier geht eine Geschäftsobjektinstanz G in eine Benutzeraktion C ein, nicht aber aus dieser wieder aus. Folglich darf dieses Geschäftsobjekt in dieser Benutzeraktion lediglich visualisiert, nicht aber zur Bearbeitung freigegeben werden. Diese drei Spezifikationsmöglichkeiten können über verschiedene Geschäftsobjekte hinweg beliebig kombiniert und somit die Zugriffsrechte einer Geschäftsrolle auf die in einer Benutzeraktion zu bearbeitenden Geschäftsobjekte festgelegt werden.

Für ein Geschäftsprozessmodell bzw. ein Workflow-Modell ist die Granularität eines Geschäftsobjektes als Einheit zur Spezifikation des Datenflusses ausreichend. Zur Ableitung einer Benutzerschnittstelle ist es jedoch notwendig, ein Geschäftsobjekt im Detail zu erfassen, um so die Eigenschaften des Geschäftsobjektes, die den Menschen als Benutzer betreffen und daher für ihn visualisiert werden müssen, feingranular spezifizieren zu können. Um die in Abbildung 42 aufgezeigten unterschiedlichen Spezifikationsmöglichkeiten der Verwendung eines Geschäftsobjektes auch detailliert für einzelne Eigenschaften eines Geschäftsobjekts durchführen zu können, müssen die in Form von Eingabe- oder Ausgabepins modellierten Geschäftsobjekte einer Benutzeraktion zunächst auf ein zur weiteren Verfeinerung geeignetes Modell abgebildet werden. Dieses Modell muss Möglichkeiten bieten, für jede einzelne Eigenschaft einer Geschäftsobjektinstanz zu bestimmen, ob diese in einer Benutzeraktion neu mit einem Wert belegt wird, ob die Eigenschaft bereits mit einem Wert belegt ist und dieser bearbeitet werden soll oder ob der bereits vorhandene Wert der Eigenschaft lediglich angezeigt werden soll. Da diese Unterscheidung für alle Eigenschaften aller Geschäftsobjekte des Anwendungssystems zu treffen ist, führt diese Arbeit in Anlehnung an [HA+97] und [NT+99] ein **benutzerzentrisches Domänenmodell** ein, das die Perspektive des Benutzers auf das Domänenmodell eines Anwendungssystems in den Vordergrund stellt und die benötigte feingranulare Spezifikation der einzelnen Eigenschaften der Geschäftsobjekte ermöglicht. Abbildung 43 zeigt als Erweiterung des in Abbildung 31 (Seite 83) dargestellten Metamodells für Geschäftsprozesse die Metamodellelemente `Property`, `UserInterface`, `UserInputInterface` und `UserOut-putInterface` auf, deren Sinn und Zweck im Folgenden erläutert werden.

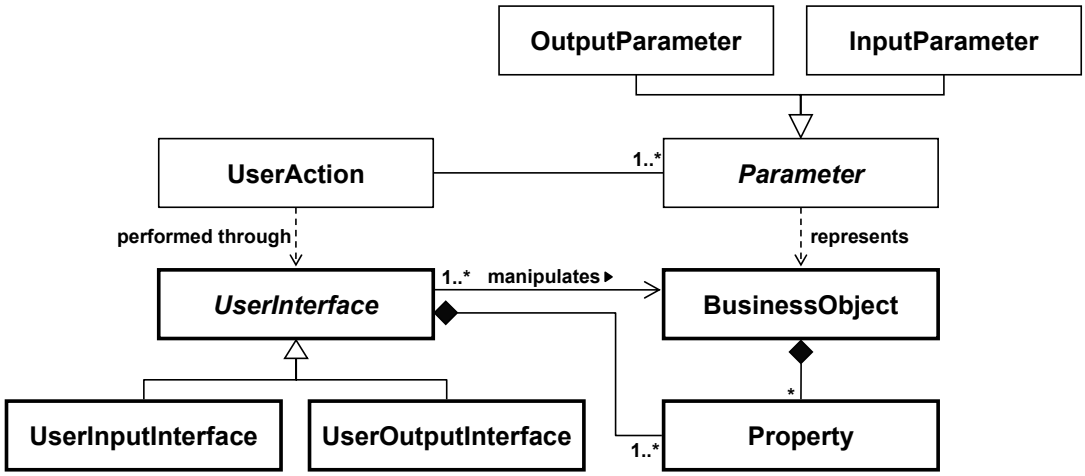

Abbildung 43: Konzeptionelles Metamodell des benutzerzentrischen Domänenmodells

Das in Abbildung 43 illustrierte konzeptionelle Metamodell ordnet jeder Benutzeraktion (`UserAction`) eines Geschäftsprozessmodells mindestens ein abstraktes Metamodellelement Benutzerschnittstelle (`UserInterface`) zu. Hierin wird die Bedingung formalisiert, dass eine Benutzeraktion immer eine Benutzerschnittstelle benötigt, um eine Interaktion des Menschen innerhalb einer Benutzeraktion zu ermöglichen. Über eine Benutzerschnittstelle kann ein Benutzer auf ein Geschäftsobjekt (`BusinessObject`) zugreifen, weshalb die Benutzerschnittstelle die Manipulation des Geschäftsobjektes ermöglicht (`manipulates`). Um diesen Bezug zwischen Geschäftsobjekt und Benutzerschnittstelle zu verfeinern, können Eigenschaften (`Property`) eines Geschäftsobjektes und einer Benutzerschnittstelle modelliert werden. Ein Geschäftsobjekt verfügt mindestens über eine Eigenschaft, die das Geschäftsobjekt näher beschreibt. Der Zugriff des Benutzers auf die Eigenschaften eines Geschäftsobjektes erfolgt über die Benutzerschnittstelle, die dementsprechend ebenfalls über diese Eigenschaften verfügen muss. Um die Unterscheidung, ob dem Benutzer der Zugriff auf eine Eigenschaft nur lesend, nur schreibend oder lesend und schreibend gewährt wird, kann durch die beiden Metamodellelemente Benutzereingabeschnittstelle (`UserInputInterface`) und Benutzerausgabeschnittstelle (`UserOutputInterface`) spezifiziert werden. Diese beiden Metamodellelemente gestatten somit eine feingranulare Ausprägung der in Abbildung 42 angedeuteten drei Spezifikationsmöglichkeiten für jede einzelne Eigenschaft eines Geschäftsobjektes. Die folgenden Abschnitte beschreiben die Metamodellelemente des benutzerzentrischen Domänenmodells im Detail.

4.4.1.1 Geschäftsobjekt

Ein Geschäftsobjekt (`BusinessObject`) repräsentiert in Anlehnung an [Oe06] eine Entität, die in einem Geschäftsprozess durch eine menschliche oder technische Ressource bearbeitet werden muss. Durch die Wahl eines Geschäftsprozesses als Ausgangsmodell des modellgetriebenen Entwicklungsvorgehens stellen die Geschäftsobjekte somit die zentralen Entitäten im Kontext dieser Arbeit dar. Sie dienen beispielsweise als Grundlage für die Spezifikation einer Benutzerschnittstelle, da in jeder Benutzeraktion ein Geschäftsobjekt durch den Benutzer bearbeitet werden muss. Müsste in einer Benutzeraktion kein Geschäftsobjekt bearbeitet werden, so wäre folglich auch keine Aktion des Benutzers notwendig. Dementsprechend wird in jeder Benutzeraktion eine Eingabe des Benutzers erwartet, die eine Instanz eines Geschäftsobjekts manipuliert. Selbst das Treffen einer Entscheidung wird durch diese Arbeit als Manipulation einer Geschäftsobjektinstanz betrachtet, da diese Entscheidung die Instanz in einen neuen Zustand überführt. Ein Zustand kann letztendlich als einfache Eigenschaft (`Property`) des Geschäftsobjektes betrachtet werden, deren Wert sich beim Fällen einer Entscheidung für die Instanz des Geschäftsobjekts, auf welche sich die Entscheidung bezieht, ändert. Als Beispiel sei die Zulassung eines Studierenden zu einer Prüfung erwähnt. Genehmigt der Dozent die Prüfung, so wechselt dementsprechend die Zustandseigenschaft dieser Instanz des Geschäftsobjekts Prüfung ihren Wert etwa von „nicht entschieden“ auf „genehmigt“.

4.4.1.2 Benutzerschnittstelle

Die Benutzerschnittstelle (`UserInterface`) repräsentiert abstrakt die in einer Benutzeraktion zur Bearbeitung eines Geschäftsobjekts notwendige Schnittstelle für den Benutzer. Durch die möglichen Kombinationen der beiden im Folgenden beschriebenen Spezialisierungen der Benutzerschnittstelle lassen sich die in Abbildung 42 identifizierten Spezifikationsmöglichkeiten (nur lesend, nur schreibend oder lesend und schreibend) modellieren. Die Bezeichnungen dieser beiden Spezialisierungen als Benutzereingabeschnittstelle (`UserInputInterface`) und Benutzerausgabeschnittstelle

(`UserOutputInterface`) werden bewusst komplementär zu den in Abschnitt 4.2.1.4 definierten Ein- und Ausgabeparametern gewählt.

Die Ausgabeparameter, die nach der Bearbeitung einer Benutzeraktion zur Verfügung stehen müssen, werden durch einen Benutzer während der Ausführung der Benutzeraktion eingegeben. Die Eingabeparameter hingegen, die einer Benutzeraktion zur Verfügung gestellt werden, müssen für den Benutzer als Bearbeiter über die Benutzerschnittstelle visualisiert und ausgegeben werden. Hieraus erklärt sich die im Folgenden detailliert beschriebene Zuordnung einer Benutzereingabeschnittstelle zu einem Ausgabeparameter und die umgekehrte Zuordnung der Benutzerausgabeschnittstelle zu einem Eingabeparameter.

Benutzereingabeschnittstelle

Die Benutzereingabeschnittstelle (`UserInputInterface`) ist eine Spezialisierung des Metamodellelements Benutzerschnittstelle. Sie dient dazu, alle Eingaben, die durch einen Benutzer in einer Benutzeraktion erwartet werden, näher zu spezifizieren und umfasst damit alle Eigenschaften der Geschäftsobjekte, die durch einen Benutzer in einer Benutzeraktion manipuliert werden können. Wird eine Eigenschaft eines Geschäftsobjektes in einer Benutzereingabeschnittstelle, nicht aber in einer Benutzerausgabeschnittstelle spezifiziert, so wird diese Eigenschaft einer Geschäftsobjektinstanz durch den Benutzer in der Benutzeraktion neu mit einem Wert belegt.

Benutzerausgabeschnittstelle

Als weitere Spezialisierung der Benutzerschnittstelle umfasst eine Benutzerausgabeschnittstelle (`UserOutputInterface`) die Eigenschaften eines Geschäftsobjektes, die für den Benutzer visualisiert werden müssen.

4.4.1.3 Eigenschaft

Eine Eigenschaft (`Property`) beschreibt einen Gegenstand näher. Im Kontext dieser Arbeit werden Eigenschaften dazu verwendet, Geschäftsobjekte und Benutzerschnittstellen näher zu beschreiben. Beispiele für Eigenschaften eines Geschäftsobjekts „Studierender“ sind „Name“, „Vorname“ oder „Adresse“. Eine Benutzerschnittstelle kann ebenfalls über Eigenschaften verfügen. Der Bezug zwischen den Eigenschaften eines Geschäftsobjekts und einer Benutzerschnittstelle entsteht aus der Herangehensweise, dass eine Eigenschaft eines Geschäftsobjekts, sofern sie für den Benutzer von Bedeutung ist, in der Benutzerschnittstelle zur Anzeige gebracht werden muss. Die für den Benutzer relevanten Eigenschaften eines Geschäftsobjekts werden daher zu anzuzeigenden Eigenschaften einer Benutzerschnittstelle.

Der durch den Autor in [LH+09] publizierte Zusammenhang zwischen Benutzeraktionen, Parametern, Geschäftsobjekten, Eigenschaften und Benutzerschnittstellen wird in Abschnitt 4.4.2 anhand eines Beispiels weiter verdeutlicht. Durch den Fokus des benutzerzentrischen Domänenmodells auf den Kunden als potenzieller Benutzer des Anwendungssystems umfasst dieses Modell im Vergleich zu einem allgemeinen Domänenmodell nur die für den Kunden wichtigen Aspekte und ist daher insbesondere für eine frühzeitige Verifikation der Anforderungen des Kunden hinsichtlich seiner Interaktion mit dem System geeignet. Zusätzlich gestattet das benutzerzentrische Domänenmodell eine klare Trennung zwischen Kundensicht und Systemsicht im Entwicklungsvorgehen. Sind alle Anforderungen des Kunden in Form von Modellen erfasst, kann im Entwicklungsvorgehen eine Systemsicht eingenommen und auch das benutzerzentrische Domänenmodell zu einem allgemeinen Domänenmodell

vervollständigt werden. Dieses umfasst alle Entitäten, deren Bearbeitung durch die IT-Systeme der IT-Unterstützung ermöglicht werden muss.

Die bisher eingeführten Metamodellelemente erlauben eine inhaltliche Spezifikation der Benutzerschnittstelle anhand der Geschäftsobjekte, die durch eine Benutzerschnittstelle bearbeitet werden sollen. Diese durch das benutzerzentrische Domänenmodell erfasste inhaltliche Perspektive muss für die Ableitung einer konkreten Benutzerschnittstelle weiter verfeinert werden. Hierzu müssen die modellierten Eigenschaften der Benutzerausgabe- und Benutzereingabeschnittstellen auf weitere Modellelemente abgebildet werden, die eine plattformunabhängige Modellierung der Struktur einer Benutzerschnittstelle ermöglichen. Die Modellierung der Struktur einer Benutzerschnittstelle dient als Grundlage für die im weiteren Verlauf der Arbeit verfolgte Ableitung einer funktionalen Benutzerschnittstelle. Hierzu fasst diese Arbeit die benötigten Modellelemente in einem eigenen als **Strukturmodell** bezeichneten Modell zusammen, dessen Metamodell und Modellelemente im Folgenden eingeführt werden.

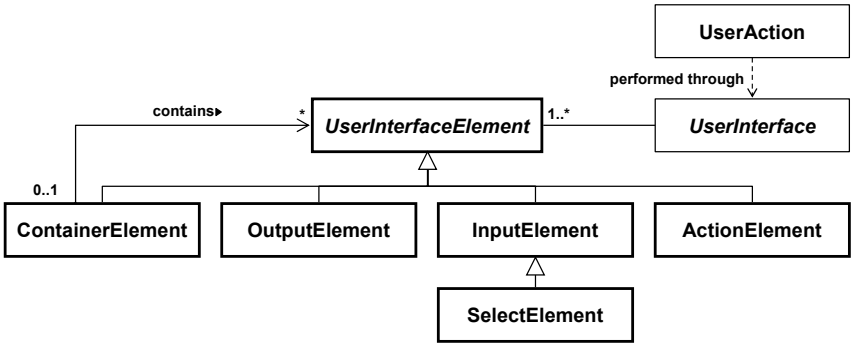

Abbildung 44: Entwurf des konzeptionellen Metamodells des Strukturmodells

Die statische Struktur eine Benutzerschnittstelle lässt sich, wie unter anderem in [BV96] oder [Ko01] angeführt, auf wenige Elemente reduzieren. Dementsprechend zeigt Abbildung 44 den Entwurf des konzeptionellen Metamodells zur Spezifikation eines Strukturmodells einer Benutzerschnittstelle und deutet den Zusammenhang zu den bisher betrachteten Metamodellelementen an.

Eine Benutzerschnittstelle (`UserInterface`), mithilfe derer ein Benutzer eine Benutzeraktion (`UserAction`) ausführen kann, besteht aus mindestens einem Benutzerschnittstellenelement (`UserInterfaceElement`). Das zunächst abstrakt betrachtete Benutzerschnittstellenelement wird durch vier konkrete Metamodellelemente spezialisiert, die sich an dem strukturellen Aufbau einer Benutzerschnittstelle orientieren. Ein Containerelement (`ContainerElement`) dient zur Aufnahme weiterer Benutzerschnittstellenelemente und bildet daher die oberste strukturelle Ebene einer Benutzerschnittstelle. Alle weiteren Benutzerschnittstellenelemente werden in ein Containerelement eingehängt. Ein Ausgabeelement (`OutputElement`) repräsentiert Inhalte, die dem Benutzer dargestellt werden. Eingabeelemente (`InputElement`) hingegen nehmen Eingaben des Benutzers entgegen. Hierbei muss nochmals differenziert werden, ob diese Eingaben völlig frei erfolgen können oder ob der Benutzer aus einer festen Menge von Eingabemöglichkeiten über ein Auswahlelement (`SelectElement`) wählen muss. Zusätzlich benötigt eine Benutzerschnittstelle Aktionselemente (`ActionElement`), über die vordefinierte Aktionen durch den Benutzer ausgelöst werden können. Um diese Metamodellelemente zur Spezifikation eines Strukturmodells einsetzen zu können, müssen weitere Verfeinerungen an den Metamodellelementen vorgenommen werden. Abbildung 45 zeigt das durch den Autor in [LS+07a] und [LS+07b] publizierte verfeinerte Metamodell des Strukturmodells einer Benutzerschnittstelle.

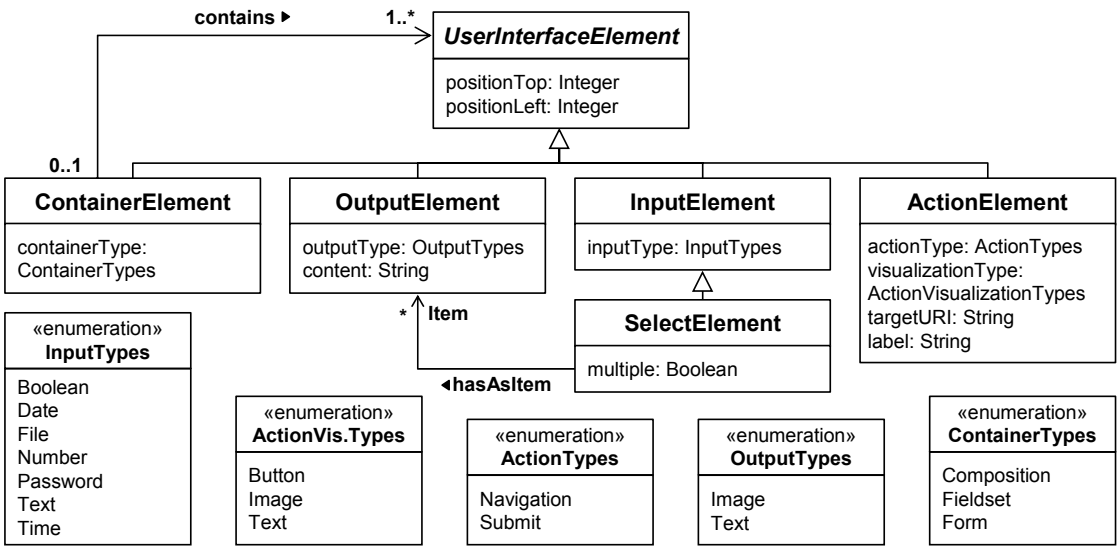

Abbildung 45: Metamodell des Strukturmodells für Benutzerschnittstellen

4.4.1.4 Benutzerschnittstellenelement

Ein Benutzerschnittstellenelement (`UserInterfaceElement`) repräsentiert die Gesamtheit aller Elemente, die einer Benutzerschnittstelle zugeordnet werden können. Als gemeinsame Eigenschaften besitzt jedes Benutzerschnittstellenelement zwei Positionsangaben, die eine zweidimensionale Positionierung des Benutzerschnittstellenelements relativ zum übergeordneten Benutzerschnittstellenelement gestattet (`positionTop`, `positionLeft`). Die Verwendung eines abstrakten Benutzerschnittstellenelements basiert auf dem Kerngedanken der Erweiterbarkeit des Metamodells. Zwar sind die im Folgenden dargestellten Elemente für das betrachtete Szenario dieser Arbeit ausreichend. Um jedoch einen einfachen Transfer der entwickelten Konzepte auf andere Szenarien und Domänen zu ermöglichen, müssen die Metamodelle entsprechende Erweiterungsmöglichkeiten bieten. Daher verwendet das Metamodell zusätzlich Enumerationen, um die im Metamodell vorhandenen konkreten Benutzerschnittstellenelemente auf schnelle und einfache Art und Weise erweitern zu können. Werden aufgrund der Betrachtung einer anderen Geschäftsdomäne oder eines anderen Szenarios beispielsweise weitere Containerelemente benötigt, so muss lediglich die entsprechende Enumeration `ContainerTypes` angepasst und gegebenenfalls erweitert werden. Das gleiche Prinzip wird daher auf alle weiteren konkreten Benutzerschnittstellenelemente des Metamodells angewendet.

Containerelement

Die statische Struktur einer Benutzerschnittstelle ist hierarchisch aufgebaut und gleicht einer Baumstruktur. Ein Benutzerschnittstellenelement steht daher mit Ausnahme der Wurzel immer in Bezug zu einem übergeordneten Benutzerschnittstellenelement. Durch das im Strukturmodell vorgesehene Containerelement (`ContainerElement`) kann diese Enthaltenseinbeziehung modelliert werden. Obwohl ein hierarchischer Aufbau des Strukturmodells einer Benutzerschnittstelle auch durch eine reflexive Assoziation am Benutzerschnittstellenelement möglich wäre, gestattet das Containerelement eine explizite Auszeichnung derjenigen Benutzerschnittstellenelemente, die andere Benutzerschnittstellenelemente beinhalten können. Durch die Assoziation zwischen dem abstrakten Benutzerschnittstellenelement und dem Containerelement kann so eine beliebig tiefe Verschachtelung der Containerelemente und damit eine beliebige Breite und Tiefe im Strukturbaum erreicht werden. Ein Containerelement muss ferner, wie im Metamodell spezifiziert, immer mindestens ein Benutzerschnittstellenelement enthalten. Leere Containerelemente sind demnach nicht zulässig. Das

Metamodell sieht drei Typen von Containerelementen vor, die durch die Enumeration `ContainerTypes` zusammenfasst. Durch die Eigenschaft `containerType` des `ContainerElements` kann so für jedes Containerelement ein eigener Containertyp festgelegt werden. Ist das Containerelement ein Formular (`ContainerTypes::Form`), so repräsentiert dieses Containerelement das Wurzelelement der gesamten Benutzerschnittstellenstruktur. Ein Containerelement kann auch dazu verwendet werden, um einen nicht näher bekannten, logischen Zusammenhang zwischen den Unterelementen des Containers durch eine Komposition (`ContainerTypes::Composition`) zu spezifizieren. Der dritte als `Fieldset` bezeichnete Containertyp (`ContainerTypes::Fieldset`) erlaubt die Spezifikation semantisch zusammengehörender Benutzerschnittstellenelemente, wie beispielsweise ein Eingabeelement und dessen Beschriftung in Form eines Ausgabeelementes.

Ausgabeelement

Das Ausgabeelement (`OutputElement`) dient der Ausgabe von Informationen, die dem Benutzer angezeigt werden müssen. Das Ausgabeelement ist daher ein einfaches Benutzerschnittstellenelement, das beispielsweise einen Text oder ein Bild repräsentiert. Den Typ eines Ausgabeelements bestimmt die Eigenschaft `outputType`, welche die in der Enumeration `OutputTypes` aufgeführten Datentypen wie beispielsweise Text (`OutputTypes::Text`) oder Bild (`OutputTypes::Image`) als Wert annehmen kann. Ein Ausgabeelement kann ferner entweder alleine oder im Kontext eines weiteren Benutzerschnittstellenelements, wie etwa als Beschriftung eines Eingabeelements betrachtet werden. Dieser semantische Zusammenhang kann auf einfache Weise durch ein übergeordnetes Containerelement vom Typ `Fieldset` hergestellt werden. Da das Ausgabeelement keinerlei Eingaben entgegen nimmt, kann der Inhalt, den ein Ausgabeelement visualisiert, durch den Benutzer später nicht – im Sinne von schreibend bearbeiten – verändert werden. Hierzu werden Eingabe- oder Auswahlelemente benötigt.

Eingabeelement

Das Eingabeelement (`InputElement`) nimmt Eingaben des Benutzers entgegen. Diese können sehr vielfältig und unterschiedlich sein, weshalb das Eingabeelement über die Eigenschaft `inputType` durch unterschiedliche Eingabetypen (`InputTypes`) näher spezifiziert werden muss. Steht dem Benutzer beispielsweise ein freies Textfeld zur Verfügung (`InputTypes::Text`), so kann von ihm eine Eingabe in Form eines einfachen Wortes wie beispielsweise ein Name bis hin zu einem nahezu beliebig langen, nicht näher strukturierten Text erfolgen. Weitere Eingabetypen sind an heute bekannte Formen der Benutzereingabe, wie etwa eine Passworteingabe (`InputTypes::Password`) oder die Angabe eines Datums (`InputTypes::Date`), angelehnt. Trotz der genauen Spezifikation des Typs der Eingabe sind die Werte, die durch einen Benutzer eingegeben werden können, insbesondere bei dem Eingabetyp Text im Voraus nicht näher spezifizierbar. Um die durch einen Benutzer erwarteten Eingaben näher spezifizieren zu können, sieht das Metamodell mit dem Auswahlelement eine Spezialisierung des Eingabeelementes vor.

Auswahlelement

Das Auswahlelement (`SelectElement`) dient als Spezialisierung des Eingabeelementes zur Spezifikation von Eingaben, bei denen der Benutzer keine freie Wahl seiner Eingabe hat, sondern aus einer Menge bestehender Eingaben wählen muss. Hat die boolesche Eigenschaft `multiple` des Auswahlelements den Wert `false`, so darf der Benutzer genau ein Element aus der Menge der zur

Verfügung stehenden Auswahlmöglichkeiten auswählen. Hat die Eigenschaft den Wert `true`, so kann der Benutzer mehrere Elemente aus der zur Verfügung stehenden Menge auswählen. Der Typ der ausgewählten Elemente wird durch das übergeordnete Eingabeelement und dessen `inputType`-Eigenschaft vererbt. Um die Auswahlmöglichkeiten für den Benutzer darstellen zu können, benötigt das Auswahlelement mindestens ein assoziiertes Ausgabeelement, durch welches die Wahlmöglichkeiten des Benutzers spezifiziert werden können. Dementsprechend stellt der erste OCL-Ausdruck sicher, dass einem Auswahlelement immer mindestens ein Ausgabeelement zugeordnet wird, falls die `multiple`-Eigenschaft des Auswahlelements den Wert `false` besitzt. Der zweite OCL-Ausdruck überprüft, ob einem Auswahlelement mehr als ein Ausgabeelement zugeordnet ist, wenn die `multiple`-Eigenschaft des Auswahlelements auf `true` gesetzt ist, da dann der Benutzer mehr als ein Element auswählen darf und er dementsprechend mehr als ein Element zur Auswahl angeboten bekommen muss. Durch einen dritten OCL-Ausdruck muss der Eingabetyp Passwort (`InputTypes::Password`) für ein Auswahlelement unterbunden werden, da dieser Eingabetyp für ein Auswahlelement nicht geeignet ist. Eine vierte Einschränkung legt fest, dass maximal nur zwei Elemente zur Auswahl stehen dürfen, sofern der Eingabetyp eines Auswahlelements ein boolescher Wert (`InputTypes::Boolean`) ist. Eine letzte Einschränkung hält fest, dass die Ausgabeelemente, welche die zur Auswahl stehenden Elemente eines Auswahlelements repräsentieren, vom Ausgabetyp Text (`OutputTypes::Text`) sein müssen, da ein Ausgabeelement die textuelle Beschreibung eines Auswahlgegenstands (`Item`) spezifiziert.

```
context SelectElement
-- If user may choose one item, there has to be at least one item to
-- choose from
inv: (self.multiple = false) implies self.outputElement -> size() > 0

-- If user may choose more than one item at least two items are needed
inv: (self.multiple = true) implies self.outputElement -> size() > 1

-- InputType::Password is no valid type for a SelectElement
inv: not (self.inputType = InputTypes::Password)

-- If Boolean, there has to be a maximum of two items to choose from
inv: (self.inputType = InputTypes::Boolean) implies
       (self.item -> size() < 3)

-- All Items have to be of OutputType Text
Inv: self.item -> forAll(outputType = OutputTypes::Text)
```

Aktionselement

Das Aktionselement (`ActionElement`) gestattet die Spezifikation einer bestimmten vordefinierten Aktion, die durch den Benutzer über die Benutzerschnittstelle ausgelöst werden kann. Durch die Eigenschaft Aktionstyp (`actionType`) kann der Typ der auszulösenden Aktion, wie beispielsweise das Beenden einer Benutzereingabe (`ActionTypes::Submit`) oder die Navigation zu einem bestimmten Ziel (`ActionTypes::Navigation`), konkretisiert werden. Alle möglichen Aktionstypen werden in der Enumeration `ActionTypes` zusammengefasst. Ferner sieht das Aktionselement Eigenschaften zur Spezifikation einer Bezeichnung (`label`) und der Art der Visualisierung (`vizualizationType`) vor. Die möglichen Arten der Visualisierung werden durch die Enumeration `ActionVisualizationTypes` zusammengefasst und sehen eine Visualisierung des Aktions-

elements durch einen einfachen Schaltknopf (`ActionVisualization-Types::Button`), durch einen Text-Verweis (`ActionVisualizationTypes::Text`) oder auch durch einen Grafik-Verweis (`ActionVisualizationTypes::Image`) vor. Abschließend besitzt das Aktionselement eine Eigenschaft Zieladresse (`targetURI`), zu welcher die beim Auslösen einer Aktion navigiert wird.

Mit den vorgestellten Benutzerschnittstellenelementen lässt sich die statische Struktur einer formularbasierten Benutzerschnittstelle spezifizieren. Das plattformunabhängige Strukturmodell einer Benutzerschnittstelle trifft jedoch keinerlei Aussagen hinsichtlich des Layouts der einzelnen Elemente einer Benutzerschnittstelle. Hierzu können erst dann Angaben gemacht werden, wenn die Parameter der Plattform, auf der die Benutzerschnittstelle zum Einsatz kommen soll, bekannt sind. Als ein Beispiel für solch einen Parameter sei die Anzeigefläche genannt. Allein dieser eine Parameter bestimmt wesentlich, wie die einzelnen Benutzerschnittstellenelemente visualisiert werden können. Da diese Arbeit eine rein funktionale Betrachtung der Benutzerschnittstelle vornimmt (vgl. dazu Prämisse P2), werden die Aspekte des Layouts, aber auch die der Benutzerfreundlichkeit im weiteren Verlauf der Arbeit nicht diskutiert. Abschnitt 7.3 greift diese Aspekte jedoch erneut auf und zeigt mögliche Erweiterungsansätze in diese Richtung auf.

4.4.2 Domänenprofil und Strukturprofil

Gleichsam den zuvor entworfenen Metamodellen müssen die im vorangegangenen Abschnitt neu hinzugekommenen Metamodelle zur Spezifikation einer Benutzerschnittstelle auf Elemente eines bestehenden Metamodells abgebildet werden. Als Basis dient erneut das Metamodell der UML, das durch zwei zusätzliche UML-Profile erweitert wird. Das erste UML-Profil adressiert die in Abbildung 43 (Seite 108) dargestellten Metamodellelemente, die zur inhaltlichen Spezifikation einer Benutzerschnittstelle durch das benutzerzentrische Domänenmodell benötigt werden. Das kurz als **Domänenprofil** bezeichnete UML-Profil erweitert die UML-Metaklasse Schnittstelle (`Interface`) um den benötigten abstrakten Stereotyp `UserInterface`. Dieser wiederum wird durch die beiden Stereotypen Benutzereingabeschnittstelle (`UserInputInterface`) und Benutzerausgabeschnittstelle (`UserOutputInterface`) spezialisiert. Das ferner benötigte Metamodellelement Geschäftsobjekt (`BusinessObject`) wird als Stereotyp spezifiziert, der die UML-Metaklasse `Class` erweitert. Durch die Umsetzung der Metamodellelemente auf das UML-Metamodell wird das Metamodellelement Eigenschaft (`Property`) nicht weiter benötigt, da das UML-Metamodell für eine Klasse und für eine Schnittstelle den Besitz von Eigenschaften bereits vorsieht. Abbildung 46 illustriert die Stereotypen des benutzerzentrischen Domänenprofils.

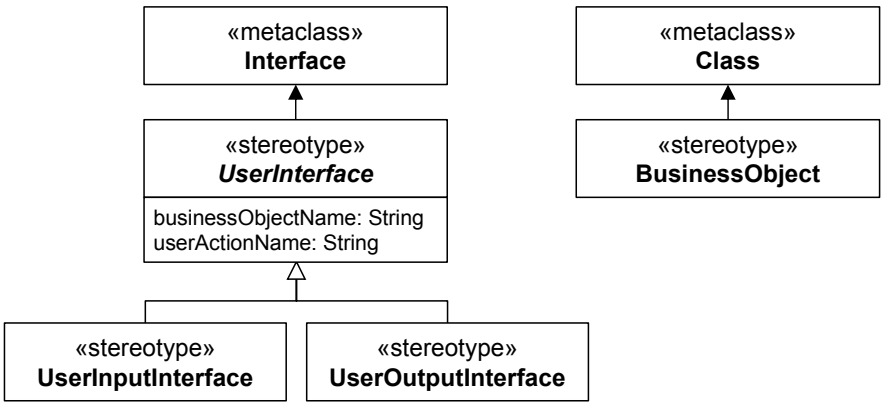

Abbildung 46: Stereotypen des benutzerzentrischen Domänenprofils

Das zweite UML-Profil adressiert die in Abbildung 44 (Seite 111) dargestellten Metamodellelemente des plattformunabhängigen Strukturmodells, durch welche die statische Struktur einer Benutzerschnittstelle spezifiziert werden kann. Dementsprechend wird das UML-Profil im weiteren Verlauf als **Strukturprofil** bezeichnet. Das abstrakte Benutzerschnittstellenelement (`UserInterface-Element`) erweitert die Metaklasse `Class` des UML-Metamodells. Das Aktions-, Ausgabe- und Eingabeelement (`Action-`, `Output-`, `InputElement`) wird analog zum vorgestellten konzeptionellen Metamodell direkt vom Benutzerschnittstellenelement abgeleitet. Im Gegensatz zum konzeptionellen Metamodell wird das Containerelement (`ContainerElement`) jedoch nicht aus dem Benutzerschnittstellenelement abgeleitet, sondern als eigenständiger Stereotyp als Erweiterung der UML-Metaklasse Paket (`Package`) bereitgestellt, da die Semantik des Containerelements im konzeptionellen Modell der Semantik des `Package` des UML-Metamodells sehr nahe kommt. Diese Art der Umsetzung des Containerelements erleichtert zusätzlich die Spezifikation der Enthaltensein-beziehung zwischen Containerelement und den weiteren Benutzerschnittstellenelementen, da das UML-Metamodell die Metaklasse `Package` für die Gruppierung zusammengehöriger Elemente vorsieht und damit die Enthaltenseinbeziehung automatisch durch diese Metaklasse ausgedrückt werden kann. Das Auswahlelement (`SelectElement`) ist erneut als Spezialisierung des Eingabeelements realisiert. Abbildung 47 illustriert die Stereotypen des Strukturprofils.

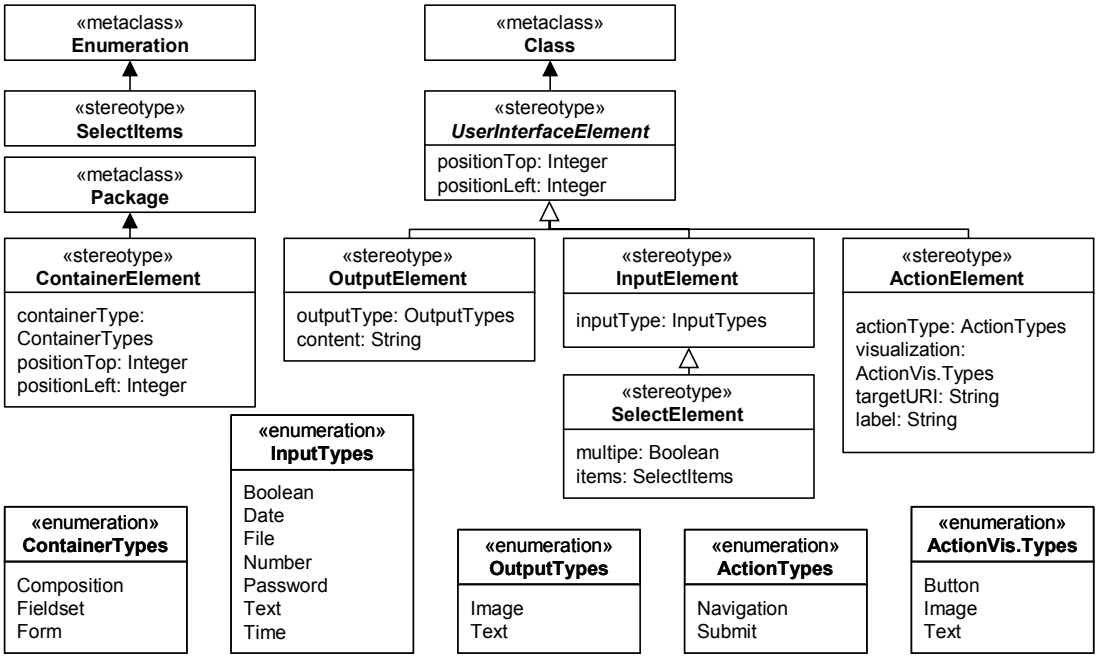

Abbildung 47: Stereotypen des Strukturprofils

Bei der Umsetzung der Metamodelle des benutzerzentrischen Domänenmodells und des Strukturmodells kommen vier Metaklassen der UML zum Einsatz, die im Folgenden näher beleuchtet werden. Die im Entwurf des Metamodells durch OCL-Ausdrücke präzisierte statische Semantik muss für das benutzerzentrische Domänenprofil und das Strukturprofil ebenfalls Gültigkeit behalten. Einige der definierten Einschränkungen werden daher beispielhaft in den folgenden Abschnitten aus dem Entwurf in die Profile übernommen.

4.4.2.1 Erweiterung der Metaklasse Klasse

Im benutzerzentrischen Domänenprofil wird die Metaklasse Klasse (`Class`) durch den Stereotyp Geschäftsobjekt (`BusinessObject`) erweitert. Da eine Klasse nach [OMG-UML2-Super] bereits

über Eigenschaften (`Properties`) verfügt, wird das zuvor spezifizierte Metamodellelement `Property` nicht mehr benötigt.

Das Strukturprofil erweitert die Metaklasse `Class` um den abstrakten Stereotyp Benutzerschnittstellenelement (`UserInterfaceElement`). Um eine relative Positionsangabe eines Benutzerschnittstellenelementes angeben zu können, besitzt der Stereotyp die beiden Eigenschaften `positionTop` und `positionLeft`. Diese beiden Eigenschaften können mit Integer-Werten belegt werden, welche die Position relativ zum Elternelement (in der Regel ein Containerelement) im Sinne von „erstes Element von oben und zweites von links“ erfassen. Das abstrakte Benutzerschnittstellenelement wird durch die drei Stereotypen Ausgabe- (`OutputElement`), Eingabe- (`InputElement`) und Aktionselement (`ActionElement`) spezialisiert. Das Auswahlelement (`SelectElement`) erhält im Vergleich zur Spezifikation im konzeptionellen Metamodell eine weitere Eigenschaft `items`, durch welche die Auswahlmöglichkeiten des Benutzers über eine Enumeration spezifiziert werden können (siehe Abschnitt 4.4.2.4). Die in Abschnitt 4.4.1.4 spezifizierte statische Semantik der Benutzerschnittstellenelemente muss auch im Strukturprofil Gültigkeit behalten. Der folgende OCL-Ausdruck formalisiert für das Strukturprofil die im Entwurf motivierte Einschränkung, dass ein Auswahlelement nicht vom Typ `Password` sein und ein Auswahlelement vom Typ `Boolean` maximal zwei `items` zur Auswahl stellen darf.

```
context SelectElement
inv: not (self.inputType = InputElementTypes::Password)
inv: (self.inputType = InputElementTypes::Boolean)
        implies (self.items.ownedLiteral -> size() < 3)
```

4.4.2.2 Erweiterung der Metaklasse Schnittstelle

Das UML-Metamodell sieht eine Metaklasse Schnittstelle (`Interface`) vor, die wie folgt definiert ist [OMG-UML2-Super]:

> *“An interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. An interface specifies a contract; any instance of a classifier that realizes the interface must fulfill that contract. [..] Since interfaces are declarations, they are not instantiable. Instead, an interface specification is implemented by an instance of an instantiable classifier, which means that the instantiable classifier presents a public facade that conforms to the interface specification.”*

Diese Definition spiegelt die im konzeptionellen Metamodell gedachte Semantik der Benutzerschnittstelle wider. Eine Benutzerschnittstelle ermöglicht dem Benutzer den Zugriff auf ein Anwendungssystem und gestattet diesem, in einer Benutzeraktion ein oder mehrere Geschäftsobjekte zu bearbeiten. Aus diesem Grund verfügt der Stereotyp Benutzerschnittstelle (`UserInterface`) als Erweiterung der UML-Metaklasse Schnittstelle (`Interface`) über die beiden Eigenschaften `userActionName` und `businessObjectName`, mittels derer der Bezug einer Benutzerschnittstelle zu der zugehörigen Benutzeraktion und den zu bearbeitenden Geschäftsobjekten hergestellt werden kann. Um eine verfeinerte Spezifikation der Eingaben, die von einem Benutzer erwartet werden und der Ausgaben, die einem Benutzer angezeigt werden müssen, umsetzten zu können, wird der abstrakte Stereotyp `UserInterface` wie im konzeptionellen Metamodell des Strukturmodells durch die beiden Stereo-

typen Benutzereingabeschnittstelle (`UserInputInterface`) und Benutzerausgabeschnittstelle (`UserOutputInterface`) spezialisiert.

Beide Benutzerschnittstellentypen umfassen Eigenschaften, die dem Benutzer angezeigt und gegebenenfalls auch zur Bearbeitung freigegeben werden müssen. Die Gesamtheit der Eigenschaften der Benutzerschnittstellentypen muss daher in einem assoziierten Geschäftsobjekt vorhanden sein. Da die Benutzerschnittstellentypen als Erweiterung der UML-Metaklasse Schnittstelle spezifiziert sind, können Benutzerschnittstellen durch ein als Klasse spezifiziertes Geschäftsobjekt implementiert werden. Dieser komplexe Sachverhalt soll an einem abstrakten Beispiel in Abbildung 48 verdeutlicht werden.

Neben den beiden Positionseigenschaften (`positionTop`, `positionLeft`) muss eine Benutzerschnittstelle zusätzlich immer über mindestens eine weitere Eigenschaft in Form eines Attributs verfügen, um aus Sicht des Benutzers leere Benutzerschnittstellen ausschließen zu können. Der folgende OCL-Ausdruck sorgt für die Einhaltung dieser Einschränkung.

```
context UserInterface inv:
self.ownedAttribute -> size() > 2
```

4.4.2.3 Erweiterung der Metaklasse Paket

Das UML-Metamodell Paket (`Package`) wird nach Definition der UML-*Superstructure* [OMG-UML2-Super] verwendet, um Elemente zu gruppieren und diesen einen gemeinsamen Namensraum bereitzustellen. Diese Semantik entspricht der des Containerelements (`Container Element`), das alle weiteren Benutzerschnittstellenelemente enthalten kann. Um den hierarchischen Aufbau eines Strukturmodells ermöglichen zu können, kann ein Containerelement wiederum weitere Containerelemente enthalten. Das Metamodell sieht auch die Möglichkeit vor, ein Benutzerschnittstellenelement eigenständig ohne zugeordneten Container spezifizieren zu können. In diesem Fall bildet das Strukturmodell an sich den Container für diese Benutzerschnittstellenelemente. Gleich dem abstrakten Benutzerschnittstellenelement verfügt das Containerelement über die beiden Positionseigenschaften `positionTop` und `positionLeft`, um eine relative Positionierung eines Containers im gesamten Strukturmodell spezifizieren zu können. Abbildung 50 zeigt eine beispielhafte Verwendung des Containerelements.

4.4.2.4 Erweiterung der Metaklasse Enumeration

Die Metaklasse Enumeration (`Enumeration`) dient der Umsetzung des Stereotyps `SelectItems`. Dieser Stereotyp wird dazu verwendet, um die Elemente, aus denen ein Benutzer durch ein Auswahlelement wählen kann, durch Literale genauer zu spezifizieren. Daher muss sichergestellt werden, dass ein `SelectItem` über die passende Anzahl an Literalen entsprechend den benötigten Auswahlmöglichkeiten des Benutzers verfügt (siehe Abschnitt 4.4.2.1).

4.4.3 Anwendung der Profile

Mit dem benutzerzentrischen Domänenprofil und dem Strukturprofil kann die statische Struktur einer Benutzerschnittstelle nach einem zweistufigen Vorgehen abgeleitet werden. Dieses Vorgehen sei anhand der Benutzeraktion C_1 des in Abbildung 34 (Seite 89) dargestellten Workflows verdeutlicht. Das Workflow-Modell stellt im Kontext der modellgetriebenen Architektur ein plattformunabhängiges Modell dar, dessen dynamische Aspekte durch ein UML-Aktivitätsdiagramm visualisiert werden

können. Aus diesem Modell kann das benutzerzentrische Domänenmodell als ein weiteres plattformunabhängiges Modell abgeleitet werden. Der auf der linken Seite von Abbildung 48 dargestellte Auszug des als Beispiel dienenden Workflow-Modells sieht für die Geschäftsrolle C eine Benutzeraktion C_1 vor, in der durch einen Benutzer das Geschäftsobjekt G bearbeitet werden muss.

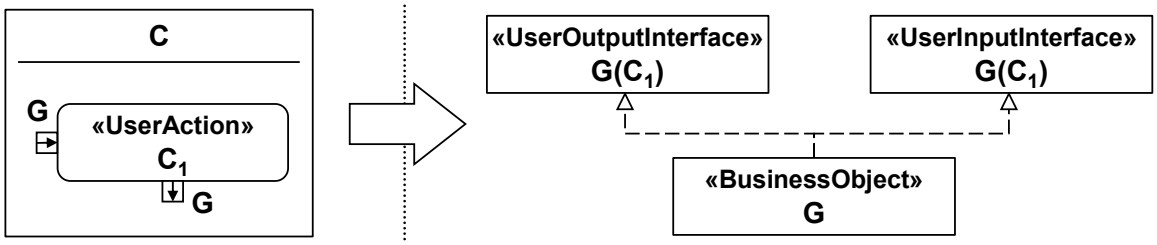

Abbildung 48: Abstraktes Beispiel zur Anwendung der Schnittstellen-Stereotypen

Da das Geschäftsobjekt G sowohl in die Benutzeraktion eingeht als auch diese wieder verlässt, muss der Benutzer schreibenden und lesenden Zugriff auf die Eigenschaften des Geschäftsobjektes erhalten. Dementsprechend muss das im rechten Teil von Abbildung 48 dargestellte benutzerzentrische Domänenmodell für die Benutzeraktion C_1 eine Benutzereingabeschnittstelle und Benutzerausgabeschnittstelle bereitstellen. Beide Stereotypen tragen die Bezeichnung $G(C_1)$, um den Bezug zu Benutzeraktion C_1 und Geschäftsobjekt G zu verdeutlichen. Da in der Benutzeraktion C_1 das Geschäftsobjekt G über die Benutzerschnittstelle bearbeitet wird, ist dieses im benutzerzentrischen Domänenmodell ebenfalls vorhanden und implementiert die beiden spezifizierten Benutzerschnittstellen $G(C_1)$.

In einem nächsten Schritt muss das benutzerzentrische Domänenmodell in Zusammenarbeit mit dem Kunden um die benötigten Eigenschaften angereichert werden, da diese aus dem Workflow-Modell nicht gewonnen werden können. Wie in Abbildung 49 dargestellt, umfasst die Ausgabeschnittstelle (`UserOutputInterface`) nach der Diskussion mit dem Kunden die Attribute 1 - 3, während die Eingabeschnittstelle (`UserInputInterface`) die Attribute 1, 2 und 5 umfasst. Dementsprechend erhält der Benutzer lesenden und schreibenden Zugriff auf die Attribute 1 und 2, da diese in beiden Schnittstellen zu finden sind. Attribut 3 wird dem Benutzer ohne Schreibrechte nur angezeigt, Attribut 5 muss durch den Benutzer neu mit einem Wert belegt werden. Das Geschäftsobjekt G implementiert beide Schnittstellen und verfügt daher über deren Eigenschaften.

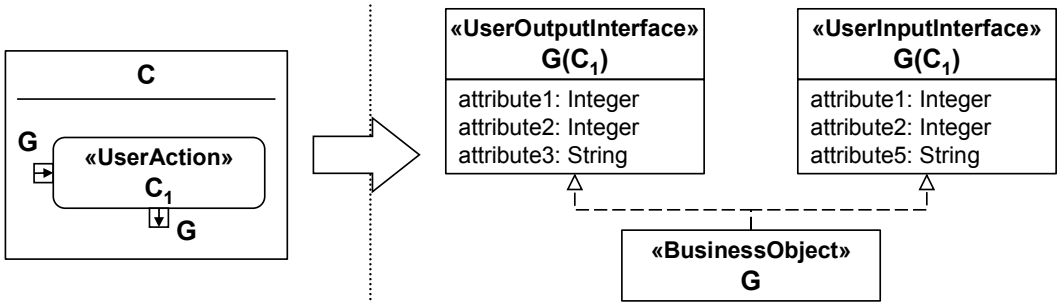

Abbildung 49: Verfeinerung des benutzerzentrischen Domänenmodells

Aus dem benutzerzentrischen Domänenmodell kann nun, wie Abbildung 50 andeutet, in einem zweiten Abbildungsschritt das Strukturmodell der Benutzerschnittstelle abgeleitet werden. Hierzu ist keine Verfeinerung des Modells mehr notwendig, da dieses bereits alle notwendigen Informationen vorhält. Zunächst muss für die Benutzeraktion C_1 ein neues Containerelement C_1 als Wurzelelement der benötigten Benutzerschnittstelle angelegt werden. Da in Benutzeraktion C_1 das Geschäftsobjekt G

bearbeitet wird, erfolgt die Erweiterung des Containerelements C_1 um ein weiteres Containerelement G. Für jede Eigenschaft des Geschäftsobjekts wird erneut ein Containerelement mit dem entsprechenden Bezeichner angelegt, um so auf einfache Art und Weise zusammengehörende Benutzerschnittstellenelemente modellieren zu können. Da Attribut 1 sowohl in der Benutzereingabeschnittstelle als auch in der Benutzerausgabeschnittstelle vorhanden ist, wird das Attribut dem Benutzer zur Bearbeitung vorgelegt und daher auf ein Eingabeelement (`InputElement`) `Attribute1` in Container `Attribute1` abgebildet. Zusätzlich enthält der Container ein Ausgabeelement (`OutputElement`) `Attribute1`, das eine geeignete Beschriftung des gleichnamigen Eingabeelements darstellt und somit dem Benutzer bei der Eingabe behilflich ist. Da Attribut 3 nur in der Benutzerausgabeschnittstelle vorhanden ist, wird es in Container `Attribute3` auf ein Ausgabeelement abgebildet, das durch den Benutzer nicht bearbeitet werden kann. Attribut 5 zeigt schließlich den umgekehrten Fall auf, dass ein Attribut nur in einer Benutzereingabeschnittstelle, nicht aber in der Benutzerausgabeschnittstelle vorhanden ist und daher in Container `Attribute5` zwar auf ein Eingabeelement abgebildet wird, dieses aber nicht mit einem Wert vorbelegt wird.

Die in Abbildung 48 und Abbildung 49 mittels Pfeilen angedeuteten Ableitungen der einzelnen Modellelemente werden im folgenden Kapitel auf eine formale Grundlage gestellt. Ziel ist es, alle Modelle automatisch durch Metamodelltransformationen entlang des skizzierten Modellierungsvorgehens aufeinander abbilden zu können. Dementsprechend widmet sich Abschnitt 5.2.4 einer automatischen Ausführung der in Abbildung 48 bis Abbildung 50 angedeuteten Abbildungsschritte und geht dort auf die benötigten formalen Abbildungsregeln detailliert ein.

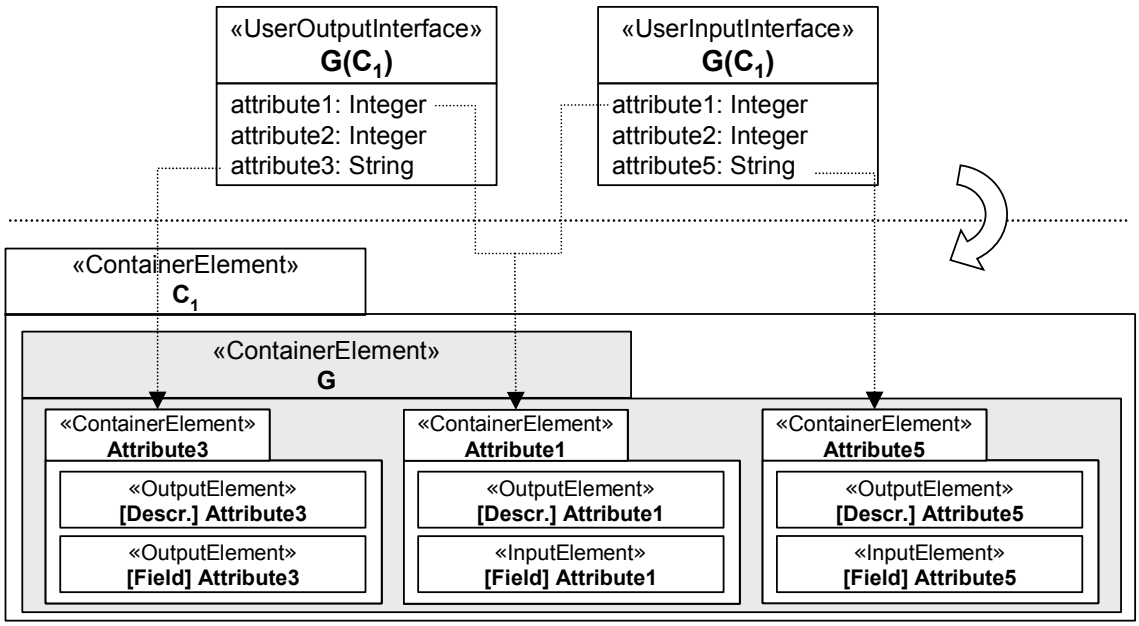

Abbildung 50: Abbildung des benutzerzentrischen Domänenmodells auf das Strukturmodell

Das Strukturmodell der Benutzerschnittstelle stellt noch keinen Bezug zu einer konkreten Plattform her. Wie der Autor in [LS+07b] zeigt, kann es somit als plattformunabhängiges Modell auf eine Vielzahl unterschiedlicher Plattformen unter Verwendung der entsprechenden plattformspezifischen Modelle abgebildet werden. Kapitel 6 zeigt in Form eines Tragfähigkeitsnachweises eine entsprechende Abbildung auf eine konkrete Plattform.

Um einen Workflow mit Benutzerinteraktion nach einem modellgetriebenen Entwicklungsvorgehen auf eine dienstorientierte Architektur abbilden zu können, werden neben den in Abschnitt 4.3 behan-

delten Benutzeraufgaben und den in diesem Abschnitt behandelten Benutzerschnittstellen geeignete Dienste benötigt, welche die Aktionen eines Workflows mit geeigneter Fachfunktionalität unterstützen. Wie im Überblick der einzelnen Beiträge dieser Arbeit in Abbildung 29 (Seite 80) festgehalten, soll daher neben den Modellen zur Spezifikation der Benutzeraufgaben und der Benutzerschnittstellen auch ein Modell zur Spezifikation der benötigten Dienste im modellgetriebenen Entwicklungsvorgehen berücksichtigt werden. Mit dem hierzu benötigten Metamodell befasst sich der folgende Abschnitt.

4.5 Spezifikation der Dienste

Workflow-Modelle umfassen diejenigen Teile der Geschäftsprozesse eines Unternehmens, die durch IT unterstützt werden können. Workflow-Modelle stellen dabei bewusst keinen direkten Bezug zur benötigten IT-Unterstützung her. Bestehende Ansätze, diesen Bezug entlang einem modellgetriebenen Entwicklungsvorgehen herzustellen, überführen ein Workflow-Modell in eine sogenannte Prozessausführungssprache wie beispielsweise BPEL. Mit BPEL kann die Kontrollflussperspektive eines Workflows in Form einer Spezifikation in XML festgehalten werden. Viele der heute vorhandenen Ausführungsumgebungen für Workflows nutzen BPEL-Spezifikationen, weshalb sich in deren Kontext häufig die aus dem Englischen entnommene Bezeichnung einer BPEL-*Engine* für eine Workflow-Ausführungsumgebung findet.

Obwohl eine Prozessausführungssprache wie BPEL einen Workflow auf technischer Ebene abbilden kann, fokussiert diese Abbildung, die durch unterschiedliche Forschungsansätze bereits untersucht wurde, im Kern den dynamischen Aspekt des Kontrollflusses eines Workflows. Neben der dynamischen und technologieunabhängigen Sicht auf einen Workflow muss jedoch zusätzlich eine technologiegetriebene Sicht eingenommen werden. Die eigentliche Funktionalität, die in einem Workflow zur Unterstützung der Aktionen benötigt wird, muss durch die IT-Systeme der IT-Unterstützung in Form von Diensten erbracht und daher näher spezifiziert werden. Hierbei nimmt diese Arbeit bewusst eine erneute Unterscheidung zwischen den Benutzeraktionen und Systemaktionen eines Workflows vor. Während zur Unterstützung einer Systemaktion völlig unterschiedliche fachfunktionale Dienste benötigt werden, ist diese Vielfalt bei einer Benutzeraktion nicht gegeben. Trifft eine Workflow-Instanz auf eine auszuführende Benutzeraktion, so muss die eigentliche Arbeitseinheit von einem Menschen und nicht von einem Anwendungssystem erbracht werden. Aufgabe der IT-Unterstützung ist es, dem Benutzer eine geeignete Benutzerschnittstelle zur Verfügung zu stellen und die Ausführung der zu instanziierenden Benutzeraufgabe zu überwachen. Diese Fachfunktionalität wird im weiteren Verlauf dieser Arbeit einerseits durch eine Benutzeraufgabenverwaltung erbracht, die ihre Fachfunktionalität als Dienst über eine geeignete Dienstschnittstelle bereitstellt (vgl. Abschnitt 2.2.2). Andererseits werden die zur Ausführung eines Workflows benötigten Benutzerschnittstellen ebenfalls als Dienste betrachtet. Dementsprechend können die präsentationsorientierten Diensten in einer logischen Unterteilung einer dienstorientierten Architektur der Präsentationsschicht zugeordnet werden, der Dienst der Benutzeraufgabenverwaltung der Kompositionsschicht (vgl. Abschnitt 2.4). Abschnitt 5.3 ordnet diese Dienste in bestehende Referenzmodelle dienstorientierter Architekturen ein.

Bedingt durch die Tatsache, dass Dienstnachrichten, die zwischen einer Benutzeraktion und der Benutzeraufgabenverwaltung zur Laufzeit ausgetauscht werden, direkt mit den Eingaben von und Ausgaben an den Benutzer über die Benutzerschnittstelle korrelieren, sind durch das in Abschnitt 4.4 konzipierte benutzerzentrische Domänenmodell bereits grundlegende Informationen zur Ableitung der zur Laufzeit benötigten Dienstnachrichten vorhanden. Daher muss das benutzerzentrische Domänenmodell als Grundlage für die Ableitung der benötigten Dienstnachrichten verwendet werden, um auf

diese Weise die Komplexität des Entwicklungsvorgehens weiter reduzieren zu können. Da ferner der zur Benutzeraufgabenverwaltung benötigte Umfang an Dienstoperationen wie beispielsweise das Anlegen einer Benutzeraufgabe bereits im Voraus bekannt ist, strebt diese Arbeit an, nicht nur die zur Kommunikation mit der Benutzeraufgabenverwaltung benötigten Dienstnachrichten, sondern auch die Benutzeraufgabenverwaltung als Dienst an sich bereits auf Modellebene spezifizieren zu können. Dementsprechend muss ein geeignetes Metamodell zur Spezifikation der Dienste im modellgetriebenen Entwicklungsvorgehen bereitgestellt und dieses in Bezug zu dem bereits konzipierten benutzerzentrischen Domänenmodell gesetzt werden. Auf diese Weise kann ein integriertes, modellgetriebenes Entwicklungsvorgehen erreicht werden, das bereits vorhandene Informationen über die Benutzeraktionen eines Workflows geeignet wiederverwendet und damit die Komplexität des Entwicklungsvorgehens reduziert.

4.5.1 Konzeption des Metamodells für Dienste

Die zur Erreichung des Zieles dieser Arbeit konzipierten Modelle umfassen vielfältige Informationen hinsichtlich der Interaktion eines Menschen in einem Workflow, die auch hinsichtlich der Spezifikation der benötigten Dienste zur Unterstützung von Benutzerinteraktion in einem modellgetriebenen Entwicklungsvorgehen weiterverwendet werden können. Ein hierzu benötigtes Metamodell zur Spezifikation von Diensten wird durch die Forschungsgruppe Cooperation & Management vorangetrieben, an dessen Konzeption der Autor beteiligt war [EK+07]. Abbildung 51 zeigt den Kern dieses Metamodells.

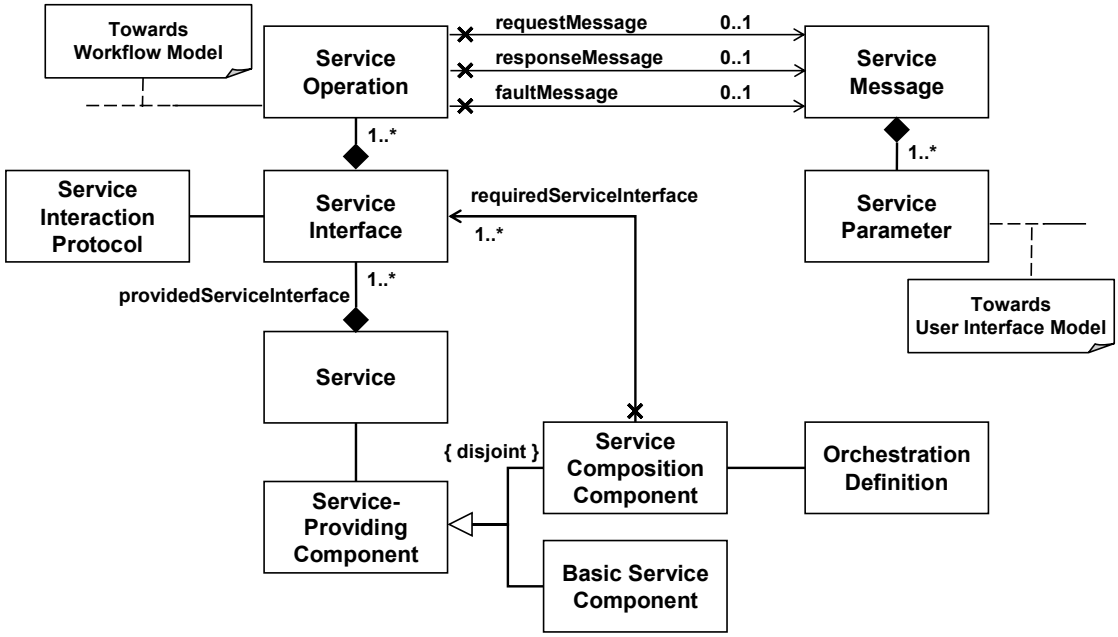

Abbildung 51: Konzeptionelles Metamodell zur Spezifikation von Diensten

Zentrales Artefakt des konzeptionellen Metamodells ist der Dienst (`Service`), der seine Fachfunktionalität an mindestens einer Dienstschnittstelle (`ServiceInterface`) zur Verfügung stellt. Jede Dienstschnittstelle verfügt über eine Menge an Dienstoperationen (`ServiceOperation`), deren Signaturen über Dienstnachrichten (`ServiceMessage`) spezifiziert werden. Einer Dienstoperation kann jeweils maximal eine Anfrage-, eine Antwort- und eine Fehlernachricht zugeordnet werden. Diese Einschränkung wird im Metamodell durch die entsprechenden Assoziationen (`request-`, `response-`, `faultMessage`) und deren Kardinalitäten mit der Multiplizität `0..1` formalisiert. Da

eine Dienstoperation mehrere Parameter benötigen beziehungsweise liefern kann, umfasst eine Dienstnachricht einen oder mehrere Dienstparameter (`ServiceParameter`). Ein Dienst muss durch eine den Dienst implementierende Komponente erbracht werden (`ServiceProvidingComponent`). Bei dieser Komponente kann es sich entweder um einen Basisdienst (`BasicServiceComponent`) oder um eine Dienstkomposition (`ServiceCompositionComponent`) handeln. Eine Dienstkomposition wird durch eine als Orchestrierung bezeichnete, eigenständige Definition der Komposition explizit gemacht (`OrchestrationDefinition`) [AC+04, Pe03]. Auf Basis dieses Metamodells können Dienste durch Modelle spezifiziert und somit in einem modellgetriebenen Entwicklungsvorgehen berücksichtigt werden.

Das in Abschnitt 4.4 eingeführte benutzerzentrische Domänenmodell, das aus einem Workflow-Modell abgeleitet wird, umfasst alle Eigenschaften der Geschäftsobjekte, die einem Benutzer zur Bearbeitung zur Verfügung gestellt werden müssen. Hat ein Benutzer beispielsweise die Aufgabe, einen Prüfungstermin für einen Studierenden auszuwählen, so wird er in der Benutzeraktion „Prüfungstermin auswählen“ Eigenschaften des Geschäftsobjektes „Prüfungsanmeldung“ bearbeiten. Dementsprechend muss zur Laufzeit einer Workflow-Instanz bei Erreichen der Benutzeraktion „Prüfungstermin auswählen“ die Benutzeraufgabenverwaltung eine neue gleichnamige Benutzeraufgabe instanziieren. Diese Instanziierung entspricht dem Aufruf einer Dienstoperation, wie etwa `createUserTask` auf dem Dienst der Benutzeraufgabenverwaltung. Die hierzu benötigten Dienstnachrichten und die in diesen Nachrichten enthaltenen Dienstparameter lassen sich direkt aus dem benutzerzentrischen Domänenmodell ableiten. Abbildung 52 zeigt anhand des bereits bekannten abstrakten Beispiels der Benutzeraktion C_1 und des Geschäftsobjekts G das verwendete Grundkonzept zur Ableitung der Dienstnachrichten aus dem benutzerzentrischen Domänenmodell auf.

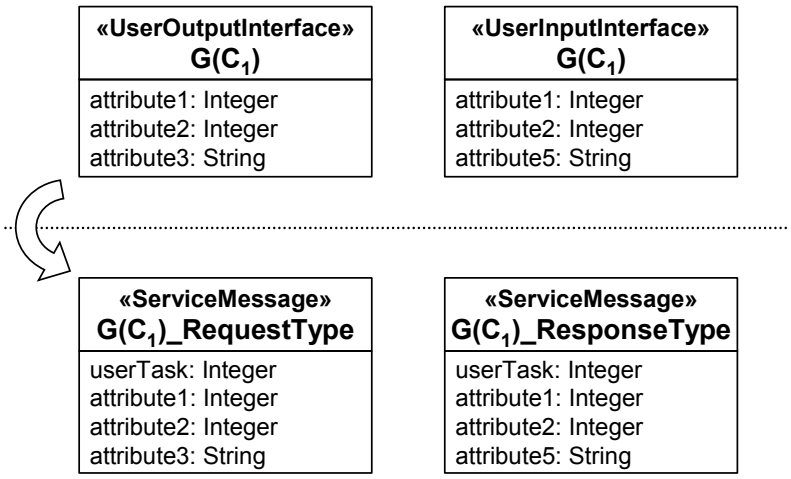

Abbildung 52: Grundkonzept der Ableitung der Dienstnachrichten

Wie in Abbildung 52 dargestellt, können die beiden Stereotypen `UserIntputInterface` und `UserOutputInterface` direkt zur Abbildung der Dienstnachrichten `G(C1)_RequestType` und `G(C1)_ResponseType` verwendet werden. Wird bei der Abarbeitung einer Workflow-Instanz die Benutzerinteraktion C_1 erreicht, dann ruft die Workflow-Ausführungsumgebung die Dienstoperation `createUserTask` mit den Eigenschaften der Dienstnachricht `G(C1)_RequestType` auf und erhält nach Abarbeitung der Instanz der Benutzeraufgabe C_1 durch den Benutzer die Werte der Eigenschaften der Dienstnachricht `G(C1)_ResponseType` als Antwort zurück, die dann der Workflow-Ausführungsumgebung zur weiteren Verarbeitung zur Verfügung stehen. Um je nach Implementie-

rung der Benutzeraufgabenverwaltung eine eindeutige Zuordnung der Dienstnachrichten zu den Benutzeraufgaben zu ermöglichen, sieht die Abbildung eine zusätzliche Eigenschaft `userTask` vor, anhand derer die zugeordnete Benutzeraufgabe spezifiziert werden kann.

Auf diesem Grundkonzept aufbauend kann für alle Benutzeraktionen das benötigte Dienstmodell abgeleitet werden. Um die Abbildung des benutzerzentrischen Domänenmodells auf das Dienstmodell in einem modellgetriebenen Entwicklungsvorgehen durchführen zu können, muss das konzipierte Metamodell des Dienstmodells durch ein konkretes Metamodell umgesetzt werden.

4.5.2 Das Dienstprofil

Die Umsetzung des Metamodells erfolgt wie bei allen anderen Metamodellen zuvor durch ein UML-Profil. Hierbei stehen insbesondere die Operationen eines Dienstes und die zu deren Aufruf benötigten Dienstnachrichten und Dienstparameter im Fokus dieses Abschnittes. Aus diesem Grund blendet das in Abbildung 53 dargestellte UML-Profil, das im weiteren Verlauf als **Dienstprofil** bezeichnet wird, die Umsetzung der konzeptionellen Metamodellelemente aus, welche die eigentliche Implementierung eines Dienstes und dessen internen Aufbau adressieren. Ob der Dienst der Benutzeraufgabenverwaltung als Basisdienst oder aus mehreren Basisdiensten in Form einer Dienstkomposition implementiert wird, ist aufgrund der Tatsache, dass beide Implementierungsarten je eine Dienstschnittstelle bereitstellen, welche die interne Realisierung verschattet, nicht weiter von Belang. Dementsprechend abstrahiert das Dienstprofil von diesen Implementierungsdetails und verschattet diese hinter der Dienstschnittstelle einer diensterbringenden Komponente (`ServiceProviding-Component`).

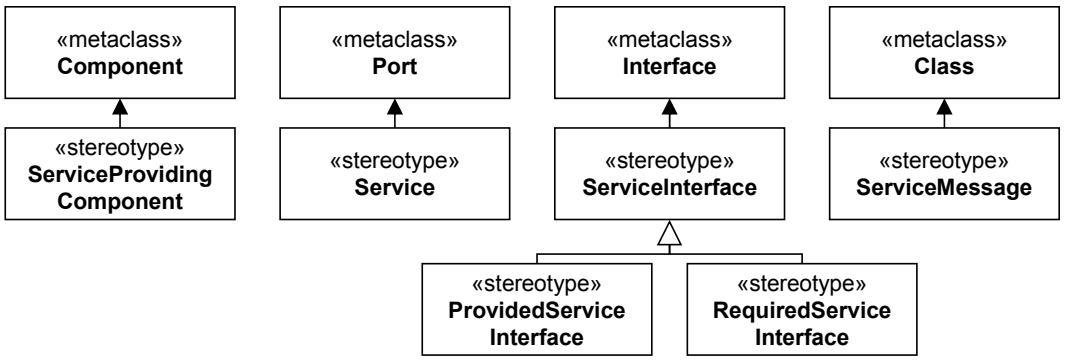

Abbildung 53: Stereotypen des Dienstprofils

4.5.2.1 Erweiterung der Metaklasse Komponente

Die UML-Metaklasse Komponente (`Component`) dient zur Spezifikation funktionaler Bausteine eines Anwendungssystems, deren interne Implementierung durch eine Schnittstelle gekapselt wird und demnach für die Außenwelt der Komponente transparent ausgetauscht werden kann. Diese Semantik entspricht der diensterbringenden Komponente (`ServiceProvidingComponent`) des Dienstmodells, die daher als Erweiterung der UML-Metaklasse Komponente im Dienstprofil spezifiziert wird. Die Unterscheidung, ob eine diensterbringende Komponente als atomar oder komponiert anzusehen ist, wird durch das Dienstprofil nicht weiter verfolgt.

4.5.2.2 Erweiterung der Metaklasse Port

Das zentrale Element des Dienstes (`Service`) ist als Erweiterung der UML-Metaklasse Port (`Port`) spezifiziert, da diese UML-Metaklasse die eigentliche Aufgabe eines Dienstes, Funktionalität an einer

Schnittstelle bereitzustellen [Le03], geeignet formalisiert. Ein Port ist immer an eine diensterbringende Komponente gekoppelt. Ein Dienst stellt wiederum mehrere Schnittstellen bereit oder benötigt diese.

4.5.2.3 Erweiterung der Metaklasse Schnittstelle

Die in Abschnitt 4.4.2.2 aus der UML-*Superstructure* [OMG-UML2-Super] zitierte Beschreibung des UML-Metamodells Schnittstelle (`Interface`) trifft ebenfalls auf die Semantik der Dienstschnittstelle des Dienstmodells zu. Das Dienstprofil erweitert daher diese Metaklasse um die beiden Stereotypen `ProvidedServiceInterface` und `RequiredServiceInterface`, durch welche die benötigten und angebotenen Dienstschnittstellen eines Dienstes modelliert und explizit ausgezeichnet werden können. Die Metaklasse Schnittstelle verfügt nach der UML-*Superstructure* zusätzlich über Operationen, weshalb das Dienstprofil die im konzeptionellen Metamodell vorgesehene Dienstoperation (`ServiceOperation`) nicht als eigenständigen Stereotyp umsetzt, sondern dieses Metamodellelement ebenfalls direkt auf eine Dienstschnittstelle abbildet. Die Signatur einer Dienstoperation kann je nach Implementierung bis zu drei typisierte Parameter umfassen, die den drei benötigten Dienstnachrichten (`request-`, `response-`, `faultMessage`) entsprechen. Durch Dienstnachrichten (`ServiceMessage`) werden die Datentypen spezifiziert, die von einer Dienstoperation erwartet beziehungsweise von einer Dienstoperation zurückgegeben werden.

4.5.2.4 Erweiterung der Metaklasse Klasse

Die UML-Metaklasse Klasse (`Class`) wird im Dienstprofil durch den Stereotyp Dienstnachricht (`ServiceMessage`) erweitert. Zur Spezifikation einer Dienstoperation werden im Wesentlichen zwei unterschiedliche Dienstnachrichten benötigt: eine Anfragenachricht (`requestMessage`) und eine Antwortnachricht (`responseMessage`). Die im konzeptionellen Metamodell zusätzlich vorhandene Fehlernachricht wird an dieser Stelle durch das Dienstprofil ausgeblendet, kann aber jederzeit durch einen dritten Parameter einer Dienstoperation ergänzt werden. Eine Dienstnachricht setzt sich, wie im konzeptionellen Metamodell gezeigt, aus mehreren Dienstparametern (`ServiceParameter`) zusammen. Da die UML-Metaklasse Klasse selbst über Eigenschaften im Sinne von Attributen verfügt, muss die im konzeptionellen Metamodell vorgenommene getrennte Betrachtung von Dienstnachricht und Dienstparameter im Dienstprofil bei Einhaltung der Semantik nicht weiter aufrecht erhalten werden, sondern kann durch den Stereotyp Dienstnachricht zusammengefasst werden.

4.5.3 Anwendung des Dienstprofils

Mit dem entwickelten Dienstprofil können die zur Ausführung einer Benutzeraktion notwendigen Dienstnachrichten aber auch die Benutzeraufgabenverwaltung als Dienst sowie deren Dienstoperationen und Dienstschnittstellen bereits auf Modellebene spezifiziert werden. Der zentrale Mehrwert des Dienstprofils liegt somit in der Möglichkeit der Wiederverwendung von Informationen hinsichtlich der Unterstützung der Benutzeraktionen eines Workflows, die bereits in dem zuvor eingeführten benutzerzentrischen Domänenmodell (Abschnitt 4.4) erfasst und formal spezifiziert wurden. Durch das Dienstprofil können diese Informationen nun in einem eigenen Modell manifestiert und damit den entsprechenden Fachentwicklern zur Verfügung gestellt werden. Abbildung 54 greift das in Abbildung 52 dargestellte Grundkonzept der Ableitung des Dienstmodells auf und zeigt beispielhaft die Anwendung der bereitgestellten Stereotypen.

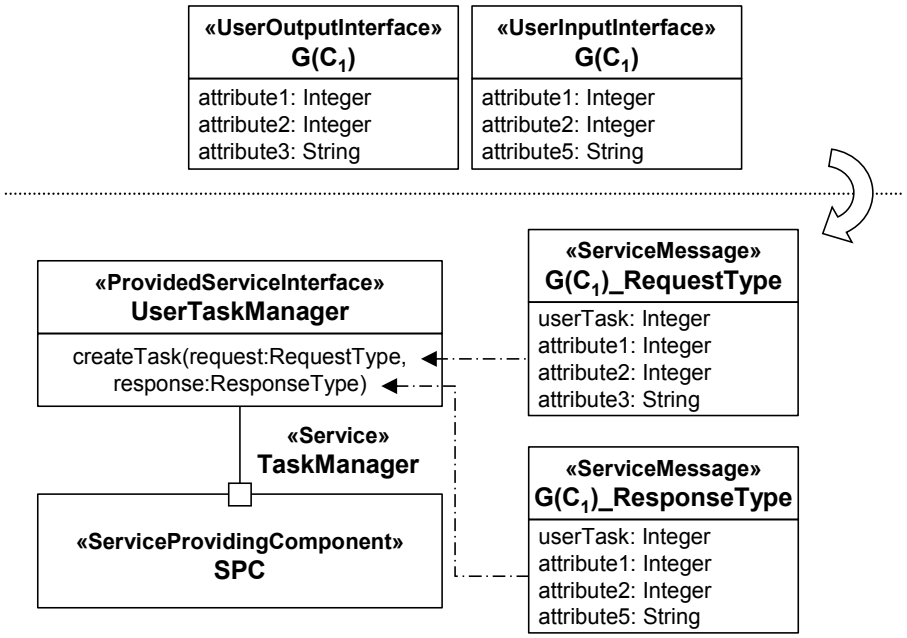

Abbildung 54: Beispielhafte Anwendung des Dienstprofils

Die als Dienst (`Service`) stereotypisierte Aufgabenverwaltung (`TaskManager`) wird durch eine diensterbringende Komponente (`ServiceProvidingComponent`) SPC realisiert. Die Aufgabenverwaltung stellt eine Dienstschnittstelle zur Benutzeraufgabenverwaltung (`UserTaskManager`) zur Verfügung (`ProvidedServiceInterface`), die über eine Dienstoperation zur Instanziierung von Benutzeraufgaben (`createTask`) verfügt. Als Eingabe- und Ausgabeparameter erwartet die Dienstoperation zwei Dienstnachrichten `request` und `response`, die durch die Datentypen `RequestType` und `ResponseType` näher spezifiziert sind. Diese Dienstnachrichten (`ServiceMessage`) können für Benutzeraktionen direkt aus den Benutzerschnittstellen des benutzerzentrischen Domänenmodells abgeleitet werden. Die ansonsten zur Umsetzung eines Workflows notwendige manuelle Spezifikation der Dienstnachrichten kann auf diese Weise vereinfacht werden. Wird ferner die in Abbildung 54 angedeutete Ableitung durch eine Modelltransformation automatisiert, so kann das bei einer manuellen Abbildung durch den Menschen vorhandene Risiko einer fehlerhaften Spezifikation der Dienstnachrichten reduziert und der zur Spezifikation der Dienstnachrichten eigentlich benötigte Entwicklungsaufwand eliminiert werden.

4.6 Resümee

Dieses Kapitel widmete sich der Konzeption unterschiedlicher Metamodelle, mittels derer die in dieser Arbeit betrachteten zentralen Anforderungen der Benutzerinteraktion durch Modelle spezifiziert und damit in einem modellgetriebenen Entwicklungsvorgehen integriert berücksichtigt werden können. In einem ersten Schritt wurden hierzu bestehende Metamodelle für Geschäftsprozesse für die Zwecke dieser Arbeit angepasst und erweitert. Ein wesentlicher Mehrwert dieser Anpassung liegt in der Möglichkeit einer differenzierten Unterscheidung der einzelnen Arbeitseinheiten eines Geschäftsprozesses. Durch das entwickelte **Benutzeraktionsprofil** kann eine Arbeitseinheit präzise als Benutzer-, System- oder manuelle Aktionen spezifiziert werden. Auf einem dementsprechend verfeinerten Workflow-Modell aufbauend wurde in Abschnitt 4.3 ein Metamodell zur Spezifikation von Benutzeraufgaben konzipiert, welches insbesondere die Ressourcenperspektive eines Workflows fokussiert. Das hieraus resultierende **Benutzeraufgabenprofil** zeigt exemplarisch, wie die Ausdrucksmächtigkeit bestehender Spezifikationssprachen erweitert werden kann, um so die spezifischen Anforderungen der

Ressourcenperspektive eines Workflows durch Modelle erfassen und eine Überwachung menschlicher Interaktion ermöglichen zu können. Abschnitt 4.4 adressierte durch die Einführung zweier Metamodelle, mittels derer Benutzerschnittstellen, die zur Interaktion eines Benutzers mit einem Anwendungssystem notwendig sind, durch Modelle spezifiziert werden können, einen weiteren Aspekt der Benutzerinteraktion. Durch das **Domänenprofil** und das **Strukturprofil** kann die inhaltliche und strukturelle Spezifikation einer Benutzerschnittstelle in das modellgetriebene Entwicklungsvorgehen integriert werden. Die durch das Domänenprofil im benutzerzentrischen Domänenmodell spezifizierten Benutzerschnittstellen können, wie in Abschnitt 4.5 demonstriert wurde, im modellgetriebenen Entwicklungsvorgehen unter Zuhilfenahme des **Dienstprofils** hinsichtlich der Spezifikation der zur Unterstützung der Benutzerinteraktion notwendigen Dienste und deren Dienstoperationen und Dienstnachrichten weiter genutzt werden.

Die konzipierten Metamodelle ermöglichen gemäß der in Abschnitt 3.1 festgehaltenen Anforderung A1.1 eine integrierte Betrachtung der Anforderungen der Benutzerinteraktion in einem gemeinsamen Vorgehensmodell. Gleichzeitig abstrahieren die Metamodelle von technologischen Details einer spezifischen Plattform und erlauben somit eine plattformunabhängige Betrachtung der Benutzerinteraktion (A1.2). Durch die explizite Berücksichtigung der Ressourcenperspektive durch das Benutzeraufgabenmodell und der Datenperspektive durch das domänenzentrische Benutzerschnittstellenmodell können die wesentlichen Perspektiven eines Workflows durch die Modelle berücksichtigt und somit die zentralen Geschäftsanforderungen auf die Modelle abgebildet werden (A1.3). Das zweistufige Modellierungsvorgehen berücksichtigt die für diese Arbeit zentralen Aspekte einer Benutzerschnittstelle, deren Anforderungen durch die bereitgestellten Modellelemente spezifiziert werden können (A1.4). Zusätzlich gestattet die getrennte Betrachtung der unterschiedlichen Aspekte der Benutzerinteraktion eine zielgerichtete und effektive Integration der Fachentwickler (A1.5).

Die konzipierten Modelle ermöglichen eine Berücksichtigung der unterschiedlichen Aspekte der Benutzerinteraktion in einem modellgetriebenen Entwicklungsvorgehen. Um dessen Komplexität zu reduzieren, sieht diese Arbeit eine Automatisierung bei der Erzeugung der benötigten Modelle vor. Der Entwicklung der hierzu notwendigen Modelltransformationen widmet sich das folgende Kapitel.

5 Automatisierte Erzeugung der Softwareartefakte der Benutzerinteraktion

Die im vorangegangenen Kapitel konzipierten Metamodelle gestatten es, unterschiedliche Aspekte der Benutzerinteraktion im Rahmen eines modellgetriebenen Entwicklungsvorgehens zu berücksichtigen. Eine weitere Herausforderung liegt in der automatisierten Umsetzung der entwickelten Spezifikationen entlang eines modellgetriebenen Entwicklungsvorgehens bis auf Softwareartefakte, die im Rahmen der IT-Unterstützung ausgeführt werden können. Dieses Kapitel widmet sich daher im ersten Teil der Entwicklung von Transformationen, die für eine automatisierte Erzeugung der Modelle benötigt werden. Im zweiten Teil steht die Konzeption einer geeigneten dienstorientierten Architektur im Mittelpunkt, deren als Dienste angebotene, fachfunktionale Komponenten die Ausführung der erzeugten Softwareartefakte ermöglichen. Der folgende Abschnitt liefert einen Überblick über die in diesem Kapitel entwickelten Beiträge und zeigt deren Zusammenhang mit dem Ziel einer automatisierten Erzeugung von Softwareartefakten für eine dienstorientierte Architektur auf.

5.1 Beiträge des Kapitels im Überblick

Das Modell eines Workflows diente im vorangegangenen Kapitel als Ausgangspunkt des modellgetriebenen Entwicklungsvorgehens, entlang dessen die unterschiedlichen Aspekte der Benutzerinteraktion durch jeweils eigenständige Modelle konkretisiert wurden (Anforderung A1.1 - A1.5, Seite 51). Die zwischen den Modellen bestehenden Abbildungsbezüge wurden bisher rein manuell betrachtet. Eine manuelle Abbildung der Modelle bringt jedoch Nachteile mit sich: Einerseits sind manuelle Abbildungen fehleranfällig, da sie durch den Menschen ausgeführt werden müssen, andererseits benötigen manuelle Abbildungen Zeit. Da gerade in frühen, iterativen Phasen eines Entwicklungsvorgehens Modelle kontinuierlich an geänderte oder verfeinerte Anforderungen angepasst und die daraus resultierenden Änderungen auf die jeweiligen Folgemodelle propagiert werden müssen, stellt die in diesem Kapitel angestrebte automatisierte Erzeugung der Modelle einen wesentlichen Mehrwert dar (Anforderung A1.6).

Analog zu Kapitel 4 dient das plattformunabhängige Modell eines Workflows als Ausgangspunkt des modellgetriebenen Entwicklungsvorgehens, auf dessen Grundlage Abschnitt 5.2 die zur Erzeugung eines Benutzeraufgabenmodells, eines benutzerzentrischen Domänenmodells und eines Strukturmodells benötigten Transformationen einführt. Alle drei Zielmodelle der in Abbildung 55 als T1 - T3 bezeichneten Modell-zu-Modell-Transformationen stellen keinen Bezug zu einer konkreten Plattform her und sind daher im Kontext der modellgetriebenen Architektur weiterhin als plattformunabhängige Modelle anzusehen. Neben der Erläuterung der wesentlichen Konzepte zur Erzeugung der plattformunabhängigen Modelle zeigt Abschnitt 5.2 ebenso die Umsetzung der Transformationen T1 - T3 durch den deklarativen Sprachteil QVT-*Relations* der in Abschnitt 2.1.3.2 eingeführten Transformationssprache *Query, View, Transformation* (QVT).

Um im nächsten Schritt des verfolgten modellgetriebenen Entwicklungsvorgehens die Erzeugung der plattformspezifischen Modelle aus den plattformunabhängigen Modellen ermöglichen zu können, müssen die im Zusammenhang mit der angestrebten dienstorientierten Architektur verwendeten plattformspezifischen Technologien und Standards zunächst konkretisiert werden. Da bestehende Referenzmodelle dienstorientierter Architekturen die Interaktion des Menschen jedoch nicht per se unterstützen, führt Abschnitt 5.3 zunächst eine Erweiterung dieser Referenzmodelle um zusätzliche,

als Dienste bereitgestellte fachfunktionale Komponenten auf der Basis bestehender Forschungsansätze und Standards ein und konkretisiert damit mögliche Zielplattformen der zu entwickelnden Transformationen. Abschnitt 5.4 erläutert anschließend beispielhaft die Entwicklung der Modell-zu-Modell-Transformation T4, die zur automatisierten Abbildung eines plattformunabhängigen Benutzeraufgabenmodells auf ein plattformspezifisches Benutzeraufgabenmodell benötigt wird. Um aus dem plattformspezifischen Benutzeraufgabenmodell im Anschluss ausführbare Softwareartefakte erzeugen zu können, wird eine zusätzliche, als T5 bezeichnete Modell-zu-Text-Transformation benötigt, die ebenfalls in Abschnitt 5.4 erläutert wird. Abbildung 55 gibt einen Überblick über die im Folgenden entwickelten Transformationen.

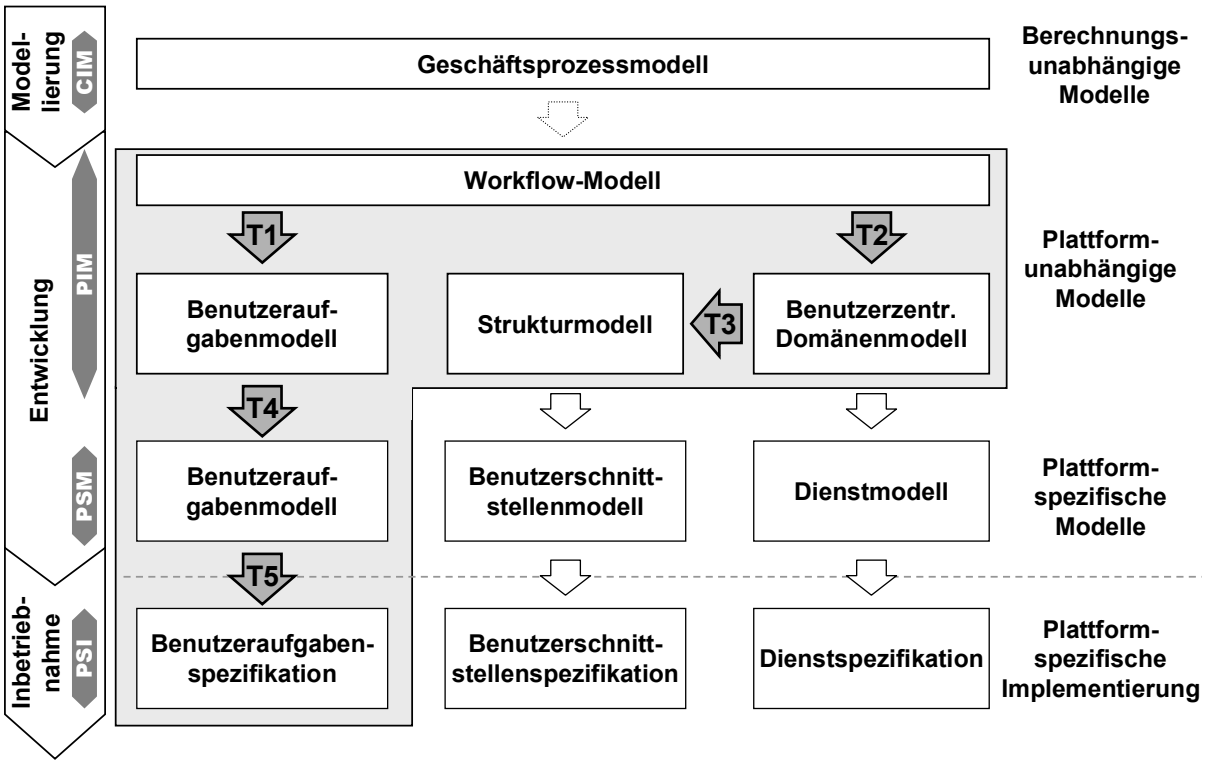

Abbildung 55: Entwickelte Transformationen im Überblick

Gemäß dem dargestellten modellgetriebenen Entwicklungsvorgehen steht zuerst die Erzeugung der plattformunabhängigen Modelle im Mittelpunkt, die der folgende Abschnitt einführt.

5.2 Automatisierte Erzeugung der plattformunabhängigen Modelle

Die im weiteren Verlauf dieses Abschnitts entwickelten Modell-zu-Modell-Transformationen T1 - T3 beruhen auf dem gleichen Transformationsvorgehen unter der Verwendung von QVT-*Relations*. Aus diesem Grund sei das generell verfolgte Vorgehen zur Entwicklung einer Transformation in QVT-*Relations* im folgenden Abschnitt kurz erläutert.

5.2.1 Einführung in das verfolgte Transformationsvorgehen

Um eine automatisierte Erzeugung der Modelle der Benutzerinteraktion zu erreichen, nutzt diese Arbeit das aus der modellgetriebenen Softwareentwicklung bekannte Konzept der Modelltransformationen. Eine Modelltransformation überführt ein Quellmodell in ein Zielmodell durch eine berechenbare Abbildung unter Zuhilfenahme einer Transformationsausführungsmaschine. Hierzu müssen die Transformationsregeln in einer für die Transformationsausführungsmaschine verarbeitbaren formalen

Sprache beschrieben werden. Zur Formalisierung der Transformationen kommt die Transformationssprache *Query, View, Transformation* (QVT) zum Einsatz, die als Standard der *Object Management Group* (OMG) publiziert ist [OMG-QVT]. Wie in Abschnitt 2.1.3.2 eingeführt, stellt QVT drei unterschiedliche Sprachteile zur Verfügung, von denen diese Arbeit den deklarativen Sprachteil QVT-*Relations* zur Spezifikation der Transformationsregeln nutzt. Da die folgenden Abschnitte auf verschiedenen in QVT-*Relations* verfassten Transformationen aufbauen, sei der prinzipielle Aufbau einer Transformation an einem kurzen Beispiel erläutert.

Die Spezifikation einer Transformation in QVT-*Relations* beginnt mit dem Schlüsselwort `transformation`, gefolgt von einem Bezeichner. Im nachfolgend angeführten Beispiel trägt die Transformation den Bezeichner `A_To_B` und erwartet zwei als `modelA` und `modelB` bezeichnete UML-Modelle als Eingabe. Auf den übergebenen Modellen können in einer Transformation mehrere Relationen über das Schlüsselwort `relation` definiert werden. Diese werden dazu verwendet, eine Gleichheit zwischen Modellen entweder zu überprüfen oder diese Gleichheit durch die Veränderung der Modelle herzustellen. Eine Relation setzt sich daher aus mindestens zwei Domänen zusammen, welche die für die Relation relevanten Teile der Modelle umfassen. Ferner besteht die Möglichkeit, eine Relation mit dem Schlüsselwort `top` besonders auszuzeichnen. Während eine *Top-Relation* nach der Ausführung einer Transformation immer gelten muss, ist die Gültigkeit aller weiteren Relationen nur dann gegeben, wenn diese direkt oder indirekt in einer *Top-Relation* verwendet werden. Im gegebenen Beispiel umfasst die *Top-Relation* `Model_To_Model` die beiden Domänen `modelA` und `modelB`, deren Muster über ihrem Metamodell UML spezifiziert sind.

```
transformation A_To_B (modelA:uml, modelB:uml) {

  top relation Model_To_Model {

    -- Simple definition of a variable modelName as String
    modelName : String;

    checkonly domain modelA sourceModel : uml::Model {
      name = modelName
    };

    enforce domain modelB targetModel : uml::Model {
      name = modelName
    };

    when { ... }
    where { ... }
  }
}
```

Eine Transformation hat prinzipiell keine fest vorgegebene Ausführungsrichtung. Die angegebene Transformation `A_To_B` kann somit die Domäne `modelA` als Quelle und die Domäne `modelB` als Ziel haben oder umgekehrt. Ob die Gültigkeit einer Relation zwischen einem Quellmodell und einem Zielmodell erzwungen wird, kann durch die beiden Schlüsselwörter `checkonly` und `enforce` spezifiziert werden. Wird eine Transformation in Richtung einer Domäne ausgeführt, die mit dem Schlüsselwort `enforce` gekennzeichnet ist, so wird die Gültigkeit der Relation hergestellt, auch wenn dazu eine Änderung des Zielmodells notwendig ist. Wird eine Transformation in Richtung einer

Domäne ausgeführt, die als `checkonly` markiert ist, so wird lediglich überprüft, ob ein entsprechendes Muster in der Zieldomäne existiert, und dementsprechend ein boolescher Wert `true` oder `false` zurückgegeben. Im angeführten Beispiel wird das Zielmodell, sofern die Domäne `modelB` das Ziel der Transformation ist, an die Relation angepasst, da diese Domäne mit dem Schlüsselwort `enforce` ausgezeichnet ist.

Um die Menge der für eine Relation relevanten Modellelemente konkreter zu bestimmen, können auf Domänen sogenannte Domänenmuster angegeben werden. Nur diejenigen Modellelemente, die dem Domänenmuster entsprechen, werden durch die Relation betrachtet. Im Beispiel sind die Domänenmuster sehr einfach gehalten: Auf Domäne `modelA` sind alle Modelle qualifiziert, die über einen Namen verfügen. Wird die Transformation in Richtung von `modelB` ausgeführt, so überprüft die Relation `Model_To_Model`, ob in Domäne `modelB` bereits ein Modell vorhanden ist, das den Namen eines gefundenen Modells aus Domäne `modelA` trägt. Ist dem nicht so, wird das Modell neu erzeugt. Diese Untersuchung wird solange wiederholt, bis die Relation für alle Modelle aus Domäne `modelA` erfüllt ist. Neben der Angabe von Domänenmustern gestattet QVT-*Relations* in einer Relation die Verwendung einer *when*- und einer *where*-Klausel. Eine *when*-Klausel umfasst Bedingungen, die neben dem spezifizierten Domänenmuster auf dem Quellmodell gelten müssen. Eine *where*-Klausel dagegen spezifiziert Bedingungen, die über alle an der Relation beteiligten Modellelemente (also Quell- und Zielmodell) Gültigkeit haben müssen.

Da die beispielhaft angeführte konkrete Syntax von QVT-*Relations* gerade bei größeren Transformationen für den Menschen nicht zwingend intuitiv erfassbar ist, stellt der QVT-Standard eine eigene grafische Notation als konkrete Syntax zur Spezifikation von Relationen bereit. Abbildung 56 zeigt ein an [OMG-QVT] angelehntes Beispiel für die Anwendung dieser grafischen Notation, die im weiteren Verlauf zur Spezifikation der benötigten Relationen durchgängig Verwendung finden soll. Die dargestellte Relation `Model_To_Model` entspricht der im Beispiel angeführten gleichnamigen *Top-Relation*, die Abkürzungen `c` und `e` entsprechen den eingeführten Schlüsselwörtern `checkonly` und `enforce`. Wird die Transformation, zu der die dargestellte Relation gehört, wie angedeutet von `source` nach `target` ausgeführt, so wird das rechts durch die Domäne `targetModel` repräsentierte Zielmodell an das links dargestellte Quellmodell angepasst.

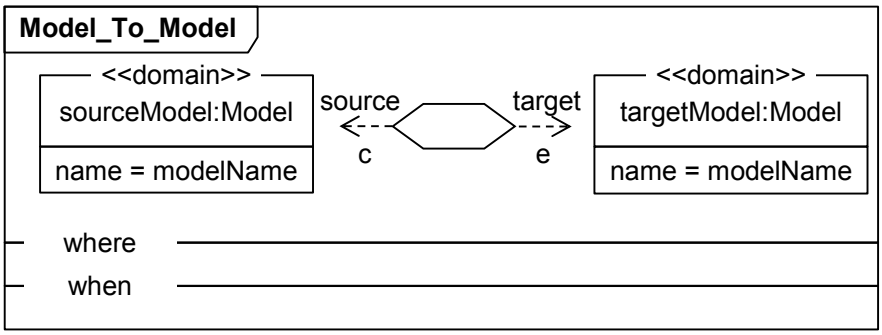

Abbildung 56: Grafische Notation einer Relation in QVT-Relations

Das eingeführte Transformationsvorgehen kommt in den folgenden Abschnitten zur Spezifikation der benötigten Relationen der zu entwickelnden Transformationen zum Einsatz. Ziel und Mehrwert dieser Transformationen liegen in einer automatisierten Erzeugung aller in dieser Arbeit betrachteten Modelle auf Basis ihrer Quellmodelle. Durch Transformationen werden manuelle, durch den Menschen durchgeführte Abbildungen im modellgetriebenen Entwicklungsvorgehen unnötig, wodurch das Risiko einer Fehlinterpretation der Modelle gesenkt und gleichzeitig die benötigte Zeit für die Abbil-

dung eines Modells reduziert werden kann. Gemäß des in dieser Arbeit verfolgten modellgetriebenen Entwicklungsvorgehens steht, wie in Abbildung 55 illustriert, die automatisierte Erzeugung eines Benutzeraufgabenmodells, des benutzerzentrischen Domänenmodells und des Benutzerschnittstellenmodells aus einem Workflow-Modell als Quellmodell im Mittelpunkt dieses Abschnittes. Analog zum verfolgten Vorgehen in Kapitel 4 wird zunächst die zur Erzeugung des Benutzeraufgabenmodells notwendige Modell-zu-Modell-Transformation T1 beleuchtet.

5.2.2 Erzeugung des Benutzeraufgabenmodells

Ein Workflow-Modell kann verschiedene Benutzeraktionen umfassen, die durch eine oder mehrere in Form von Geschäftsrollen spezifizierte menschliche Ressourcen bearbeitet werden müssen. Um den im Workflow-Modell bestehenden Zusammenhang zwischen den Geschäftsrollen und den Benutzeraktionen im Benutzeraufgabenmodell reflektieren zu können, muss eine erste Relation R1.1 der Transformation T1 für jedes Workflow-Modell ein eigenständiges Benutzeraufgabenmodell erzeugen, das alle Benutzeraufgaben, die aus den Benutzeraktionen eines Workflows heraus entstehen, aufnehmen kann. Aus diesem direkten Zusammenhang ergeben sich unterschiedlichste Vorteile. Beispielsweise kann auf diese Weise im Benutzeraufgabenmodell spezifiziert werden, dass eine konkrete menschliche Ressource R_1, die eine Benutzeraufgabe C_1 auszuführen hat, ebenfalls mit der Bearbeitung von Benutzeraufgabe C_2 zu beauftragen ist oder aber diese explizit nicht ausführen darf. Ferner muss eine Aufgabenrolle wie etwa ein Administrator nur einmal spezifiziert werden und kann dann verschiedenen Benutzeraufgaben gleichzeitig zugeordnet werden.

Eine zweite Relation R1.2 muss die in einem Workflow-Modell als Quellmodell bereits vorhandenen zahlreichen Informationen über die zu erzeugenden Benutzeraufgaben wiederverwenden. Wie in Abbildung 57 an einem Auszug des Beispiel-Workflows dargestellt, kann durch R1.2 für jede Benutzeraktion (`UserAction`) des Workflow-Modells eine Benutzeraufgabe (`UserTask`) des Benutzeraufgabenmodells erzeugt werden. Um den Bezug zwischen Benutzeraktion und Benutzeraufgabe kenntlich zu machen, übernimmt R1.2 den Bezeichner einer Benutzeraktion und ordnet diesen einer Benutzeraufgabe zu (siehe Abbildung 57, R1.2.1).

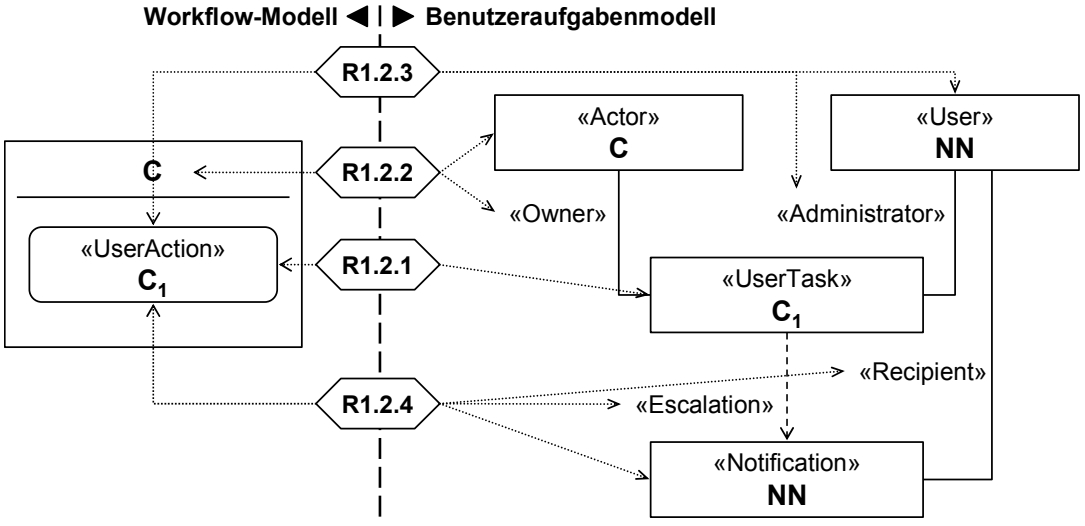

Abbildung 57: Entwurf der Relation R1.2 der Transformation T1

Jeder Benutzeraktion des Workflow-Modells ist ferner eine Geschäftsrolle (`BusinessRole`) zugeordnet, welche die Ausführung der Benutzeraktion übernehmen muss. Dementsprechend kann diese Information dazu genutzt werden, der Benutzeraufgabe eine menschliche Ressource in der

Aufgabenrolle des Eigentümers (`Owner`) zuzuordnen (R1.2.2). Das Workflow-Modell trifft allerdings keine genauere Aussage darüber, ob eine Geschäftsrolle nun eine organisatorische Einheit, eine Rolle oder einen Benutzer repräsentiert. Daher muss die Transformation die Geschäftsrolle zunächst auf das abstrakte Metamodellelement Akteur (`Actor`) abbilden. Um dennoch den Bezug zur Geschäftsrolle des Workflow-Modells im Benutzeraufgabenmodell widerzuspiegeln, übernimmt die R1.2 zusätzlich den Bezeichner der Geschäftsrolle als Bezeichner des Akteurs. Im Anschluss an die Transformation kann das erzeugte Modellelement Akteur durch die beteiligten Entwicklungsrollen mit einem konkreten Stereotyp wie beispielsweise Benutzer (`User`) ausgeprägt werden.

Neben den aus dem Workflow-Modell ableitbaren Details können aus dem Kontext einer Benutzeraufgabe zusätzliche Informationen für die Entwicklung der Relationen von Transformation T1 genutzt werden. Da jede Benutzeraufgabe zu ihrer Ausführungszeit überwacht werden muss, wird für jede Benutzeraufgabe ein Administrator benötigt, der diese Überwachung durchführen und die Instanzen der Benutzeraufgabe während der Ausführungszeit beeinflussen kann. Wie bereits erörtert, muss diese Aufgabenrolle aufgrund der notwendigen eindeutigen Zuständigkeit durch einen konkreten Benutzer übernommen werden und darf daher schon zur Entwurfszeit keiner organisatorischen Einheit oder Rolle zugeordnet werden. Demnach muss R1.2 für jede Benutzeraktion (`UserAction`) einen einzelnen Benutzer (`User`) erzeugen und diesen über die Aufgabenrolle Administrator (`Administrator`) mit der Benutzeraufgabe assoziieren (R1.2.3). Gleichzeitig werden zur Überwachung einer Benutzeraufgabe immer ein Eskalationsmechanismus und eine Eskalationsaufgabe benötigt, da ohne diese Maßnahmen eine Überwachung nicht sinnvoll durchführbar ist. R1.2 muss daher, wie in Abbildung 57 angedeutet, für jede Benutzeraufgabe die Eskalationsaufgabe Benachrichtigung (`Notification`) erzeugen und diese über die Aufgabenrolle Empfänger (`Recipient`) mit dem Benutzer, der durch die Aufgabenrolle Administrator mit der Benutzeraufgabe in Verbindung steht, assoziieren. Um eine Benachrichtigung zur Laufzeit auslösen zu können, ist für die Eskalationsaufgabe zusätzlich eine Assoziation zu der Benutzeraufgabe (`UserTask`) über die Aufgabenrolle Eskalation (`Escalation`) erforderlich. Die genauen Eigenschaftswerte der Eskalation und der Eskalationsaufgabe, wie beispielsweise eine späteste Startzeit von 14 Tagen nach Erzeugung der Benutzeraufgabeninstanz, könnten durch die Relation ebenfalls bereits festgelegt werden. Da diese Eigenschaftswerte jedoch sehr individuell auf jedes einzelne Anwendungsszenario abgestimmt werden müssen, erzeugt R1.2 lediglich das Gerüst zur Überwachung der Benutzeraufgabe, das im Anschluss an die Transformation durch die Entwickler im Modell mit konkreten Werten verfeinert werden muss.

Um die Erzeugung des Benutzeraufgabenmodells automatisiert durchführen zu können, muss die skizzierte Modell-zu-Modell-Transformation T1 formalisiert werden. Die folgenden beiden Abschnitte zeigen die Umsetzung der benötigten Relationen R1.1 und R1.2 durch QVT-*Relations*.

5.2.2.1 Relation R1.1 – Model_To_Model

Die entsprechend der grafischen Notation von QVT-*Relations* in Abbildung 56 dargestellte und als `Model_To_Model` bezeichnete Relation R1.1 der Transformation T1 erwartet zwei UML-Modelle als Eingabeparameter. Die Relation `Model_To_Model` sucht auf der als `checkonly` ausgezeichneten Domäne des Quellmodells alle Modelle mit einem Namen und erzeugt in der als `enforce` ausgezeichneten Domäne des Zielmodells ein neues Benutzeraufgabenmodell mit gleichem Bezeichner, sofern dieses noch nicht existiert. In der *where*-Klausel der Relation wird die Gültigkeit der Relation `UserAction_To_UserTask` (R1.2) gefordert und dieser die durch die Domänenmuster jeweils selektierten Modelle als Parameter übergeben.

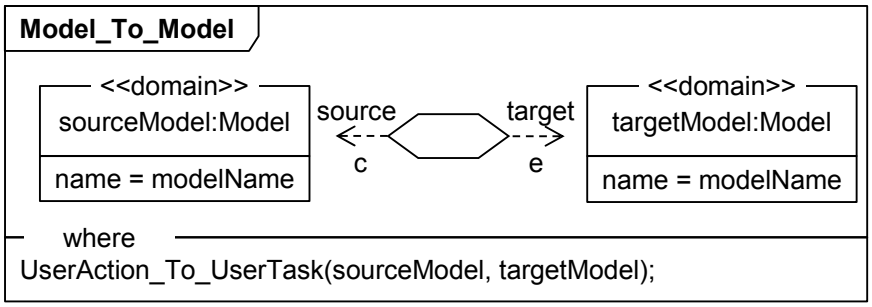

Abbildung 58: Model_To_Model-Relation in QVT-Relations

5.2.2.2 Relation R1.2 – UserAction_To_UserTask

Die in Abbildung 59 dargestellte Relation R1.2 stellt die Gleichheit zwischen Quell- und Zielmodell gemäß des in Abbildung 57 illustrierten Entwurfs her. Das auf dem Quellmodell spezifizierte Domänenmuster durchsucht das übergebene UML-Modell nach allen Aktionen, die durch den Stereotyp `UserAction` des Benutzeraufgabenprofils ausgeprägt wurden, und hält deren Bezeichner in der Variable `userActionName` fest. Gleiches gilt für die Aktivitätspartitionen (`ActivityPartition`), in denen die Benutzeraktionen spezifiziert sind (`inPartition`). Da deren Bezeichner den Geschäftsrollen des Workflow-Modells entsprechen, werden auch diese Bezeichner in einer Variable `partitionName` festgehalten. Auf dem Zielmodell erzeugt die Relation für jede Benutzeraktion des Quellmodells einen Akteur, der den Namen der Partition, in der die Benutzeraktion im Workflow-Modell enthalten ist, übernimmt. Ebenso werden alle benötigten Stereotypen aus dem Benutzeraufgabenprofil wie beispielsweise eine Benutzeraufgabe (`UserTask`) oder der Eigentümer (`Owner`) bereitgestellt. Da einige Details aus dem Quellmodell nicht abgeleitet werden können, erzeugt R1.2 einige Standardwerte wie etwa einen Bezeichner „NN“ für den Benutzer in der Aufgabenrolle des Administrators.

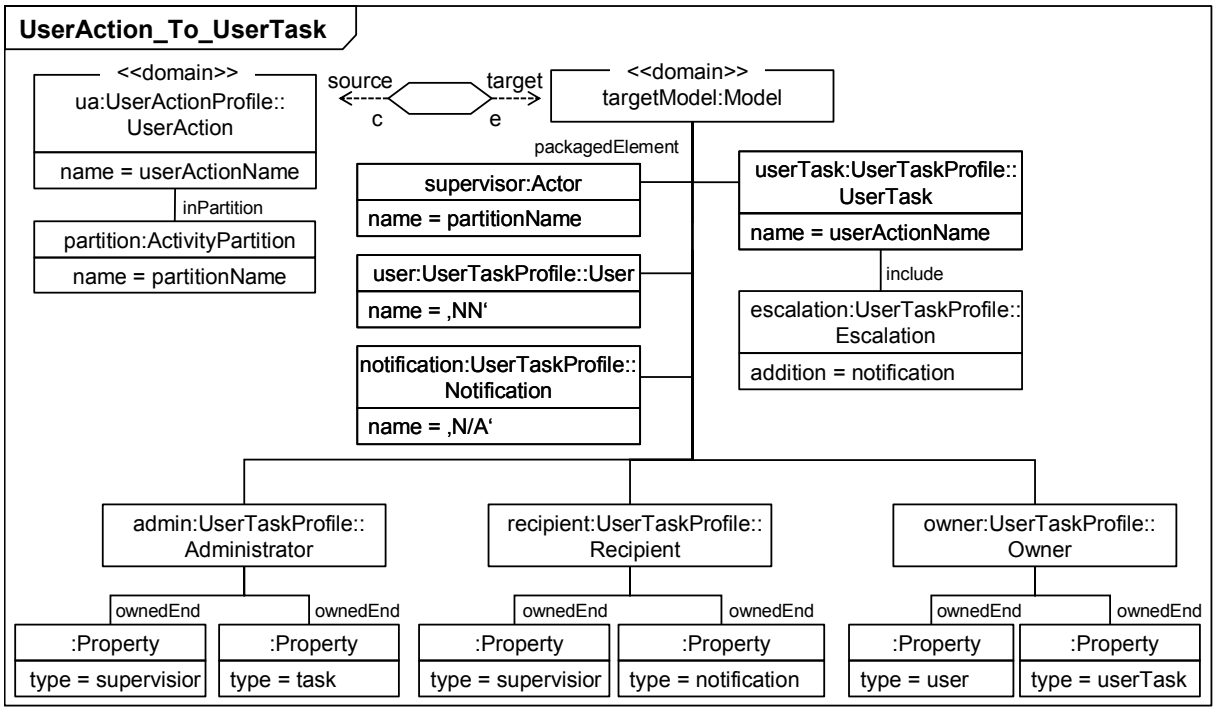

Abbildung 59: UserAction_To_UserTask-Relation in QVT-Relations

5.2.2.3 Demonstration der Anwendung

Die Anwendung der Modell-zu-Modell-Transformation T1 auf ein Workflow-Modell erzeugt ein gleichnamiges Benutzeraufgabenmodell, das im Anschluss an die Transformation von entsprechenden Experten um weitere Details angereichert werden muss. Abbildung 60 zeigt das Ergebnis der Anwendung der Transformation auszugsweise an der Benutzeraktion C_1 des Beispiel-Workflows. Erneut kommen hierbei die in Abschnitt 4.2.2 eingeführten eigenen grafischen Repräsentationen der Stereotypen des Benutzeraufgabenprofils zum Einsatz, die eine übersichtliche und entwicklerfreundliche Spezifikation eines Benutzeraufgabenmodells gestatten. Zusätzlich können die Entwickler das Benutzeraufgabenmodell je nach dem gegebenen Anwendungsszenario beliebig verändern und beispielsweise zusätzliche Eskalationsmechanismen hinzufügen oder weitere Aufgabenrollen wie etwa einen Delegierten spezifizieren.

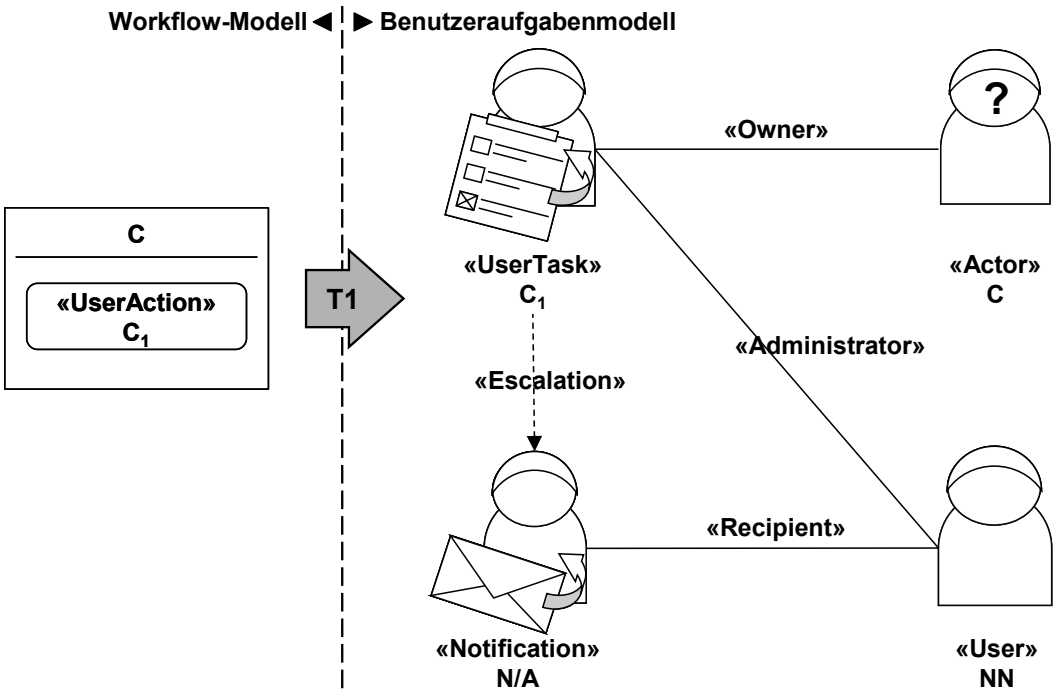

Abbildung 60: Ergebnis der Anwendung der Transformation T1

Das erzeugte Benutzeraufgabenmodell stellt noch keinerlei Bezug zu technischen Details einer Plattform her. Im Sinne der modellgetriebenen Architektur ist es daher als plattformunabhängig anzusehen und kann auf beliebige Zielplattformen ausgeprägt werden. Die automatisch erzeugten Benutzeraufgabenmodelle weisen folglich einen hohen Grad an Portabilität auf und können in unterschiedlichsten Kontexten wiederverwendet werden. Ein konkretes Beispiel für eine Ausprägung des Benutzeraufgabenmodells auf ein plattformspezifisches Modell wird im zweiten Teil dieses Kapitels in Abschnitt 5.4 gegeben. Entlang des verfolgten modellgetriebenen Entwicklungsvorgehens muss zuvor jedoch die Erzeugung aller weiteren plattformunabhängigen Modelle beleuchtet werden.

Als zweite vertikale Modell-zu-Modell-Transformation adressiert T2 daher die Abbildung des Workflow-Modells auf das benutzerzentrische Domänenmodell, welches zusammen mit dem Strukturmodell zur Spezifikation der Benutzerschnittstelle benötigt wird. Der folgende Abschnitt führt mit Transformation T2 die automatisierbare Erzeugung des benutzerzentrischen Domänenmodells ein, Abschnitt 5.2.4 widmet sich der Transformation T3, die das benutzerzentrische Domänenmodell als Quellmodell aufgreift und dieses zur Erzeugung des Strukturmodells der Benutzerschnittstelle verwendet.

5.2.3 Erzeugung des benutzerzentrischen Domänenmodells

Damit ein Benutzer im Rahmen einer Benutzeraktion mit den IT-Systemen der IT-Unterstützung interagieren kann, benötigt er eine geeignete Benutzerschnittstelle. Grundlage für die Erzeugung einer Benutzerschnittstelle sind die einzelnen Geschäftsobjekte, die in einem Workflow durch unterschiedliche Benutzeraktionen manipuliert und verarbeitet werden. Ausgangsmodell der Modell-zu-Modell-Transformation T2 ist daher erneut das plattformunabhängige Workflow-Modell.

Da in einem Workflow-Modell in unterschiedlichen Benutzeraktionen dasselbe Geschäftsobjekt bearbeitet werden kann, muss diesem Bezug dadurch Ausdruck verliehen werden, dass alle Benutzeraktionen eines Workflow-Modells auf ein gemeinsames benutzerzentrisches Domänenmodell abgebildet werden. Da die Eingabe- und Ausgabepins die in einer Benutzeraktion bearbeiteten Geschäftsobjekte repräsentieren, müssen alle Ein- und Ausgabepins einer Benutzeraktion zur Erzeugung von Benutzerschnittstellenelementen im benutzerzentrischen Domänenmodell verwendet werden. Abbildung 61 gibt einen Überblick über die hierzu benötigten Relationen R2.1 und R2.2 der Transformation T2.

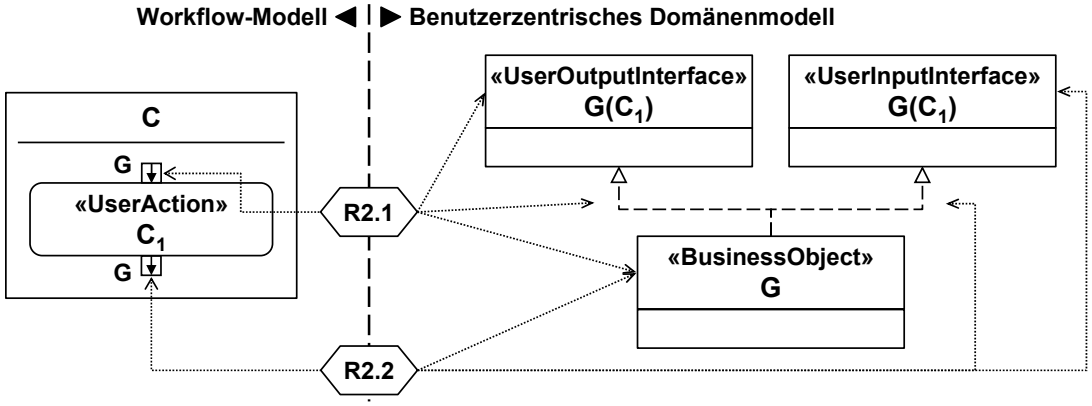

Abbildung 61: Entwurf der Relationen R2.1 und R2.2 der Transformation T2

Durch eine erste Relation R2.1 müssen die assoziierten Eingabepins (`InputPin`) einer Benutzeraktion (`UserAction`) und die dadurch spezifizierten Geschäftsobjekte auf den Stereotyp Benutzerausgabeschnittstelle (`UserOutputInterface`) abgebildet werden. Diese komplementäre Zuordnung von Eingabepin zu Benutzerausgabeschnittstelle mag im ersten Augenblick verwunderlich erscheinen. Da ein Eingabepin nach Spezifikation der UML [OMG-UML2-Super] jedoch genau diejenigen Eigenschaften spezifiziert, die einer Benutzeraktion zur Verfügung gestellt werden müssen, bevor mit der Ausführung der Benutzeraktion begonnen werden kann, liegen demnach die Werte dieser Eigenschaften zu Beginn der Ausführung vor und können dem Benutzer in der Benutzerschnittstelle angezeigt, sprich ausgegeben werden. Im umgekehrten Fall müssen die Werte der durch Ausgabepins spezifizierten Eigenschaften am Ende der Ausführung einer Benutzeraktion zur Verfügung stehen und daher durch einen Benutzer über eine Benutzereingabeschnittstelle eingegeben werden. Daher erzeugt R2.1 für jeden Eingabepin einer Benutzeraktion des Quellmodells eine Benutzerausgabeschnittstelle im Zielmodell.

Da dasselbe Geschäftsobjekt (`BusinessObject`) in unterschiedlichen Benutzeraktionen bearbeitet werden kann, muss der Bezeichner der Benutzerausgabeschnittstelle aus einer Kombination von Bezeichner der Benutzeraktion und Bezeichner des Geschäftsobjekts erzeugt werden. Ferner muss R2.1 überprüfen, ob das Geschäftsobjekt, das in einer Benutzeraktion bearbeitet werden soll, im benutzerzentrischen Domänenmodell bereits vorhanden ist. Ist dies der Fall, so muss das bereits

vorhandene Geschäftsobjekt mit der neuen Benutzerausgabeschnittstelle über eine Implementierungsassoziation in Bezug gestellt werden. Die UML sieht hierzu die in Abbildung 61 als gestrichelten Pfeil dargestellte gerichtete Assoziation `interfaceRealization` vor, welche diese Implementierungsbeziehung ausdrückt. Ist das Geschäftsobjekt beim Anlegen der Benutzerausgabeschnittstelle noch nicht vorhanden, dann muss es durch die Relation R2.1 neu angelegt und anschließend über die Implementierungsassoziation mit der Benutzerausgabeschnittstelle in Bezug gesetzt werden. Da R2.1 aus den Eingabepins einer Benutzeraktion im Workflow-Modell Benutzerausgabeschnittstellen im benutzerzentrischen Domänenmodell erzeugt, muss die zweite Relation R2.2 der Transformation T2 symmetrisch zu Relation R2.1 aufgebaut werden und dementsprechend für die Ausgabepins der Benutzeraktionen Benutzereingabeschnittstellen einfordern.

Um die Modell-zu-Modell-Transformation T2 automatisiert ausführen zu können, soll auch diese Transformation durch die Transformationssprache QVT formalisiert werden. Um sicherstellen zu können, dass bei der automatisierten Ausführung der Transformation keine Duplikate im Zielmodell erzeugt werden, müssen in der Transformationsspezifikation zunächst sogenannte *key*-Direktiven vorgesehen werden. Durch die *key*-Direktiven wird festgelegt, wie erzeugte Modellelemente eindeutig identifiziert und damit Duplikate verhindert werden können.

```
key UML::Model {name};
key DomainProfile::BusinessObject {name};
```

Da alle Benutzeraktionen eines Workflow-Modells auf ein benutzerzentrisches Domänenmodell abgebildet werden, würden ohne die erste *key*-Direktive mehrere Zielmodelle erzeugt, die alle den gleichen Bezeichner des Quellmodells tragen würden. Dies wird mit der Festlegung, dass ein Modell eindeutig über den Namen identifiziert werden kann, durch die *key*-Direktive unterbunden. Wird ferner ein Geschäftsobjekt G (`BusinessObject`) in einem Workflow-Modell durch mehrere Benutzeraktivitäten bearbeitet, so verhindert die zweite dargestellte *key*-Direktive anhand des Bezeichners (`name`) der Modellelemente, dass beispielsweise das Geschäftsobjekt G im benutzerzentrischen Domänenmodell (Zielmodell) mehrfach angelegt wird. Eine eindeutige Zuordnung der Benutzerschnittstellen zu einem Geschäftsobjekt wird somit ermöglicht. Neben diesen vorbereitenden Spezifikationen müssen zusätzlich die beiden entworfenen Relationen R2.1 und R2.2 durch die Transformation T2 umgesetzt werden.

5.2.3.1 Relation R2.1 – InputPin_To_UserOutputInterface

Die in Abbildung 62 dargestellte Relation `InputPin_To_UserOutputInterface` (R2.1) erwartet zwei UML-Modelle als Eingabe. Wird die Transformation entsprechend der durch die beiden Domänenbezeichner `source` und `target` angezeigten Richtung ausgeführt, so wird auf das Quellmodell das im linken Teil dargestellte Domänenmuster zur Suche nach qualifizierten Modellelementen angewandt. Von den Benutzeraktionen des Quellmodells (das Workflow-Modell) aus werden die Benutzeraktionen ausgewählt, die über einen Eingabepin verfügen. Gleichzeitig wird dabei der Bezeichner der Benutzeraktion in der Variablen `uaName` sowie der Bezeichner des Eingabepins, der dem Namen des zu bearbeitenden Geschäftsobjekts entspricht, in der Variablen `boName` festgehalten. Das Domänenmuster des Zielmodells (das benutzerzentrische Domänenmodell) überprüft zunächst, ob überhaupt schon ein Zielmodell mit dem Bezeichner des Quellmodells vorhanden ist, und legt ein neues Modell mit dem Bezeichner `modelName` an, falls dies nicht der Fall ist. Zusätzlich wird für jeden Eingabepin überprüft, ob im Zielmodell bereits eine Benutzerausgabeschnittstelle, ein Geschäftsobjekt und die Implementierungsassoziation vorhanden sind.

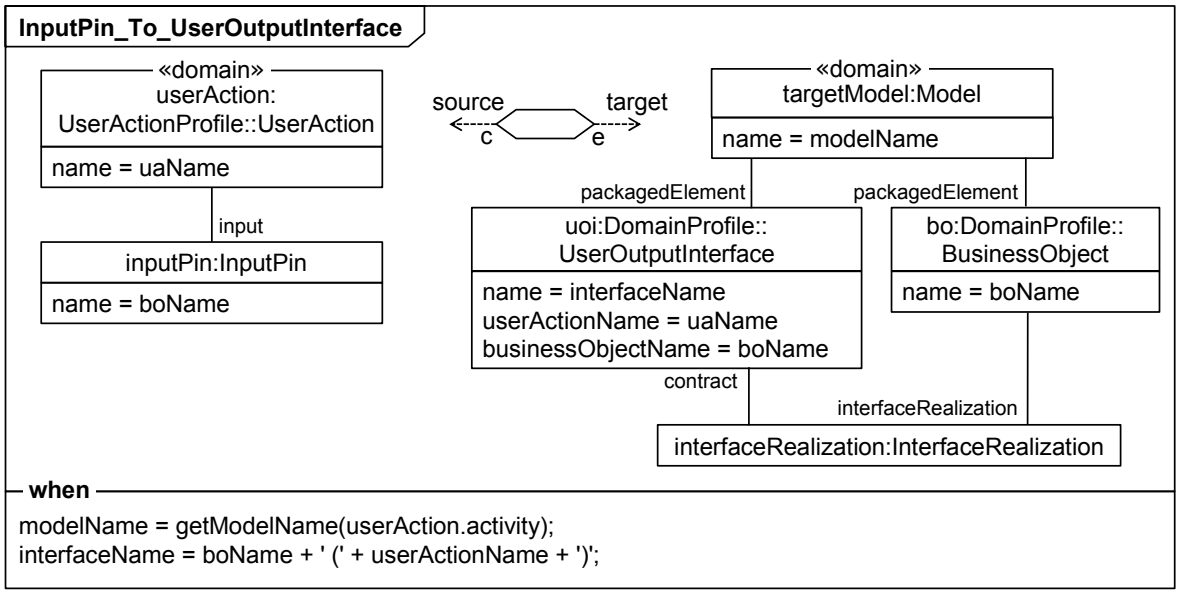

Abbildung 62: InputPin_To_UserOutputInterface-Relation in QVT-Relations

Um die ursprüngliche Zuordnung der Benutzerschnittstellen zu deren Benutzeraktionen und Geschäftsobjekten als wichtige Information für alle folgenden Modelle erhalten zu können, verfügt sowohl die Benutzerausgabe- als auch die Benutzereingabeschnittstelle über die beiden Eigenschaften `userActionName` und `businessObjectName` (vgl. Abschnitt 4.4), die diese Informationen durch die in der Relation vorgesehenen Zuweisungen aufnehmen. Um an diese Informationen zu gelangen, muss jedoch berücksichtigt werden, dass eine Benutzeraktion in einer beliebigen Tiefe in UML-Aktivitäten enthalten sein kann. Daher benötigt R2.1 eine Hilfsmethode, die von der Benutzeraktion aus rekursiv über die vorhandenen Aktivitäten bis zu einem Modell aufsteigt und den Namen des Modells zurückliefert. Die nachfolgend dargestellte Hilfsmethode `getModelName` liefert diese Funktionalität, indem sie das übergebene Element auf seine Eigenschaften überprüft. Liegt ein Modell vor, so wird der Name des Modells zurückgegeben und ansonsten die Hilfsmethode rekursiv auf dem Elternelement aufgerufen.

```
query getModelName(pElement : uml::packageableElement) : String
{
  if(pElement.owningPackage.oclIsTypeOf(uml::Model))
  then
    pElement.owningPackage.name
  else
    getModelName(pElement.owningPackage)
  endif
}
```

5.2.3.2 Relation R2.2 – OutputPin_To_UserInputInterface

Die zweite Relation `OutputPin_To_UserInputInterface` (R2.2) ist nahezu symmetrisch zu R2.1 und unterscheidet sich lediglich in der Tatsache, dass R2.2 alle Ausgabepins (`OutputPins`) eines Workflow-Modells aufgreift und diese auf Benutzereingabeschnittstellen (`UserInputInterface`) überführt. Dementsprechend ähneln sich beide Relationen im groben Aufbau sehr, unterscheiden sich aber im Detail.

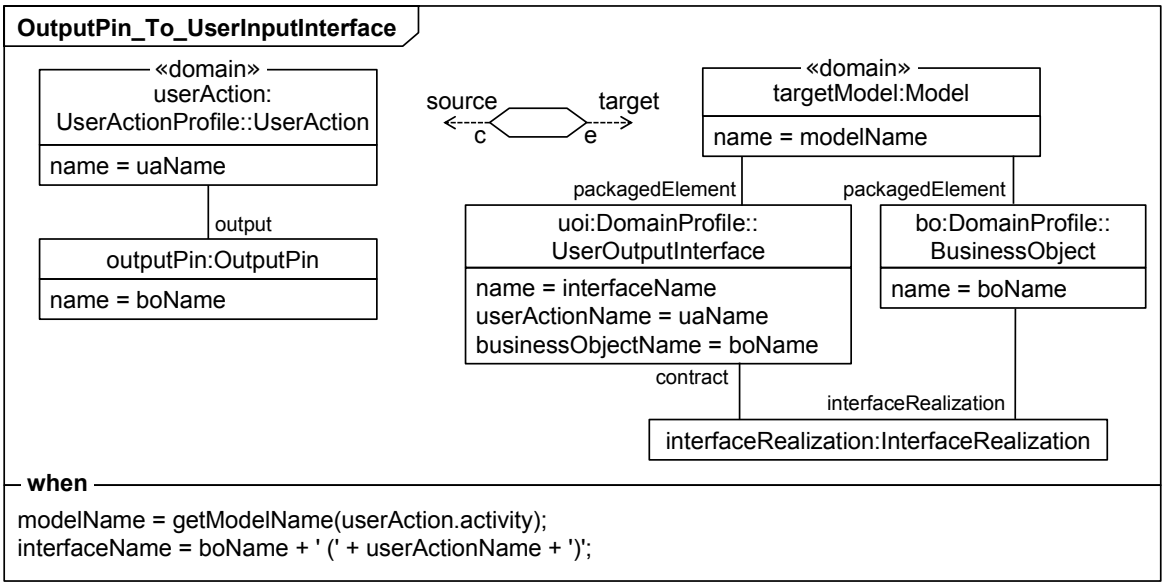

Abbildung 63: OutputPin_To_UserInputInterface-Relation in QVT-Relations

5.2.3.3 Demonstration der Anwendung

Transformation T2 erzeugt für alle Ein- und Ausgabepins der Benutzeraktionen eines Workflow-Modells durch die spezifizierten Relationen R2.1 und R2.2 Benutzerschnittstellen in einem benutzerzentrischen Domänenmodell. Da in einem Workflow unterschiedliche Benutzeraktionen Zugriff auf ein Geschäftsobjekt benötigen können, entsteht durch den verfolgten Ansatz, alle Benutzeraktionen auf ein gemeinsames benutzerzentrisches Domänenmodell abzubilden, ein zentrales Modell, das in übersichtlicher Weise den Zugriff auf die vorhandenen Geschäftsobjekte abbildet. Ferner kann durch diese Art der Modellierung für jede Benutzerschnittstelle der Zugriff auf ein Geschäftsobjekt genauestens festgelegt werden.

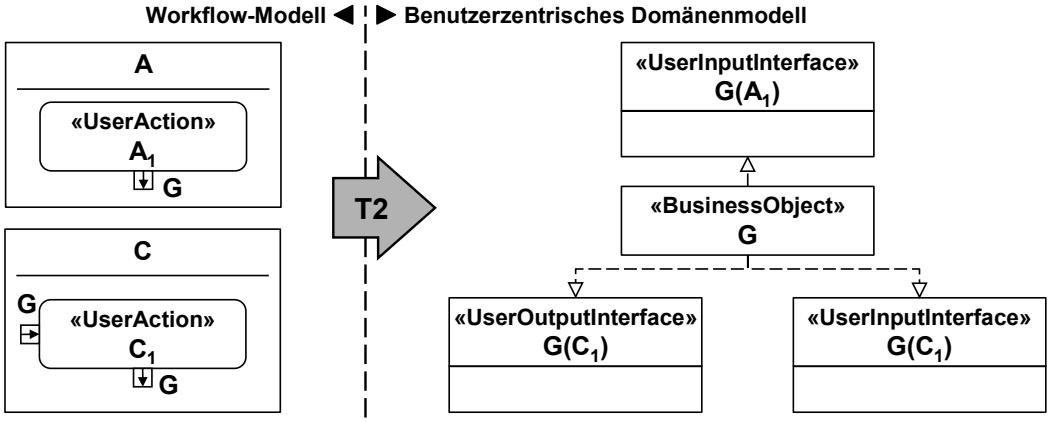

Abbildung 64: Ergebnis der Anwendung von Transformation T2

Die Anwendung der Transformation T2 resultiert zunächst jedoch, wie in Abbildung 64 dargestellt, in einem benutzerzentrischen Domänenmodell, das als Gerüst noch keine Detailinformationen über die in den einzelnen Benutzerschnittstellen benötigten Eigenschaften der Geschäftsobjekte enthält. Diese Detailinformationen können erst in Gesprächen mit dem Kunden gewonnen werden. Erfolgt aufgrund der Diskussion mit dem Kunden die Zuordnung einer Eigenschaft zu einer Benutzerausgabeschnittstelle, dann muss der Wert dieser Eigenschaft dem Benutzer zur Laufzeit zur Anzeige gebracht werden. Wird der Benutzerausgabeschnittstelle $G(C_1)$ beispielsweise die Eigenschaft „Name“ zuge-

ordnet und ist G ein Studierender, muss in Benutzeraktion C_1 dem Bearbeiter folglich der Name des Studierenden angezeigt werden. Wird die Eigenschaft auch der Benutzereingabeschnittstelle $G(C_1)$ zugeordnet, dann erhält der Bearbeiter zusätzlich das Recht, den Wert der Eigenschaft in der ihm vorgelegten Instanz des Geschäftsobjekts „Studierender“ zu bearbeiten.

Das durch die Transformation T2 erzeugte und in Zusammenarbeit mit dem Kunden[4] verfeinerte benutzerzentrische Domänenmodell dient im nächsten Entwicklungsschritt als Quellmodell der Modell-zu-Modell-Transformationen T3, die für eine automatisierte Erzeugung eines plattformunabhängigen Strukturmodells einer Benutzerschnittstelle aus einem benutzerzentrischen Domänenmodell benötigt wird (vgl. Abbildung 55, Seite 130).

5.2.4 Erzeugung des Strukturmodells

Das zuvor betrachtete plattformunabhängige benutzerzentrische Domänenmodell umfasst in verschiedenen Benutzereingabe- und Benutzerausgabeschnittstellen die Eigenschaften der Geschäftsobjekte, die durch einen Benutzer über eine Benutzerschnittstelle im Rahmen einer Benutzeraktion bearbeitet werden können. Da eine Benutzeraktion und die daraus resultierende Benutzeraufgabe als eine in sich abgeschlossene Arbeitseinheit anzusehen ist (vgl. Abschnitt 2.2.2), erscheint es sinnvoll, die Benutzerschnittstelle zu einer Benutzeraktion ebenfalls als eine Einheit zu betrachten und daher jeder Benutzeraktion eine eigene Benutzerschnittstelle zuzuordnen. Dementsprechend muss eine erste Relation R3.1 der Transformation T3 die Benutzereingabe- und Benutzerausgabeschnittstellen aus dem benutzerzentrischen Domänenmodell extrahieren und diese pro Benutzeraktion in ein eigenständiges Strukturmodell überführen. Die Information, zu welcher Benutzeraktion eine Benutzereingabe- oder Benutzerausgabeschnittstelle gehört, ist in den Eigenschaften beider Stereotypen hinterlegt (vgl. Abschnitt 4.4.1.2). Da ferner alle erzeugten Strukturmodelle über das Workflow-Modell als Ursprungsmodell in Bezug zueinander stehen, können diese übergeordneten gemeinsamen Modellelemente zugeordnet werden. Das UML-Metamodellelement Paket (`Package`) ist aufgrund seiner Definition in der UML-*Superstructure* hierfür geeignet (vgl. [OMG-UML2-Super]). Hierdurch entsteht pro Benutzeraktion ein eigenes Strukturmodell, das je Workflow einem gemeinsamen Paket zugeordnet werden kann. Somit können auf einfache Art und Weise die Strukturmodelle pro Workflow an Fachentwickler beispielsweise zur Verfeinerung weitergegeben werden. Da ein Benutzer immer über die Benutzerschnittstelle gegenüber der IT-Unterstützung das Ende der Bearbeitung seiner Benutzeraufgabe anzeigen können muss, erzeugt R3.1 abschließend für jedes Strukturmodell ein entsprechendes Aktionselement.

Eine zweite und dritte Relation R3.2 und R3.3 müssen, wie in Abbildung 65 skizziert, die Eigenschaften der Benutzereingabe- und Benutzerausgabeschnittstellen in Modellelemente des Strukturprofils überführen. Dabei kann der hierarchischen Struktur einer Benutzerschnittstelle Ausdruck verliehen werden, indem jede Eigenschaft einer Benutzereingabe- und Benutzerausgabeschnittstelle im Strukturmodell durch ein eigenes Containerelement (`ContainerElement`) repräsentiert wird. Dementsprechend müssen R3.2 und R3.3 das benutzerzentrische Domänenmodell hinsichtlich beider Schnittstellentypen und deren Eigenschaften untersuchen und für jede Eigenschaft im Strukturmodell ein Containerelement anlegen, sofern dieses noch nicht vorhanden ist. Um dem Benutzer den späteren Umgang mit der Benutzerschnittstelle zu erleichtern, ist es zweckmäßig, jedes Containerelement, das

[4] Um den Entwicklern die Zusammenarbeit mit dem Kunden zu erleichtern, kann aufgrund der Tatsache, dass alle in dieser Arbeit verwendeten Modelle auf Metamodellen fundiert sind, ohne größeren Aufwand ein je nach Anwendungsszenario geeigneter grafischer Editor zur Bearbeitung der Modelle genutzt werden.

eine Eigenschaft eines Geschäftsobjekts wie beispielsweise `Attribute2` repräsentiert, mit zwei Benutzerschnittstellenelementen zu füllen. Einerseits wird ein Feld (`Field`) benötigt, das dem Benutzer den Wert der Eigenschaft des Containerelements anzeigt und gegebenenfalls auch dessen Bearbeitung ermöglicht. Andererseits muss dem Feld eine Beschreibung (`Description`) beigefügt werden, die den Benutzer darüber informiert, welches Feld er gerade betrachtet oder bearbeitet. Der in Abbildung 65 dargestellte Entwurf zeigt daher das Containerelement `Attribute2` mit den beiden als `Description` und `Field` bezeichneten Benutzerschnittstellenelementen.

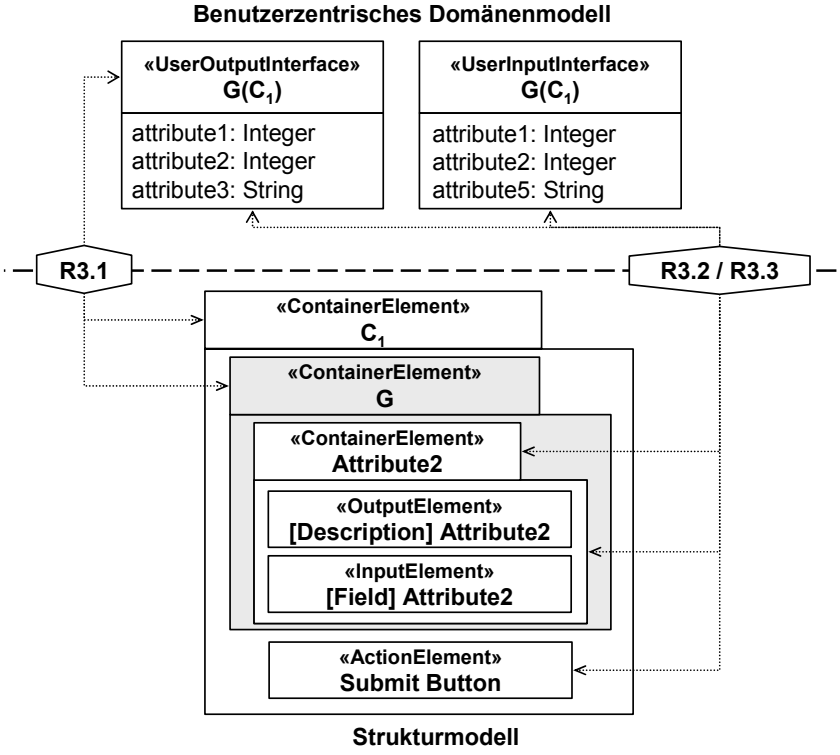

Abbildung 65: Entwurf der Relation R3.1 und R3.2 der Transformation T3

Um eine automatisierte Erzeugung des plattformunabhängigen Strukturmodells zu ermöglichen, soll die Modell-zu-Modell-Transformation T3 und ihre drei Relationen in QVT-*Relations* implementiert werden. Die folgenden Abschnitte widmen sich dieser Implementierung.

5.2.4.1 Relation R3.1 – Model_To_Package

Die in Abbildung 67 dargestellte Relation `Model_To_Package` (R3.1) erstellt für jede Benutzeraktion des benutzerzentrischen Domänenmodells ein eigenständiges Strukturmodell und fügt diese dem gemeinsamen Paket `uiPackage` hinzu. Hierzu wählt das Domänenmuster über dem Quellmodell alle Modellelemente aus, die durch den Stereotyp Benutzerschnittstelle (`UserInterface`) ausgezeichnet sind. Aus den *Tagged Values* dieses Stereotyps können die benötigten Bezeichner der zugeordneten Benutzeraktion (`userActionName`) und des zugeordneten Geschäftsobjekts (`businessObjectName`) entnommen und neuen Variablen zugewiesen werden. Im Strukturmodell wird zugleich ein weiteres Containerelement `uiContainer` vom Typ Formular (`ContainerTypes::Form`) erstellt, das die Wurzel der Benutzerschnittstelle bildet. Abschließend fügt die Relation dem `uiContainer` einen Absendeknopf (`submitButton`) hinzu und erzeugt für jedes Geschäftsobjekt, das in dieser Benutzerschnittstelle bearbeitet wird, ein weiteres Containerelement `boContainer` vom Typ Komposition (`ContainerTypes::Composition`).

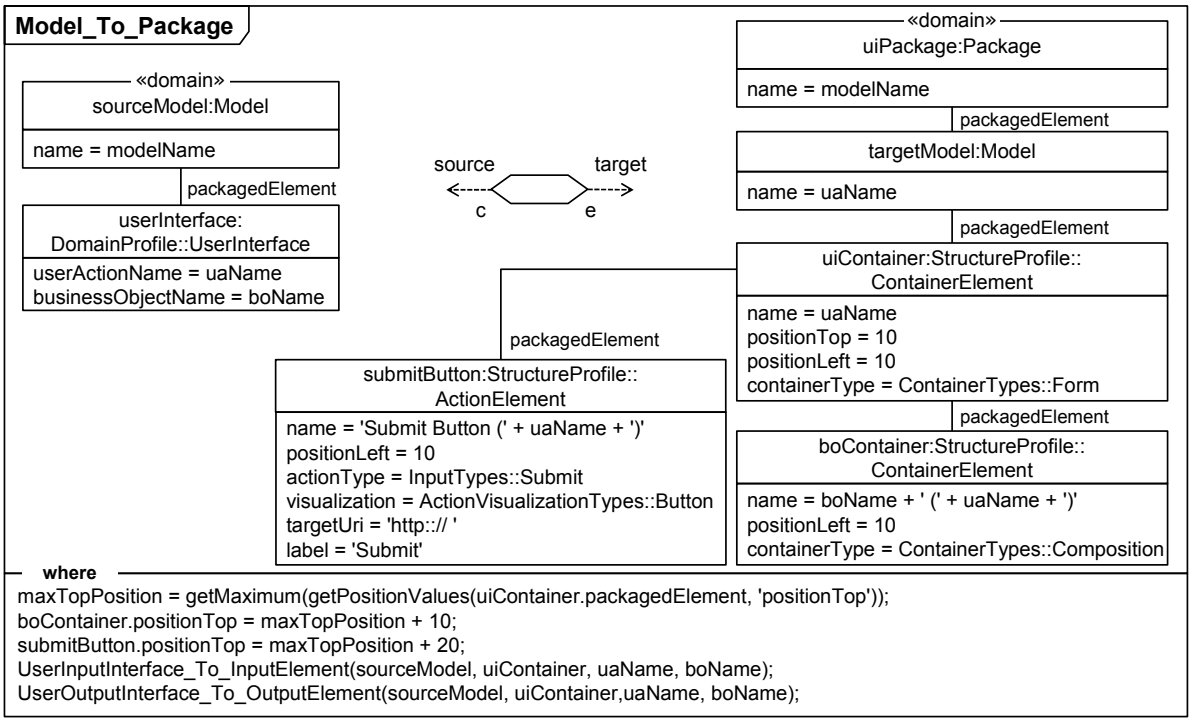

Abbildung 66: Model_To_Package-Relation in QVT-Relations

Da das Strukturmodell bereits einfache Positionsangaben der Modellelemente enthalten muss, setzt R3.1 die Positionsangabe `positionLeft` der erzeugten Modellelemente auf den Standardwert 10. Der gleiche Standardwert wird der vertikalen Position (`positionTop`) des `uiContainer` zugewiesen. Für die ebenfalls erzeugten Modellelemente `boContainer` und `submitButton` müssen die vertikalen Positionsangaben zuerst ermittelt werden, da in einer Benutzeraktion mehrere Geschäftsobjekte in verschiedenen `boContainern` bearbeitet werden können.

Damit es zu keinen Überschneidungen mit bereits im `uiContainer` vorhandenen `boContainern` kommt, wird der höchste vertikale Positionswert durch die in Anhang A.1 aufgeführte Hilfsmethode `getMaximum` ermittelt und dieser bei jeder Neuzuweisung um 10 inkrementiert. Abschließend wird durch die *where*-Klausel der Relation die Gültigkeit der im Anschluss beschriebenen Relationen `UserInputInterface_To_InputElement` und `UserOutputInterface_To_OutputElement` gefordert. Die Reihenfolge, in der die Gültigkeit der beiden Relationen eingefordert wird, ist dabei wesentlich. Wird zu einem Benutzerschnittstellenelement ein Eingabeelement (`InputElement`) erzeugt, so darf für die gleiche Eigenschaft, sofern sie sowohl in einer Benutzereingabe- als auch in einer Benutzerausgabeschnittstelle enthalten ist, kein zusätzliches Ausgabeelement (`OutputElement`) erzeugt werden. Die hierzu notwendige Überprüfung wird durch R3.3 durchgeführt. Dementsprechend muss zuerst die Gültigkeit von R3.2 eingefordert werden.

5.2.4.2 Relation R3.2 – UserInputInterface_To_InputElement

Die Relation `UserInputInterface_To_InputElement` (R3.2) werden als Parameter das benutzerzentrische Domänenmodell (`sourceModel`), ein Containerelement (`boContainer`) des Strukturmodells sowie die Bezeichner der zu betrachtenden Benutzeraktion (`uaNameParam`) und des assoziierten Geschäftsobjekts (`boNameParam`) übergeben. Die beiden letzten Parameter werden benötigt, da in der übergeordneten Relation R3.1 das Strukturmodell für eine bestimmte Benutzeraktion in Bezug auf ein konkretes Geschäftsobjekt angelegt wurde. Daher stellt die *when*-Klausel von

R3.2 zuerst sicher, dass die aktuell betrachtete Benutzereingabeschnittstelle (`userInputInterface`) des benutzerzentrischen Domänenmodells zu der passenden Benutzeraktion und dem passenden Geschäftsobjekt gehört. Ist diese Bedingung erfüllt, dann wird für jede Eigenschaft (`Property`) der Benutzereingabeschnittstelle ein Containerelement `fieldContainer` vom Typ `Fieldset` erzeugt. Dieser Abbildungsregel liegt der Gedanke zugrunde, dass die Darstellung einer Eigenschaft wie beispielsweise „Name“ in der Benutzerschnittstelle immer auf zwei Benutzerschnittstellenelemente abgebildet werden muss. Zum einen wird ein Feld-ähnliches Benutzerschnittstellenelement benötigt, über das der Benutzer die Eigenschaft betrachten oder auch bearbeiten kann. Zum anderen benötigt das Feld eine Beschriftung, damit der Benutzer auch erkennen kann, welches Feld er gerade bearbeitet. Aus diesem Grund fordert die Relation für jede Eigenschaft ein Containerelement ein und fügt diesem ein Eingabeelement als Feld (`field`) und ein Ausgabeelement als Beschreibung (`fieldDescription`) hinzu. Um den genauen Typ (`inputType`) des Eingabeelements bestimmen zu können, bedient sich die Relation der Hilfsmethode `determineInputType`, die in Anhang A.1 spezifiziert ist.

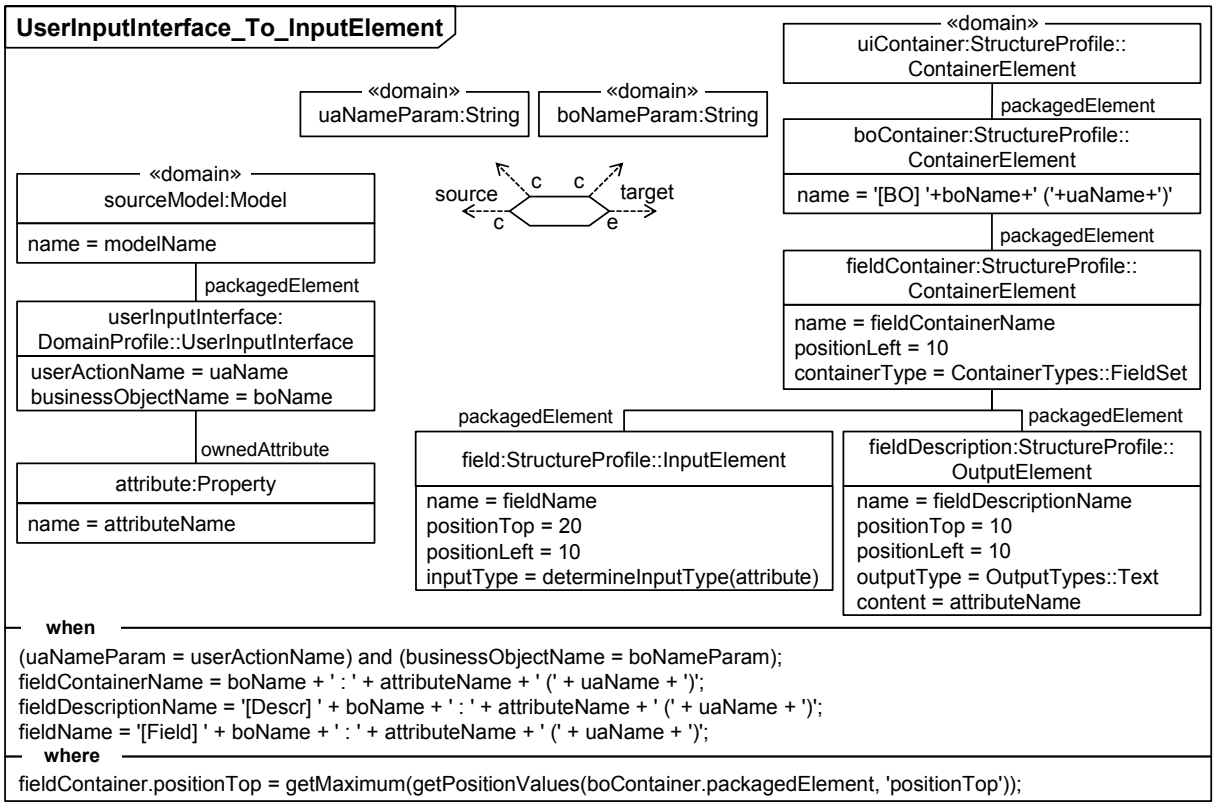

Abbildung 67: UserInputInterface_To_InputElement-Relation in QVT-Relations

Nachdem die Gültigkeit der Relation hergestellt ist, gleichen die in der *where*-Klausel der Relation aufgerufenen Hilfsmethoden die Positionsangaben der neu erzeugten Modellelemente an.

5.2.4.3 Relation R3.3 – UserOutputInterface_To_OutputElement

Die dritte Relation `UserOutputInterface_To_OutputElement` (R3.3) entspricht im Wesentlichen der zuvor eingeführten Relation 3.2, allerdings greift R3.3 Benutzerausgabeschnittstellen (`UserOutputInterface`) anstelle von Benutzereingabeschnittstellen auf und erzeugt aus diesen Ausgabeelemente (`OutputElement`). Hierbei muss durch die Hilfsmethode `elementExiststInContainer` (Anhang A.1) sichergestellt werden, dass die durch R3.3 im

Containerelement `fieldContainer` zu erzeugenden Ausgabeelemente nicht bereits in Form von Eingabeelementen in R3.2 erzeugt wurden. Diese Situation kann genau dann eintreten, wenn eine Eigenschaft sowohl in einer Benutzereingabe- als auch in einer Benutzerausgabeschnittstelle spezifiziert wurde. Ferner füllt R3.3 die Ausgabeelemente mit Standardwerten, um auf diese Weise dem Kunden in einem erzeugten Prototyp der Benutzerschnittstelle das Verhalten zur Laufzeit demonstrieren zu können (vgl. Abschnitt 6.2.6).

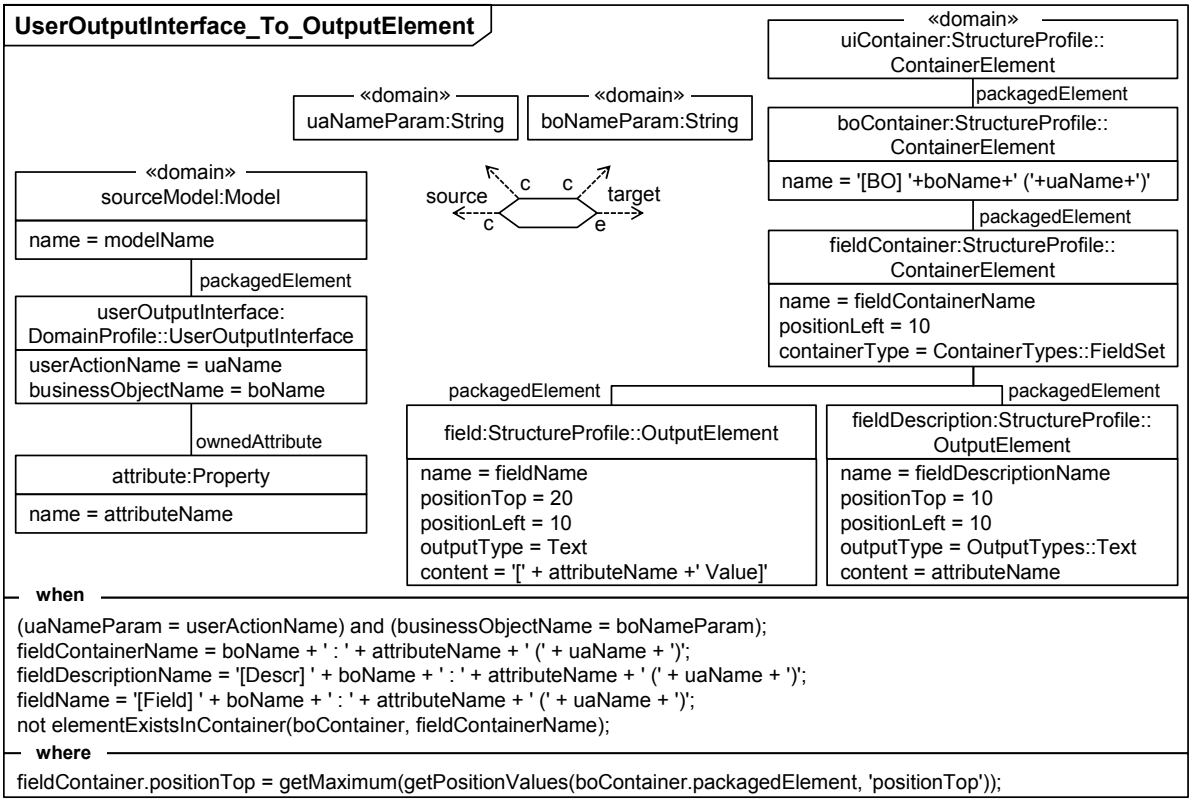

Abbildung 68: UserOutputInterface_To_OutputElement-Relation in QVT-Relations

5.2.4.4 Demonstration der Anwendung

Das benutzerzentrische Domänenmodell als Quellmodell der im vorangegangenen entwickelten Transformation T3 umfasst alle Eigenschaften der Geschäftsobjekte, die ein Benutzer im Kontext der Benutzeraktionen eines Workflows in den verschiedenen Benutzerschnittstellen einsehen oder bearbeiten möchte. Da die Benutzerschnittstelle somit der primäre Berührungspunkt des Benutzers mit dem Anwendungssystem ist, können anhand der Benutzerschnittstelle die Anforderungen des Kunden als späterem Benutzer des Anwendungssystems verifiziert werden. Durch die entwickelte Transformation T3 kann als wesentlicher Mehrwert die Abbildung des benutzerzentrischen Domänenmodells auf ein Strukturmodell automatisiert und damit im Gegensatz zu einer manuellen Abbildung durch den Menschen in deutlich kürzerer Zeit und mit einer geringeren Fehlerwahrscheinlichkeit durchgeführt werden. Abbildung 69 zeigt beispielhaft das Ergebnis der Anwendung der Transformation T3 auf ein benutzerzentrisches Domänenmodell.

Das aus dem benutzerzentrischen Domänenmodell erzeugte Strukturmodell enthält keine plattformspezifischen Details und ist im Sinne der modellgetriebenen Architektur daher ebenfalls als ein plattformunabhängiges Modell anzusehen. Auf diese Weise kann die grundlegende Struktur einer Benutzerschnittstelle allgemeingültig im Strukturmodell erfasst und anschließend durch weitere Transformationen für beliebige Plattformen ausgeprägt werden. Gerade im Bereich der Benutzer-

schnittstellen, in dem durch die vielen möglichen Technologien, Entwicklungsrahmenwerke aber auch Endgeräte eine breite Vielfalt an Zielplattformen zur Verfügung steht, bringt die vorgenommene Abstraktion von den spezifischen Details einer konkreten Plattform daher einen wesentlichen Mehrwert mit sich und hilft, die vorhandene Heterogenität zu überwinden.

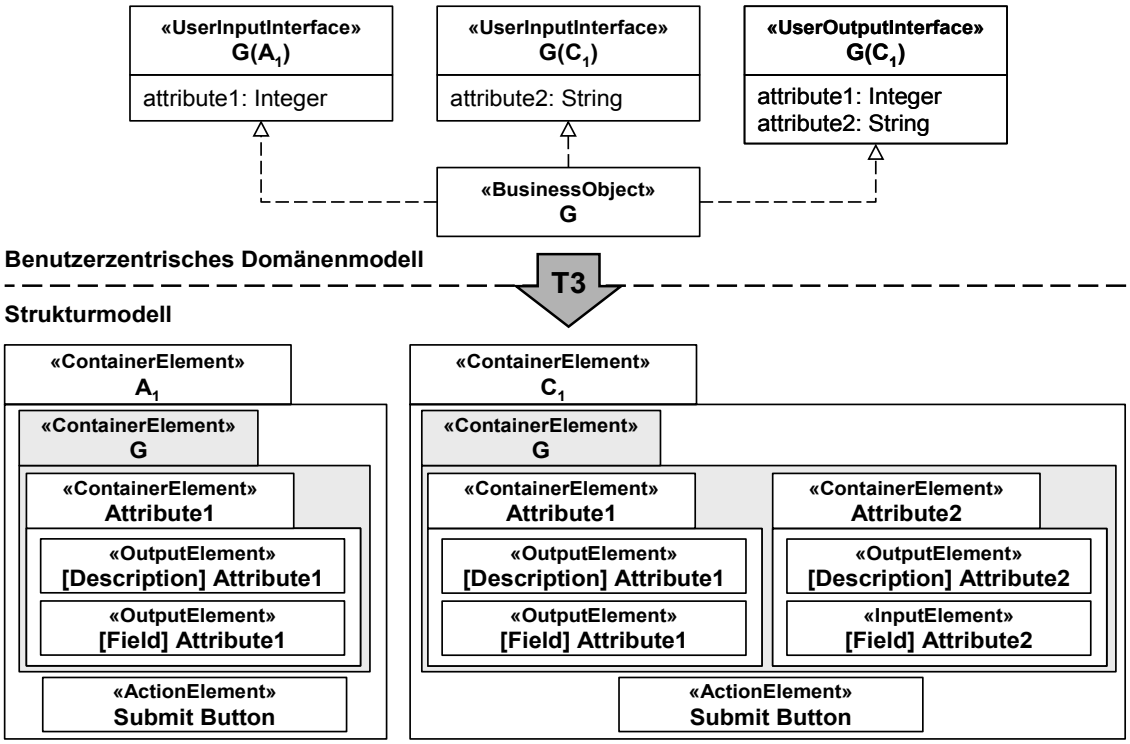

Abbildung 69: Ergebnis der Anwendung der Transformation T3

Die in diesem Abschnitt erzeugten plattformunabhängigen Modelle müssen nun in einem nächsten Entwicklungsschritt auf plattformspezifische Modelle und anschließend auf ausführbare Softwareartefakte abgebildet werden. Um diesen Entwicklungsschritt durchführen zu können, muss jedoch zunächst geklärt werden, welche konkreten Plattformen die Ziele der angestrebten Transformationen sein sollen. Daher widmet sich der folgenden Abschnitt der Konzeption einer dienstorientierten Architektur zur Unterstützung von Benutzerinteraktion und beleuchtet in diesem Zusammenhang bestehende Standards und Spezifikationen als mögliche Zielplattformen der Transformationen.

5.3 Dienstorientierte Architektur zur Unterstützung von Benutzerinteraktion

Dienstorientierte Architekturen (engl. *Service-Oriented Architecture*, SOA) konnten sich in der jüngsten Vergangenheit als Paradigma für den Entwurf einer flexiblen IT-Unterstützung etablieren. Die in Form einer Dienstkomposition durchgeführte Verschaltung mehrerer Dienste ermöglicht eine Abbildung von Geschäftsprozessen auf die dienstorientierte Architektur, die in Bezug auf Anforderungsänderungen flexibel ist und damit den Ansprüchen heutiger Unternehmen gerecht zu werden vermag. Aus diesem Grund ist es das Ziel dieser Arbeit, bei der Abbildung eines Geschäftsprozesses auf eine dienstorientierte Architektur die Anforderungen der Benutzerinteraktion zu berücksichtigen. Hierzu müssen bestehende Referenzmodelle für dienstorientierte Architekturen unter Beachtung der in Abschnitt 3.1.2 festgehaltenen Anforderungen nach einer generellen Unterstützung von Benutzerinteraktion (A2.1), aber auch hinsichtlich der benötigten interoperablen Dienstschnittstellen (A2.2) und der Integrationsfähigkeit (A2.3) erweitert werden.

5.3.1 Erweiterung der dienstorientierten Architektur

Die Unterstützung menschlicher Interaktion bei der Ausführung von Geschäftsprozessen erfordert zusätzliche Fachfunktionalität wie beispielsweise eine Benutzeraufgabenverwaltung oder eine Benutzerschnittstelle, die in bestehende Referenzmodelle für dienstorientierte Architekturen integriert werden muss. Um diese Erweiterung geeignet im Sinne einer durchgängigen Dienstorientierung durchführen zu können, müssen zunächst bestehende Kernkomponenten einer dienstorientierten Architektur betrachtet und deren Kommunikationsbeziehungen näher beleuchtet werden. Der für eine Erweiterung dienliche Kern einer dienstorientierten Architektur ist in Anlehnung an [EL+06] und das Referenzmodell der Forschungsgruppe Cooperation & Management (vgl. Abbildung 16, Seite 45) in Abbildung 70 dargestellt.

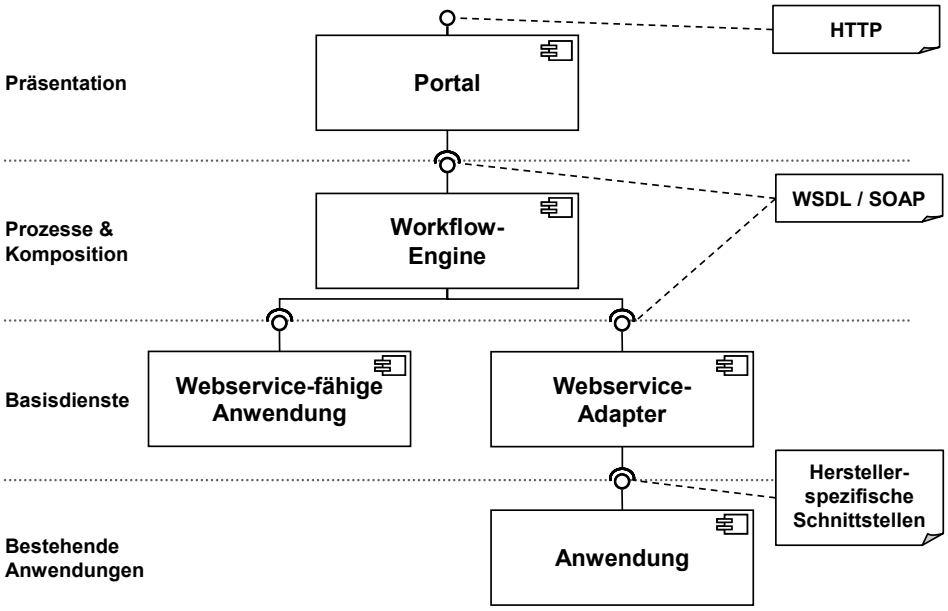

Abbildung 70: Kernkomponenten einer dienstorientierten Architektur

Das in einer dienstorientierten Architektur zum Einsatz kommende Paradigma der Dienstorientierung ermöglicht die Wiederverwendung bestehender Anwendungen, sofern deren Fachfunktionalität über geeignete Dienstschnittstellen zur Verfügung gestellt werden kann. Die bestehenden Anwendungen bilden logisch gesehen die unterste Schicht einer dienstorientierten Architektur. Wie bereits in Abschnitt 1.2 festgehalten, können dienstorientierte Architekturen, auf der logischen Schicht der bestehenden Anwendungen aufbauend, in unterschiedlichen Reifegraden und technischen Ausprägungen etabliert werden. Diese Arbeit geht hier von einer Webservice-basierten dienstorientierten Architektur aus (vgl. Prämisse P4). Die Verwendung der herstellerunabhängigen und standardisierten *Web Service Description Language* (WSDL) [W3C-WSDL1.1] als Beschreibungssprache der Dienstschnittstellen und die Nutzung des Kommunikationsprotokolls SOAP (ehemals *Simple Object Access Protocol*) [W3C-SOAP1.2] gestattet eine bestmögliche Interoperabilität der einzelnen Webservices innerhalb der dienstorientierten Architektur.

Ferner wird angenommen, dass eine bestehende Webservice-Infrastruktur gegeben ist, in der entweder per se Webservice-fähige Anwendungen vorhanden sind oder die Fachfunktionalität von bestehenden Anwendungen, die über keine eigenen Webservice-Schnittstellen verfügen, mittels entsprechender Webservice-Adapter bereitgestellt wird. Diese als Basisdienste bezeichneten Webservices werden von einer Workflow-*Engine* entweder direkt oder über verschiedene Kompositionen von Webservices zur Unterstützung der Workflow-Ausführung genutzt. In der logischen Schicht der Prozesse und Kompo-

sition findet somit die eigentliche Abbildung von Workflows statt, da die Workflow-*Engine* die Ausführung von Workflow-Spezifikationen, die beispielsweise in der XML-basierten Ausführungssprache *Business Process Execution Language* (BPEL) [OASIS-WS-BPEL2] bereitgestellt werden, ermöglicht. Da sich neben BPEL bisher kein weiterer Standard zur maschinenlesbaren Beschreibung von Workflows etablieren konnte, findet sich in der Literatur häufig auch der synonym für Workflow-*Engine* verwendete Begriff der BPEL-*Engine*.

Die Präsentationsschicht einer dienstorientierten Architektur wird heute häufig durch ein Portal realisiert (vgl. Abschnitt 2.4.3). Ein Portal verschattet die Komplexität einer dienstorientierten Architektur vor dem Anwender und gestattet ihm, mit bestehenden Anwendungen wie etwa einem klassischen Webbrowser auf die angebotenen Dienste einer dienstorientierten Architektur zuzugreifen. Im Portal findet dementsprechend die Umsetzung der HTTP-basierten Kommunikationsbeziehung mit dem Benutzer in die Technologien der Webservice-basierten dienstorientierten Architektur statt.

Wie bereits erörtert, können vollautomatisierbare Workflows auf Dienstkompositionen abgebildet werden und durch die Workflow-*Engines* heutiger dienstorientierter Architekturen abgearbeitet werden. Um bei dieser Abbildung auch Workflows mit Benutzerinteraktion berücksichtigen zu können, ist eine Erweiterung der in Form von Diensten bereitgestellten Fachfunktionalität einer dienstorientierten Architektur notwendig. Da das Paradigma der Dienstorientierung einen hohen Grad an Wiederverwendung und eine lose Kopplung der Dienste ermöglicht, muss es auch bei dieser Erweiterung konsequent weiterverfolgt werden. Alle neu eingeführten fachfunktionalen Komponenten müssen daher ebenfalls entsprechende Webservice-Schnittstellen anbieten, um sich nahtlos in die bestehende Webservice-basierte dienstorientierte Architektur einzufügen. Abbildung 71 zeigt eine in [LH+08] publizierte dienstorientierte Architektur, deren Erweiterungen im Folgenden motiviert werden.

Der Zugriff eines Benutzers auf die Fachfunktionalität einer dienstorientierten Architektur erfolgt über ein Portal. Wird ein Benutzer mit der Ausführung einer Benutzeraufgabe betraut, so benötigt er hierzu zunächst eine geeignete Benutzerschnittstelle, die ihm durch das Portal bereitgestellt wird. Ein Portal ermöglicht die Darstellung individueller personifizierter Inhalte für einen Benutzer, unter anderem durch die Integration unterschiedlicher *Portlets* auf einer Portalseite (vgl. Abschnitt 2.4.3.1). Über *Portlets* ist es somit möglich, dem Benutzer innerhalb einer Portalseite unterschiedliche Benutzerschnittstellen für die jeweils zu bearbeitende Benutzeraufgabe zur Verfügung stellen zu können. Um zusätzlich eine flexible Bereitstellung der Benutzerschnittstellen ermöglichen zu können, muss das Paradigma der Dienstorientierung auch auf der Präsentationsschicht konsequent weiterverfolgt werden. Im Kontext der Webservice-basierten dienstorientierten Architektur bedeutet dies, dass ein *Portlet* über eine Webservice-Schnittstelle dienstorientiert zur Verfügung gestellt werden muss. Die angebotene Fachfunktionalität eines solchen präsentationsorientierten Webservice liefert somit die zu einer Benutzeraktion passende Benutzerschnittstelle.

Die auf diesem Wege realisierte lose Kopplung zwischen Portal und *Portlets* ermöglicht eine flexible Nutzung der *Portlets* innerhalb der logischen Präsentationsschicht der dienstorientierten Architektur. Ein weiterer Vorteil dieses Ansatzes liegt in der Wiederverwendbarkeit der Benutzerschnittstellen. Typische Benutzeraktionen wie beispielsweise die Eingabe von Kundendaten können immer wieder in den Workflows eines Unternehmens auftreten. Bietet ein präsentationsorientierter Dienst die dazu benötigte Benutzerschnittstelle über eine Webservice-Schnittstelle an, so kann diese Benutzerschnittstelle in unterschiedlichen Anwendungsszenarien mehrfach wiederverwendet werden.

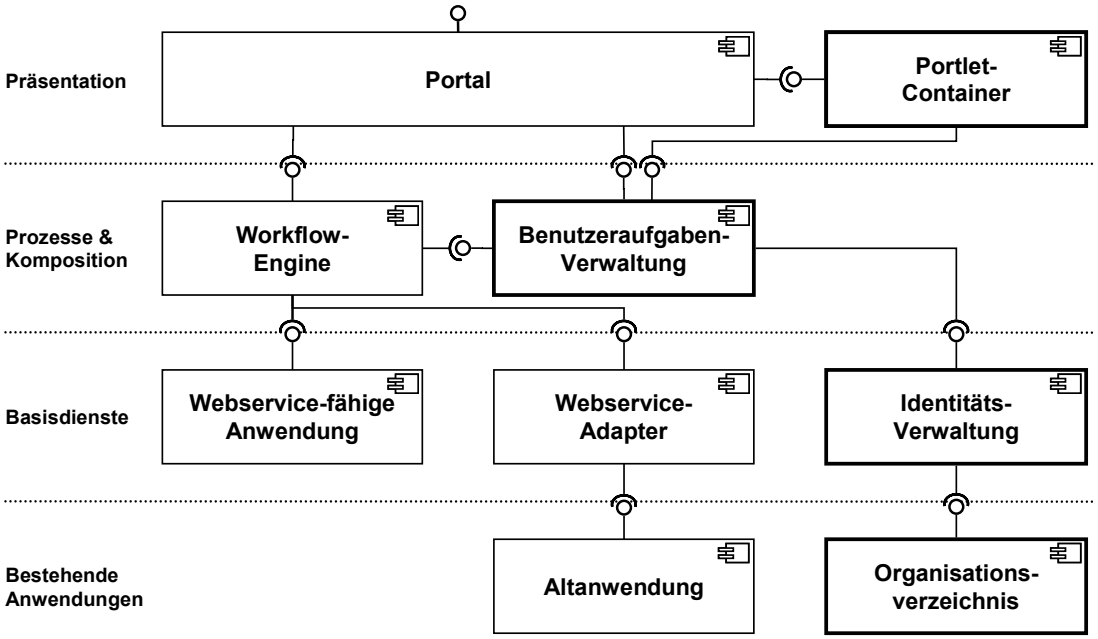

Abbildung 71: Erweiterte dienstorientierte Architektur

Der Ansatz, eine Benutzerschnittstelle als Dienst bereitzustellen, wird insbesondere durch den in Abschnitt 2.4.3 beleuchteten OASIS-Standard *Web Services for Remote Portlets* (WSRP) [OASIS-WSRP2] verfolgt, der daher im weiteren Verlauf als Grundlage für die Bereitstellung präsentationsorientierter Webservices dienen soll. Ziel muss es sein, eine durch das modellgetriebene Entwicklungsvorgehen dieser Arbeit erzeugte Benutzerschnittstelle als Webservice entsprechend der WSRP-Spezifikation zur Verfügung stellen zu können. Verschiedene Hersteller von Portal-Servern ermöglichen eine WSRP-konforme Bereitstellung von *Portlets* über eine Webservice-Schnittstelle, sofern die *Portlets* nach bestehenden Standards wie beispielsweise JSR-168 [JCP-JSR168] entwickelt wurden. Abschnitt 6.2.6.3 zeigt dementsprechend, wie ein nach dem modellgetriebenen Vorgehen dieser Arbeit entwickeltes *Portlet* als WSRP-*Portlet* bereitgestellt werden kann.

Neben einer dienstorientierten Bereitstellung der benötigten Benutzerschnittstellen muss auch die Ausführung und Überwachung von Benutzeraufgaben durch eine dienstorientierte Architektur ermöglicht werden. Während die in Abschnitt 2.2.2 beleuchteten Workflow-Management-Systeme die Fachfunktionalität einer Benutzeraufgabenverwaltung integriert in einer Workflow-*Engine* vorsehen, schlägt der Autor in [LH+08] eine eigenständige fachfunktionale Komponente vor, die über eine Webservice-Schnittstelle die für eine Benutzeraufgabenverwaltung benötigte Fachfunktionalität bereitstellt. Eine eigenständige Benutzeraufgabenverwaltung bringt unterschiedlichste Vorteile mit sich. Zum einen kann auf diese Weise eine gegebenenfalls bereits vorhandene Workflow-*Engine* wiederverwendet werden, die lediglich um zusätzliche Fachfunktionalität erweitert werden muss. Zum anderen ermöglicht eine getrennte Betrachtung von Workflow-*Engine* und Benutzeraufgabenverwaltung die Integration mehrerer fachfunktionaler Komponenten zur Benutzeraufgabenverwaltung, die je nach Anwendungsszenario in verschiedenen Fachbereichen eines Unternehmens vorhanden sein können. Beide fachfunktionalen Komponenten sind damit unabhängig voneinander austauschbar. Im Umkehrschluss jedoch kann die Fachfunktionalität der Workflow-*Engine* und der Benutzeraufgabenverwaltung durch ein einziges IT-System erbracht werden, sofern dieses geeignete Dienstschnittstellen für den Zugriff auf die benötigte Fachfunktionalität bereitstellt.

Um die motivierte eigenständige Betrachtung der Benutzeraufgabenverwaltung umsetzen zu können, muss die Benutzeraufgabenverwaltung gegenüber der Workflow-*Engine* verschiedene Operationen zur

Verwaltung von Benutzeraufgaben, wie etwa das Anlegen neuer oder die Löschung abgearbeiteter Benutzeraufgabeninstanzen, anbieten. Dementsprechend ist es erforderlich, dass die Benutzeraufgabenverwaltung intern über verschiedene Benutzeraufgabenspezifikationen verfügt, die durch einen Aufruf der Workflow-*Engine* instanziiert und ausgeführt werden können. Gleichzeitig benötigt die Benutzeraufgabenverwaltung eine Webservice-Schnittstelle zum Portal, um auf diesem die zu einer Benutzeraufgabe zugehörige Benutzerschnittstelle in Form eines *Portlet* aufrufen zu können. Neben den für verschiedene Benutzeraufgaben benötigten *Portlets* muss dem Benutzer im Portal auch seine Aufgabenliste (*Tasklist*) angezeigt werden. Auch dieser Teil der personalisierten Portalseite des Benutzers wird in Form eines *Portlet* realisiert. Dementsprechend fragt die Aufgabenliste je nach konkreter Implementierung in unterschiedlichen Abständen die Benutzeraufgabenverwaltung nach den aktuellen Aufgaben des Benutzers an und stellt diese dem Benutzer auf seiner Portalseite dar.

Eine weitere für die Unterstützung von Benutzerinteraktion wichtige fachfunktionale Komponente ist die Identitätsverwaltung, die mit der Benutzeraufgabenverwaltung und dem Portal kommuniziert. Da die Beschreibung von Benutzeraufgaben häufig abstrakte Ressourcendefinitionen wie beispielsweise organisatorische Einheiten, nicht aber konkrete Benutzernamen beinhaltet, muss die Benutzeraufgabenverwaltung diese Ressourcendefinitionen durch Nutzung einer Identitätsverwaltung auflösen. Die Identitätsverwaltung nimmt daher Anfragen der Benutzeraufgabenverwaltung entgegen, stellt eine Anfrage an das angebundene Organisationsverzeichnis und liefert die ermittelten konkreten Benutzernamen zurück. Des Weiteren wird die als Webservice konzipierte Identitätsverwaltung auch zur Authentifizierung und Autorisierung konkreter Benutzer, beispielsweise bei der Anmeldung eines Benutzers am Portal oder bei der Zuteilung einer Benutzeraufgabe zu geeigneten menschlichen Ressourcen, verwendet. Eine für diese Arbeit passende Realisierung einer Identitätsverwaltung ist in [Em08] beschrieben und soll daher für diese Arbeit als gegeben angenommen werden.

Neben den motivierten fachfunktionalen Erweiterungen werden ferner geeignete Spezifikationssprachen für Softwareartefakte benötigt, die durch diese Erweiterungen ausgeführt werden können. Für den *Portlet*-Container steht hierzu beispielsweise die Java-*Portlet*-Spezifikation JSR168 [JCP-JSR168] zur Verfügung. Im Bereich der Benutzeraufgabenverwaltung konnte sich bisher jedoch kein Standard etablieren. Allerdings stellen Agrawal et al. in [AA+07a] und [AA+07b] einen nach [RA07] vielversprechenden Ansatz für eine Spezifikationssprache von Benutzeraufgaben in Form zweier Erweiterungen der standardisierten Prozessausführungssprache WS-BPEL [OASIS-WS-BPEL2] vor. Da die Integration der Interaktion eines Menschen in einen Workflow und die daraus resultierenden Benutzeraufgaben einen zentralen Bestandteil dieser Arbeit bilden, werden die vorgeschlagenen Erweiterungen von WS-BPEL im folgenden Abschnitt eingeführt.

5.3.2 Spezifikation von Benutzeraufgaben

Zur Spezifikation vollständig IT-unterstützter Workflows kommt im Webservice-Umfeld die *Web Service Business Process Execution Language* (kurz WS-BPEL) zum Einsatz, die durch die OASIS standardisiert ist [OASIS-WS-BPEL2]. Im Jahre 2002 erstmals unter dem Namen *Business Process Execution Language* (BPEL) als Vereinigung der *Web Service Flow Language* (WSFL) von IBM [IBM-WSFL] und XLANG von Microsoft [MS-XLANG] veröffentlicht, stellt WS-BPEL heute die populärste Sprache zur Spezifikation von Webservice-Verschaltungen dar [Ha05]. Dabei ist WS-BPEL zunächst auf die Untermenge von Workflows beschränkt, die ohne menschliche Interaktion auskommen. Kloppmann et al. adressieren in [KK+05] die fehlende Unterstützung von Benutzeraktivitäten in WS-BPEL durch eine als WS-BPEL *Extension for People* (kurz BPEL4People) bezeichnete Erweiterung. Weitestgehend auf den Grundsätzen von [KK+05] aufbauend publizieren Agrawal et al.

mit BPEL4People [AA+07a] und *Web Services Human Task* (kurz WS-HumanTask) [AA+07b] zwei auf je einer eigenen XML-Syntax basierende Ansätze, welche die Spezifikation von Benutzeraktivitäten in einer Webservice-Komposition gestatten. Beide Spezifikationen liegen aktuell der OASIS zur Standardisierung vor.

Die drei Spezifikationen WS-BPEL, BPEL4People und WS-HumanTask müssen zunächst getrennt voneinander betrachtet werden. Da keine der in WS-BPEL vorhandenen Aktivitäten die Spezifikation einer Benutzeraktivität unterstützt, integriert BPEL4People Benutzeraktivitäten durch die Verwendung des Erweiterungsmechanismus von WS-BPEL [OASIS-WS-BPEL2] und stellt eine neue, als Personenaktivität (engl. *People Activity*) bezeichnete Basisaktivität als Erweiterung der in WS-BPEL gegebenen Erweiterungsaktivität (engl. *Extension Activity*) bereit. Aus einer solchen Personenaktivität heraus resultiert eine Benutzeraufgabe, die im Detail durch WS-HumanTask ausspezifiziert werden muss. Die Spezifikation von Benutzeraufgaben im Kontext eines Workflows kann dabei auf unterschiedliche Art und Weise erfolgen, wie Abbildung 72 verdeutlicht.

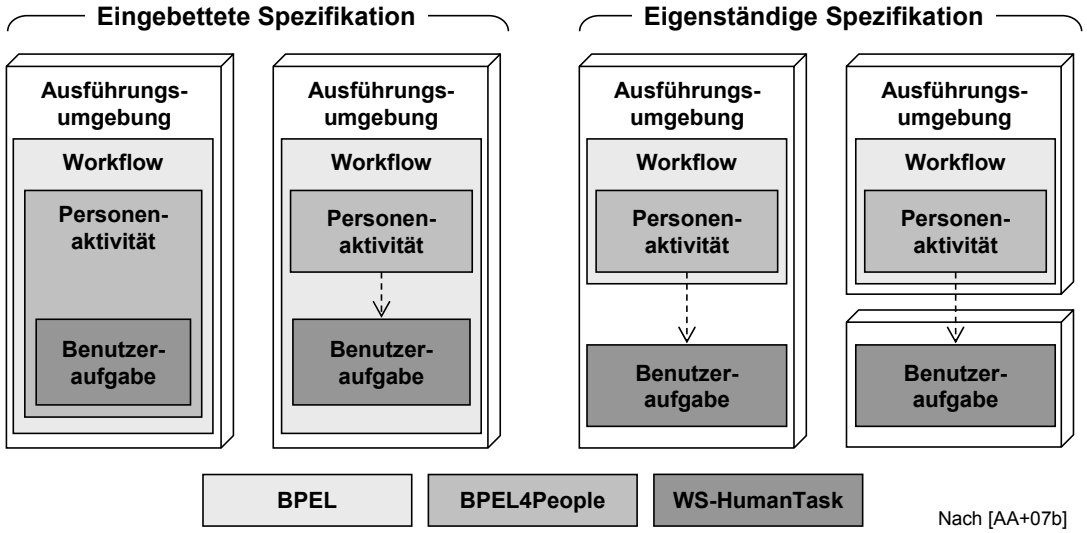

Abbildung 72: Spezifikationsmöglichkeiten einer Benutzeraufgabe

Benutzeraufgaben können entweder eingebettet in der Spezifikation eines Workflows oder eigenständig im Sinne von getrennt von der eigentlichen Workflow-Spezifikation erfasst werden. Eingebettete Benutzeraufgaben werden entweder direkt in einer Personenaktivität eines Workflows oder an zentraler Stelle in der Workflow-Spezifikation (z. B. am Ende der Datei) eingefügt. Der Vorteil der eingebetteten Definition liegt in der Möglichkeit eines direkten Zugriffs einer Benutzeraufgabeninstanz zur Laufzeit auf den aktuellen Kontext des übergeordneten Workflows. Allerdings kann bei dieser Variante eine eingebettete Benutzeraufgabenspezifikation von anderen Workflows nicht wiederverwendet werden. Erfolgt die Spezifikation einer Benutzeraufgabe hingegen eigenständig an einer zentralen Stelle, dann kann diese Spezifikation wiederverwendet werden, sie hat jedoch zur Laufzeit keinen Zugriff auf den Kontext des Workflows. Werden Benutzeraufgabenspezifikationen zusätzlich, wie im rechten Teil von Abbildung 72 dargestellt, innerhalb einer eigenständigen Laufzeitumgebung für Benutzeraufgaben vorgehalten und ausgeführt, dann muss diese Laufzeitumgebung definierte Schnittstellen für den Zugriff bereitstellen. Agrawal et al. schlagen hier eine Webservice-Schnittstelle vor, um die Portabilität der Benutzeraufgaben zu gewährleisten. Genau diesem Vorschlag folgt diese Arbeit, indem die Benutzeraufgabenverwaltung als eigenständiger Webservice konzipiert wird, der mit den bestehenden Webservices einer dienstorientierten Architektur über WSDL-Schnittstellen kommuniziert. Diese Kommunikation zwischen den unterschiedlichen Diensten der erweiterten dienstorientier-

ten Architektur ist für eine Verdeutlichung des Zusammenspiels der einzelnen Dienste wesentlich und soll daher im folgenden Abschnitt genauer beleuchtet werden.

5.3.3 Kommunikationsbeziehungen in der erweiterten dienstorientierten Architektur

Um die Kommunikationsbeziehungen zwischen den einzelnen Diensten der erweiterten dienstorientierten Architektur zu verdeutlichen, sei beispielhaft die Abarbeitung einer Benutzeraufgabe durch einen Benutzer näher beleuchtet. Abbildung 73 stellt die für diesen Vorgang notwendigen Schritte auszugsweise in einer chronologischen Abfolge in Form eines UML-Sequenzdiagramms dar.

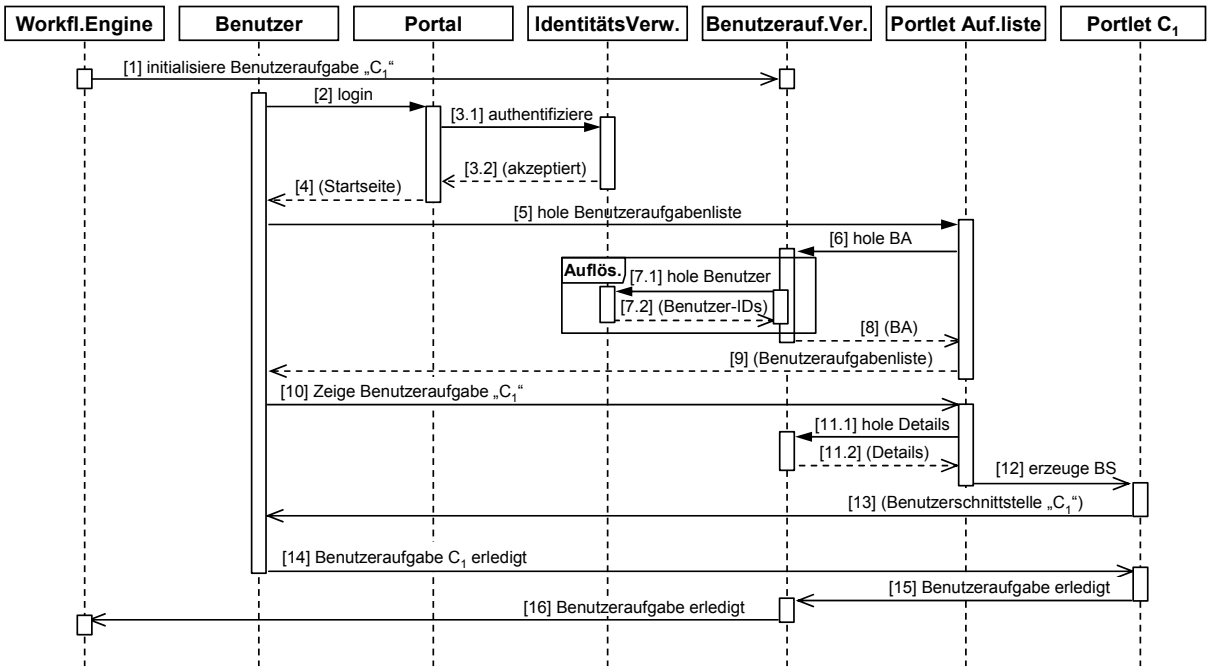

Abbildung 73: Kommunikationsbeziehungen in der erweiterten dienstorientierten Architektur

Erreicht die Workflow-*Engine* während der Abarbeitung einer Workflow-Instanz eine Benutzeraktion C_1, so ruft sie auf der Benutzeraufgabenverwaltung eine Operation zur Initialisierung der Benutzeraufgabe C_1 auf [1]. Meldet sich nun ein Benutzer am Portal an [2], dann wird dieser zunächst durch die Identitätsverwaltung authentifiziert [3.1]. Ist die Authentifizierung erfolgreich [3.2], so zeigt das Portal dem Benutzer seine personalisierte Startseite an [4]. Im nächsten Schritt möchte der Benutzer die ihm aktuell zugeordneten Aufgaben betrachten und fordert daher über das im rechten Teil von Abbildung 73 dargestellte *Portlet* Aufgabenliste seine Benutzeraufgabenliste an [5]. Das *Portlet* ruft dementsprechend auf der Benutzeraufgabenverwaltung alle Benutzeraufgaben ab, die dem Benutzer zugeordnet sind [6]. Da eine Benutzeraufgabenspezifikation abstrakte menschliche Ressourcen wie Rollen und organisatorische Einheiten enthalten kann (vgl. dazu Abschnitt 4.3), müssen diese durch die Identitätsverwaltung aufgelöst [7.1] und durch konkrete Benutzer ersetzt werden [7.2]. Nach Erledigung dieser Aufgabe kann die Benutzeraufgabenverwaltung die aktuellen Benutzeraufgaben des Benutzers an die Aufgabenliste zurückgeben [8] und diese die Benutzeraufgaben für den Benutzer geeignet visualisieren [9], beispielsweise in Form einer tabellarischen Liste (vgl. Abbildung 13, Seite 31). Aus dieser Liste kann der Benutzer nun eine Benutzeraufgabe wie etwa C_1 zur Bearbeitung auswählen [10]. Um dem Benutzer die zu C_1 passende Benutzerschnittstelle anzeigen zu können, muss die Aufgabenliste zunächst die Details zu der Benutzeraufgabe, wie beispielsweise die aus der Benutzeraktion übergebenen Eingabeparameter, anfragen [11.1], bevor mit diesen Details [11.2] *Portlet* C_1 als

Benutzerschnittstelle der Benutzeraufgabe C_1 [12] für den Benutzer dargestellt werden kann [13]. Mit dieser Benutzerschnittstelle kann der Benutzer seine Benutzeraufgabe abarbeiten und schließlich das Ende der Bearbeitung anzeigen [14]. Die durch den Benutzer eingegebenen Daten werden über die Benutzeraufgabenverwaltung [15] zurück an die Workflow-*Engine* übergeben [16], die dann mit der Ausführung der Workflow-Instanz fortfahren kann.

Je nach konkreter Implementierung der einzelnen Dienste können die tatsächlichen Kommunikationsbeziehungen gegenüber den in Abbildung 73 dargestellten Beziehungen abweichen. Beispielsweise wird im beschriebenen Ablauf implizit angenommen, dass eine Kommunikationsbeziehung zwischen ***Portlets*** etwa zwischen der Aufgabenliste und C_1 bei [12] möglich ist. Für die WSRP-Spezifikation ist genau diese Kommunikation jedoch erst ab Version 2.0 vorgesehen. Für eine frühere Version von WSRP müssen die Kommunikationsbeziehungen entsprechend angepasst werden.

Nachdem durch die vorgenommenen Erweiterungen Workflows mit Benutzerinteraktion durch die dienstorientierte Architektur ausgeführt werden können, müssen in einem nächsten Schritt des modellgetriebenen Entwicklungsvorgehens die in diesem Abschnitt betrachteten Spezifikationen als Zielplattformen bei der Erzeugung der plattformspezifischen Modelle berücksichtigt werden. Dabei ist zwischen der WS-HumanTask-Spezifikation und dem WSRP-Standard jedoch zunächst ein wesentlicher Unterschied festzumachen. Während WS-HumanTask eine vollständige Spezifikationssprache für Benutzeraufgaben bereitstellt und zusätzlich die zum Zugriff auf die spezifizierten Benutzeraufgaben benötigten Webservice-Schnittstellen adressiert, sieht WSRP bezüglich der zur internen Realisierung eines präsentationsorientierten Webservice benötigten Spezifikationssprachen keine Einschränkungen vor. Die interne Implementierung eines WSRP-*Portlet* ist somit nicht standardisiert und kann daher beispielsweise gegen die Java-*Portlet*-Spezifikation 168 [JCP-JSR168], aber auch gegen die durch Microsoft vorangetriebene *WebPart*-Spezifikation von ASP.NET (*Active Server Pages* .NET) entwickelt werden. Aufgrund des nicht vorhandenen Standards für die Implementierung von Präsentationskomponenten können daher die zur Erzeugung eines plattformspezifischen Strukturmodells der Benutzerschnittstelle notwendigen Transformationen nicht als allgemeingültig angesehen werden und müssen je nach Anwendungsszenario und den dort angestrebten Zielplattformen angepasst werden. Deswegen wird eine automatisierte Erzeugung des plattformspezifischen Strukturmodells in diesem Kapitel nicht weiter verfolgt, sondern erst in Abschnitt 6.2.6.2 durch den Tragfähigkeitsnachweis wieder aufgegriffen.

Da mit WS-HumanTask ein vielversprechender Standardisierungsvorschlag für Benutzeraufgaben vorliegt und daher die zur Abbildung auf diese Zielplattform benötigten Transformationen als allgemeingültig angesehen werden können, zeigt der folgende Abschnitt, wie das in Abschnitt 5.2 erzeugte plattformunabhängige Benutzeraufgabenmodell auf ein plattformspezifisches Benutzeraufgabenmodell für die Zielplattform WS-HumanTask abgebildet werden kann.

5.4 Automatisierte Erzeugung der Benutzeraufgabenspezifikation

Die in Abschnitt 5.2 entwickelten Transformationen T1 - T3 überführen ein plattformunabhängiges Workflow-Modell in weitere plattformunabhängige Modelle der Benutzerinteraktion. Diese horizontalen Modell-zu-Modell-Transformationen stellen daher keinen Bezug zu plattformspezifischen Details her, sondern fokussieren jeweils einen bestimmten Aspekt der Benutzerinteraktion, wie beispielsweise Benutzeraufgaben, durch ein eigenständiges Modell. In einem nächsten Entwicklungsschritt müssen diese plattformunabhängigen Modelle durch vertikale Modell-zu-Modell-Transformationen für konkrete Plattformen ausgeprägt und in einem finalen Entwicklungsschritt durch eine Modell-zu-Text-

Transformation auf ausführbare Softwareartefakte abgebildet werden. Wie im vorangegangenen Abschnitt erläutert, kommen im Kontext einer dienstorientierten Architektur aufgrund der Kapselung von Fachfunktionalität durch Dienste unterschiedliche Plattformen für die interne Realisierung der diensterbringenden Komponenten infrage (vgl. Abschnitt 4.5). Zur Spezifikation von Benutzeraufgaben soll die zuvor eingeführte Spezifikationssprache WS-HumanTask verwendet werden. Da diese Arbeit die Verwendung Metamodell-basierter Transformationen anstrebt (Anforderung A1.6), ist eine hinreichende Kenntnis über das zugrunde liegende Metamodell Voraussetzung für die Nutzbarkeit von WS-HumanTask als Zielplattform.

5.4.1 Ableitung des WS-HumanTask-Metamodells

Um das Metamodell von WS-HumanTask in den modellgetriebenen Entwicklungsprozess integrieren und dort als plattformspezifisches Metamodell verwenden zu können, bestünde analog zur Vorgehensweise bei den plattformunabhängigen Metamodellen die Möglichkeit, die Konzepte von WS-HumanTask auf das UML-Metamodell und ein UML-Profil umzusetzen. Diese Herangehensweise hat jedoch gerade für plattformspezifische Metamodelle einen wesentlichen Nachteil. Die Metaklassen der UML haben per se unterschiedlichste Eigenschaften, die den Stereotypen eines UML-Profils vererbt werden. Beispielsweise kann eine Instanz der UML-Metaklasse `Class` über Assoziationen zu Instanzen anderer UML-Metaklassen verfügen. Gleiches gilt für zwei Instanzen des Stereotyps `MyClass` als Erweiterung der UML-Metaklasse `Class`. Die somit implizit vererbte Komplexität der UML muss gegebenenfalls nachträglich durch Einschränkungen wieder reduziert werden, um einen Entwickler bei der Überprüfung seiner Modelle auf syntaktische Korrektheit gegen die Spezifikationen einer Plattform in einem Entwicklungswerkzeug sinnvoll unterstützen zu können. Um diesen Aufwand zu vermeiden, kann anstatt einer Nutzung des Metamodells der UML der zur modellgetriebenen Architektur konforme Ansatz[5] der Spezifikation eines eigenen Metamodells verfolgt werden, das wie das UML-Metamodell ebenfalls auf dem Meta-Metamodell MOF beruht. Dieser Ansatz wird durch das im Kontext der *Eclipse Foundation* entstandene *Eclipse Modeling Framework* (EMF) [EF-EMF, SB+09] unterstützt. EMF stellt eine Implementierung des Meta-Metamodells *ECore* zur Verfügung, das die als wesentlich erachtete Teilmenge *essential* MOF (eMOF) von MOF umfasst. Die abstrakte Syntax der *ECore*-basierten Modelle wird in Form von UML-Klassendiagrammen, die statische Semantik mittels der OCL spezifiziert. Durch die Fokussierung von *ECore* auf eMOF umfassen *ECore*-basierte Metamodelle nicht die vollständige Komplexität der MOF-basierten Metamodelle wie beispielsweise das UML-Metamodell. Ein weiterer Vorteil des EMF liegt in der Tatsache begründet, dass durch die vorhandene umfassende Werkzeugunterstützung *ECore*-Metamodelle aus unterschiedlichsten Quellen wie beispielsweise einem XML-Schema [W3C-XSD], einem UML-Klassendiagramm, aber auch Java-Schnittstellen gewonnen und somit umgehend zur Modellierung verwendet werden können. Die oftmals aufwendige Entwicklung eines Metamodells für eine Plattform kann daher auf diese Weise deutlich vereinfacht werden.

Zur Erzeugung des plattformspezifischen Benutzeraufgabenmodells wird der skizzierte *ECore*-basierte Ansatz gewählt, da das Metamodell der angestrebten Zielplattform WS-HumanTask als XML-Schema zur Verfügung steht [AA+07b]. Daher kann die in [EF-XSD2ECORE] näher beschriebene Importfunktion des EMF zur Erzeugung des WS-HumanTask-Metamodells genutzt und der Aufwand, der bei einer Umsetzung auf das UML-Metamodell durch die benötigten umfangreichen OCL-

[5] Da die modellgetriebene Architektur lediglich ein MOF-basiertes Metamodell fordert [MM03], können neben der UML beliebige eigene Metamodelle verwendet werden, solange diese auf der MOF aufbauen.

Einschränkungen entstünde, vermieden werden. Mit dem WS-HumanTask-*ECore*-Modell steht somit ein Zielmodell für die benötigte Modell-zu-Modell-Transformation T4 bereit. Um die verwendeten Metamodelle, Modelle und Transformationen besser einordnen zu können, gibt Abbildung 74 einen detaillierten Überblick über das verfolgte modellgetriebene Entwicklungsvorgehen als Verfeinerung des allgemeinen, in Abbildung 55 (Seite 130) dargestellten Gesamtüberblicks.

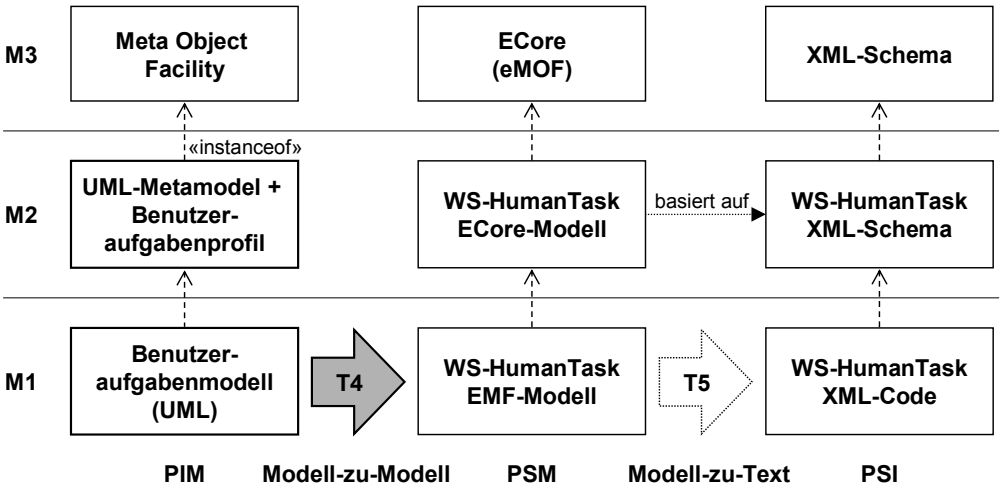

Abbildung 74: Verwendetes Vorgehen zur Erzeugung der Benutzeraufgabenspezifikation

Um die Erzeugung eines WS-HumanTask-EMF-Modells aus einem plattformunabhängigen Benutzeraufgabenmodell durch die Modell-zu-Modell-Transformation T4 automatisieren zu können, müssen die Abbildungsbezüge zwischen beiden Modellen hergestellt und anschließend formalisiert werden. Der folgende Abschnitt widmet sich dieser Aufgabe.

5.4.2 Erzeugung des plattformspezifischen Benutzeraufgabenmodells

Eine erste Relation R4.1 der Transformation T4 muss anhand des übergebenen plattformspezifischen Benutzeraufgabenmodells zunächst überprüfen, ob auf dem Zielmodell ein WS-HumanTask-EMF-Modell (resp. Paket) mit gleichem Bezeichner existiert. Ist ein solches Modell noch nicht vorhanden, so muss es durch R4.1 angelegt werden. Da eine Benutzeraufgabe das zentrale Element des Benutzeraufgabenmodells darstellt, kann R4.1 ferner alle Benutzeraufgaben in das entsprechende Modellelement `Task` des WS-HumanTask-EMF-Modells überführen. Durch weitere Relationen der Transformation müssen im Anschluss die einzelnen Modellelemente des Benutzeraufgabenmodells auf die Modellelemente des WS-HumanTask-EMF-Modells abgebildet werden. Da die Metamodelle des Benutzeraufgabenmodells und des WS-HumanTask-*ECore*-Modells teilweise unterschiedliche Bezeichnungen für Modellelemente verwenden, ist eine Abbildung entsprechend der in Abbildung 75 dargestellten Relationen notwendig. Die im Benutzeraufgabenprofil spezifizierte Aufgabenrolle des Assistenten wird durch WS-HumanTask nicht unterstützt und daher nicht weiter betrachtet.

Die entworfenen Relationen folgen einem einfachen Schema: In der ersten Relation R4.1 werden alle Benutzeraufgaben (`UserTask`) in Aufgaben (`Tasks`) des WS-HumanTask-EMF-Modells überführt. Zusätzlich fordert R4.1 die Gültigkeit derjenigen Relationen ein, die Modellelemente adressieren, die mit einer Benutzeraufgabe assoziiert sein können (etwa eine Eskalation). Die auf diese Weise verknüpften Relationen fordern selbst nach dem gleichen Schema die Gültigkeit weiterer Relationen ein. Auf diese Weise werden alle Modellelemente eines Benutzeraufgabenmodells von einer Benutzeraufgabe an schrittweise transformiert. Zur besseren Verdeutlichung dieses Prinzips werden als veranschaulichende Beispiele die Relationen R4.1 und R4.6 in den folgenden beiden Abschnitten einge-

führt. Alle weiteren Relationen sind nach dem gleichen Schema aufgebaut und können [Ry09] entnommen werden. Um die Lesbarkeit der im Folgenden mittels QVT-*Relations* formalisierten Relationen zu verbessern, wird das WS-HumanTask-*ECore*-Modell als WSHT-ECM abgekürzt.

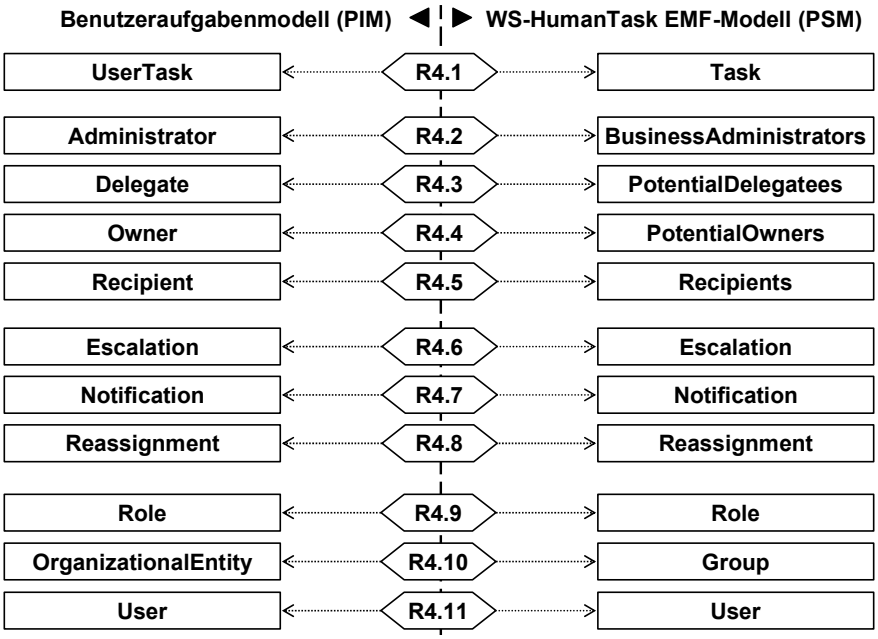

Abbildung 75: Entwurf der Relationen der Transformation T4

5.4.2.1 Relation R4.1 – UserTask_To_Task

Die Relation `UserTask_To_Task` überprüft, ob auf der Zieldomäne bereits ein Zielmodell (`targetModel`) erzeugt wurde. Wenn nein, wird das Modell erzeugt und alle Benutzeraufgaben (`userTask`) des Quellmodells in Aufgaben (`task`) des Zielmodells mit gleichem Bezeichner überführt. Dabei ist zu beachten, dass *ECore* das Konzept eines Modells an sich nicht kennt, für eine zusammengehörende Menge von Modellelementen jedoch ein `EPackage` vorsieht, das hier zum Einsatz kommt (vgl. dazu [Ja08, SB+09]).

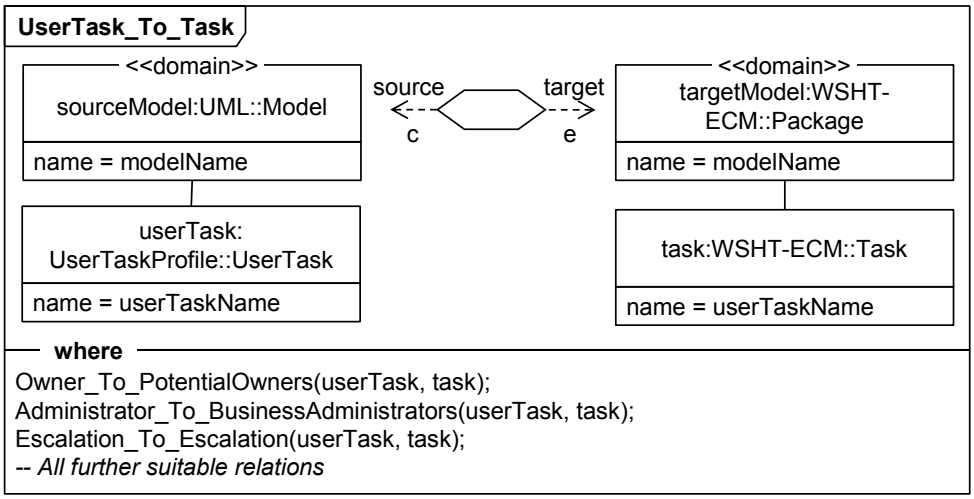

Abbildung 76: UserTask_To_Task-Relation in QVT-Relations

Für jede erzeugte Aufgabe fordert R4.1 in der *where*-Klausel die Gültigkeit derjenigen Relationen ein, die Modellelemente transformieren, die mit einer Benutzeraufgabe assoziiert sein können. Dies

können Aufgabenrollen wie der Eigentümer (`Owner`) oder der Administrator (`Administrator`), aber auch Eskalationen sein. Alle in der *where*-Klausel aufgeführten Relationen fordern selbst nach dem gleichen Prinzip die Gültigkeit weiter Relationen ein.

5.4.2.2 Relation R4.6 – Escalation_To_Escalation

Relation `Escalation_To_Escalation` bekommt von Relation 4.1 eine Benutzeraufgabe (`userTask`) sowie eine Aufgabe (`task`) als Parameter übergeben und wählt alle über eine `include`- Beziehung mit der Benutzeraufgabe assoziierten Eskalationen (`escalation`) aus.

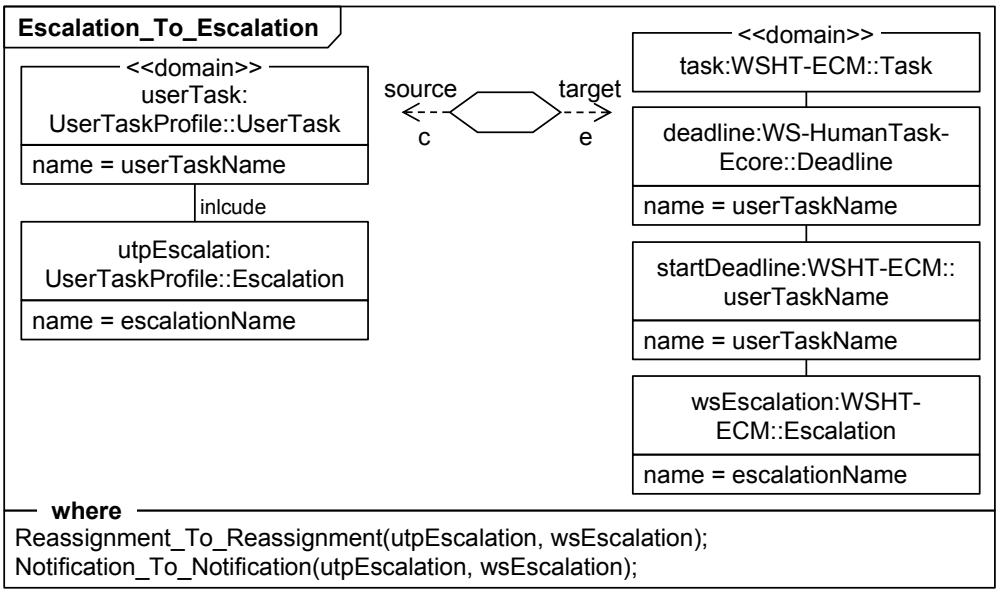

Abbildung 77: Escalation_To_Escalation-Relation in QVT-Relations

Eskalationen sind in der WS-HumanTask-Spezifikation als ein Unterelement einer `Deadline` und `startDeadline` erfasst. Daher erzeugt die Relation beide Modellelemente auf dem Zielmodell und ordnet diesen die Eskalation unter. Da einer Benutzeraufgabe mehrere Eskalationen zugeordnet werden können, werden die Modellelemente `Deadline` und `startDeadline` jeweils mit dem Bezeichner der Benutzeraufgabe (`userTaskName`) versehen, die Eskalation übernimmt den Bezeichner der Eskalation (`escalationName`) des Quellmodells. Nach dem zuvor bereits erläuterten Prinzip fordert R4.6 in der *where*-Klausel die Gültigkeit weiterer Relationen ein, die Modellelemente transformieren, die direkt im Bezug zu einer Eskalation stehen können. In diesem Fall sind dies die Eskalationsaufgaben Neuzuweisung (`Reassignment`) und Benachrichtigung (`Notification`), die beide die Eskalation des Quellmodells (`utpEscalation`) und des Zielmodells (`wsEscalation`) als Parameter übergeben bekommen.

5.4.2.3 Demonstration der Anwendung

Die Anwendung der Transformation T4 überführt ein plattformunabhängiges Benutzeraufgabenmodell, das durch die Modellelemente des UML-Metamodells und des Benutzeraufgabenprofils spezifiziert ist, automatisiert in ein WS-HumanTask-EMF-Modell, das auf den Modellelementen des WS-HumanTask-*ECore*-Modells basiert. Eine manuelle Abbildung ist somit nicht mehr notwendig. Abbildung 78 zeigt an der Benutzeraktion C_1 des Beispiel-Workflows das Ergebnis der Anwendung der Transformation T4.

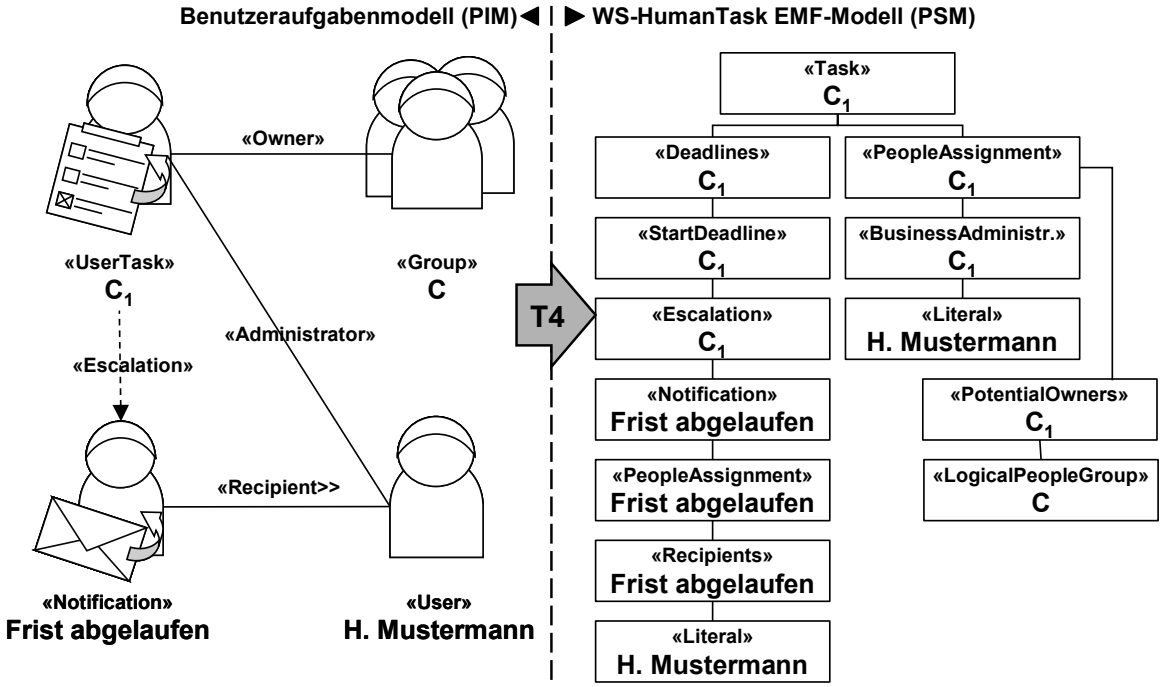

Abbildung 78: Ergebnis der Anwendung von Transformation T4

In einem nächsten Entwicklungsschritt kann das automatisiert erzeugte WS-HumanTask-EMF-Modell durch entsprechende Fachentwickler weiter verfeinert werden. Hierzu kann beispielsweise der grafische Editor Verwendung finden, der durch die vorhandenen Werkzeuge des EMF bei der Erzeugung eines Metamodells aus einer der unterstützten Quellen (etwa ein XML-Schema) gleichzeitig mit generiert werden kann. Da der Funktionsumfang dieses grafischen Editors jedoch beschränkt und für verschiedene Anwendungsszenarien nicht ausreichend ist, sieht das EMF eine zusätzliche, als *Graphical Modeling Framework* (GMF) [EF-GMF] bezeichnete Erweiterung vor, durch die ein geeigneter grafischer Editor für ein eigenes Metamodell entwickelt werden kann. Abschnitt 6.3 zeigt im Rahmen des Tragfähigkeitsnachweises dieser Arbeit ein konkretes Beispiel für die Nutzung des GMF auf. Nach der Erzeugung des WS-HumanTask-EMF-Modells muss dieses in einem letzten Entwicklungsschritt auf ausführbare Softwareartefakte im Sinne von WS-HumanTask-XML-Quellcode überführt werden.

5.4.3 Erzeugung des WS-HumanTask-XML-Quellcode

Die Nutzung des EMF zur Erzeugung eines eigenen Metamodells auf Basis eines XML-Schemas bringt einen weiteren wesentlichen Vorteil mit sich: Bei dieser Erzeugung werden durch das EMF die Zuordnungen zwischen den Modellelementen des erzeugten *ECore*-Modells und den Elementen des XML-Schemas als Quellmodell in den Metadaten des *ECore*-Modells hinterlegt. Wird ein EMF-Modell gespeichert, so kann dieses anhand der Metadaten in validen XML-Code serialisiert werden. In der umgekehrten Richtung kann ein in Form von XML gespeichertes EMF-Modell beim Laden basierend auf dem *ECore*-Modell transformiert bzw. deserialisiert werden. Durch diese Vorgehensweise von EMF kann das WS-HumanTask-EMF-Modell bereits als ausführbarer Quellcode betrachtet werden. Die in Abbildung 74 als T5 angedeutete Modell-zu-Text-Transformation zwischen WS-HumanTask-EMF-Modell und plattformspezifischer Implementierung kann somit direkt durch EMF ausgeführt werden. Eine eigenständige Entwicklung der Transformation ist daher nicht erforderlich. Im Rahmen des Tragfähigkeitsnachweises zeigt Kapitel 6 die Serialisierung eines WS-HumanTask-EMF-Modells in WS-HumanTask-XML-Quellcode.

5.5 Resümee

In diesem Kapitel wurden auf Basis der Metamodell-basierten Transformationssprache QVT zunächst drei Modell-zu-Modell-Transformationen T1 - T3 entwickelt, die eine automatisierte Erzeugung der in Kapitel 4 vorgestellten plattformunabhängigen Modelle der Benutzerinteraktion aus einem Workflow-Modell in einem modellgetriebenen Entwicklungsvorgehen entsprechend der in Abschnitt 3.1.1 aufgestellten Anforderung A1.6 ermöglichen. Um mögliche Zielplattformen für die Erzeugung der plattformspezifischen Modelle der Benutzerinteraktion aufzeigen zu können, wurde nachfolgend eine dienstorientierte Architektur konzipiert, deren über interoperable Webserviceschnittstellen (A2.2) bereitgestellte Fachfunktionalität die Ausführung von Geschäftsprozessen mit Benutzerinteraktion (A2.1) ermöglicht und gleichzeitig eine flexible Integration bestehender IT-Systeme gestattet (A2.3). Als zentrale Erweiterungen dieser dienstorientierten Architektur standen eine eigenständige Benutzeraufgabenverwaltung und eine nach den Empfehlungen des WSRP-Standards konzipierte Benutzerschnittstelle im Mittelpunkt. Am Beispiel der für die Benutzeraufgabenverwaltung geeigneten Spezifikationssprache WS-HumanTask wurde ferner die automatisierte Erzeugung eines plattformspezifischen Benutzeraufgabenmodells durch eine weitere Modell-zu-Modell-Transformation demonstriert.

Die automatisierte Erzeugung der Modelle der Benutzerinteraktion hat im Vergleich zu einer manuellen Abbildung der Modelle zwei wesentliche Vorteile. Eine durch den Menschen schrittweise durchgeführte manuelle Abbildung birgt eine nicht unerhebliche Fehlerwahrscheinlichkeit in sich, da die Interpretation der Modelle durch den Menschen vorgenommen wird. Durch die entwickelten Transformationen kann diese Fehlerwahrscheinlichkeit gemindert werden. Ferner erhöhen die entwickelten Transformationen die Nachvollziehbarkeit der einzelnen Abbildungen, da die verwendeten Transformationsregeln formalisiert und somit für jeden Entwickler einsehbar sind. Durch die Ermöglichung einer automatisierten Erzeugung der Modelle wird somit gleichzeitig ein Teil der Komplexität des Entwicklungsvorgehens in die Transformationen verlagert und damit durch die Wiederverwendbarkeit der Transformationen insgesamt reduziert. Ein weiterer Vorteil der präsentierten Transformationen liegt in deren Umsetzung durch QVT als einer durch die OMG standardisierten Transformationssprache. Hierdurch können die entwickelten Transformationen auf jeder Transformationsausführungsmaschine, die den Standard QVT unterstützt, ausgeführt werden.

Durch die vorgenommenen Erweiterungen einer dienstorientierten Architektur kann die Abbildung von Geschäftsprozessen mit Benutzerinteraktion auf die dienstorientierte Architektur ermöglicht und damit eine flexible Unterstützung dieser Geschäftsprozesse erreicht werden. Geschäftsprozesse, die bisher implizit durch den Menschen selbst gesteuert werden mussten, können nun durch die Workflow-*Engine* in Zusammenarbeit mit der Benutzeraufgabenverwaltung und der als Portal realisierten Benutzerschnittstelle abgearbeitet werden. Eine flexible und überwachbare Integration menschlicher Ressourcen in die Geschäftsprozesse eines Unternehmens wird somit möglich.

6 Demonstration der Tragfähigkeit

Um die Tragfähigkeit der in dieser Arbeit entwickelten Konzepte demonstrieren zu können, wird deren Umsetzung am konkreten Szenario der Forschung und Lehre des Karlsruher Instituts für Technologie (KIT) gezeigt. Anhand eines hieraus entnommenen Geschäftsprozesses kommt das zuvor vorgestellte modellgetriebene Entwicklungsvorgehen zum Einsatz. Unter Anwendung der in Kapitel 4 konzipierten Metamodelle und der in Kapitel 5 entwickelten Transformationen wird ein Modell dieses Geschäftsprozesses erstellt, schrittweise um die Aspekte der Benutzerinteraktion verfeinert und schließlich bis auf ausführbare Softwareartefakte für die in Abschnitt 5.3 entworfene dienstorientierte Architektur transformiert. Hierzu werden im Tragfähigkeitsnachweis unterschiedliche Laufzeitumgebungen verwendet, um die Vorteile des modellgetriebenen Entwicklungsvorgehens voll ausschöpfen und demonstrieren zu können.

6.1 Beschreibung des Szenarios

Die Geschäftsprozesse aus dem Szenario der Forschung und Lehre und die dazu notwendige dienstorientierte IT-Unterstützung werden am KIT maßgeblich durch das universitätsweite Projekt „Karlsruher Integriertes InformationsManagement“ (KIM) [UKA-KIM, Ma05] vorangetrieben, in dem der Autor mitwirkte. Ziel und Inhalt des Projekts sollen im folgenden Abschnitt kurz beleuchtet werden.

6.1.1 Das Projekt „Karlsruher Integriertes InformationsManagement“

Das KIM-Projekt wurde im Jahre 2005 mit dem Ziel, eine Verbesserung der Exzellenz in Forschung und Lehre zu erreichen, am KIT etabliert. Im Fokus stehen seit Beginn des Projekts insbesondere die organisationsübergreifenden Geschäftsprozesse der Lehre und Lehrorganisation sowie deren durchgängige Unterstützung durch IT. Aufgrund der dezentralen Organisationsform des KIT und der damit einhergehenden Heterogenität der vorhandenen IT-Systeme stellt das Ziel einer durchgängigen IT-Unterstützung der Geschäftsprozesse eine nicht unerhebliche organisatorische und technische Herausforderung dar. Die Funktionalität der vorhandenen IT-Systeme, die durch verschiedenste Organisationseinheiten betrieben werden, genügt in den meisten Fällen den lokal vorhandenen Anforderungen. Jedoch fordert der zunehmende Wettbewerb der Hochschulen um die besten Studierenden ein Umdenken, unter anderem im Hinblick auf die zum Einsatz kommende IT-Unterstützung. Effiziente und flexible Geschäftsprozesse, die den Alltag der Studierenden, aber auch aller Mitarbeiter einer Hochschule erleichtern, müssen sowohl organisatorisch als auch technisch ermöglicht werden.

Im KIM-Projekt wird daher das Ziel verfolgt, eine Erhöhung der Exzellenz in der Lehre durch ein besseres Verständnis der vorhandenen Geschäftsprozesse, aber auch der vorhandenen IT-Systeme zu erreichen. Über dieses Verständnis soll eine Optimierung der Zusammenarbeit verschiedener Organisationseinheiten, wie beispielsweise zwischen den zentralen Verwaltungsabteilungen und den einzelnen Fakultäten, angestrebt und damit eine erhöhte Konsistenz und eine verbesserte Unterstützung der Geschäftsprozesse ermöglicht werden. Grundlage dieses Ziels bildet eine integrative IT-Unterstützung in Form einer dienstorientierten Architektur. Die verteilt angebotenen Funktionalitäten der unterschiedlichen IT-Systeme sollen durch eine gemeinsame dienstorientierte Architektur integriert und als einheitliche IT-Unterstützung für die Geschäftsprozesse aller Aufgabenbereiche des KIT bereitgestellt werden. Einer dieser Aufgabenbereiche im Kontext der Lehre und Lehrorganisation ist die Verwaltung von Prüfungen und Prüfungsleistungen, die Studierende am KIT ablegen. Dieser Aufgabenbereich soll zu einer weiteren Konkretisierung des verwendeten Szenarios dienen.

6.1.2 Prüfungsverwaltung einer Hochschule

Die Verwaltung von Prüfungen und Prüfungsleistungen von Studierenden stellt eine Kernaufgabe einer Hochschule dar. Von der Erstellung einer Prüfungsordnung eines Studiengangs bis hin zur Ausstellung eines Zeugnisses zur Bestätigung aller erbrachten Prüfungsleistungen sind unterschiedlichste Geschäftsprozesse im Aufgabenbereich der Prüfungsverwaltung angesiedelt. In nahezu allen Geschäftsprozessen dieses Aufgabenbereiches ist die Gruppe der Studierenden als zentrale Geschäftsrolle involviert. Nach einer im Rahmen des KIM-Projekts durchgeführten Umfrage unter rund 300 Studierenden des KIT fordern Studierende insbesondere im Bereich der Studienassistenz eine deutliche Verbesserung der angebotenen Dienstleistungen und eine bessere IT-Unterstützung der in diesem Bereich vorhandenen Geschäftsprozesse [AB+08]. Ein Teil dieser Geschäftsprozesse kann bereits heute durch einzelne am KIT vorhandene IT-Systeme unterstützt werden. Jedoch führt die Tatsache, dass diese IT-Systeme als Einzellösungen und nicht integriert in einer gemeinsamen Softwarearchitektur betrieben werden, zu einer unflexiblen IT-Unterstützung der Geschäftsprozesse. Zusätzlich muss ein nicht unbeachtlicher Anteil der Arbeitseinheiten der Geschäftsprozesse aufgrund nicht vorhandener oder nicht geeignet eingesetzter IT-Systeme immer noch manuell ohne jegliche Unterstützung durch die IT durchgeführt werden.

Als Beispiel sei der Geschäftsprozess der Anmeldung eines Studierenden zu einer Individualprüfung erwähnt, für den heute keine durchgängige IT-Unterstützung gegeben ist. Die Ausführung des Geschäftsprozesses „Prüfungsanmeldung“ erfordert daher von allen Prozessbeteiligten die Abarbeitung vieler manueller Arbeitseinheiten, deren Abfolge zusätzlich durch die Prozessbeteiligten eigenständig koordiniert werden muss. Da eine durchgängige IT-Unterstützung den Prozessbeteiligten eine vereinfachte und beschleunigte Abwicklung des Geschäftsprozesses ermöglichen würde, greift diese Arbeit den Geschäftsprozess „Prüfungsanmeldung“ auf und zeigt dessen Umsetzung für die im KIM-Projekt verfolgte dienstorientierte Architektur auf. Im Fokus stehen dabei Individualprüfungen, bei denen ein Studierender mit einem Hochschullehrer einen dedizierten Prüfungstermin vereinbaren muss.

6.2 Modellgetriebene Entwicklung eines Geschäftsprozesses

Der Geschäftsprozess „Prüfungsanmeldung“ soll als Workflow auf die am KIT vorhandenen IT-Systeme umgesetzt werden, um eine bessere Unterstützung aller beteiligten Geschäftsrollen bei der Ausführung des Geschäftsprozesses zu erreichen. Für eine zielgerichtete Umsetzung des Geschäftsprozesses müssen die in Kapitel 4 und 5 entwickelten Modelle und Transformationen im Rahmen eines Vorgehensmodells zur Softwareentwicklung angewendet werden. Als eine mögliche Ausprägung eines Vorgehensmodells wird die in Abschnitt 2.2.4 eingeführte geschäftsgetriebene Entwicklung aufgegriffen und die Anwendung der konzipierten Modelle und Transformationen entlang der durch die geschäftsgetriebene Entwicklung vorgegebenen Phasen demonstriert. Die dabei im Folgenden eingeführten Entwicklungsrollen, die zur Ausführung des Entwicklungsvorgehens beitragen, sind dabei ebenso wie das Vorgehensmodell an sich als beispielhafte Ausprägung zu verstehen. Je nach Anwendungsszenario können die entwickelten Modelle und Transformationen für unterschiedliche Vorgehensmodelle und verschiedene Entwicklungsrollen angepasst werden.

6.2.1 Überblick über das Entwicklungsvorgehen

Die geschäftsgetriebene Entwicklung sieht im Kern fünf zentrale Phasen vor. In der **Modellierungsphase** werden die Geschäftsziele und -anforderungen eines Unternehmens identifiziert und darauf aufbauend die resultierenden Geschäftsprozesse modelliert. In der **Entwicklungsphase** werden die

modellierten Geschäftsprozesse dann schrittweise mittels Transformationen auf Softwareartefakte für die IT-Systeme einer bestehenden IT-Umgebung abgebildet. In der **Inbetriebnahmephase** werden die entwickelten Softwareartefakte schließlich in den Betrieb überführt und in der **Monitorphase** gestestet und überwacht. Ergeben sich in der Monitorphase Änderungen an den Modellen, so werden diese in der **Analyse-** und **Adaptionsphase** erfasst und in einem weiteren Entwicklungszyklus der Modellierungsphase zur Verfügung gestellt. Die geschäftsgetriebene Entwicklung verfolgt damit entsprechend den Prinzipien der modellgetriebenen Softwareentwicklung einen *Forward Engineering*-Ansatz, bei dem Änderungen durch iterative Zyklen immer in den Ausgangsmodellen reflektiert und durch Transformationen auf die Folgemodelle propagiert werden (vgl. [MM03, PM06, SV05]).

Zur Umsetzung des geschäftsgetriebenen Entwicklungsvorgehens müssen den beteiligen Entwicklungsrollen geeignete Werkzeuge zur Spezifikation der Modelle, aber auch zur Ausführung der Transformationen zur Verfügung gestellt werden. Als Modellierungswerkzeug kommt hierbei der von IBM entwickelte **Rational Software Architect** (RSA) [IBM-RSA] zum Einsatz, der auf der Entwicklungsumgebung Eclipse der *Eclipse Foundation* [EF-Eclipse] aufbaut. Die Ausführung der Modell-zu-Modell-Transformationen wird durch das Open-Source-Werkzeug **MediniQVT** [IKV-Medini] übernommen, das als *Plugin* für Eclipse den Sprachteil QVT-*Relations* des QVT-Standards implementiert. Zur Ausführung der Modell-zu-Text-Transformationen dient das ebenfalls als Eclipse-*Plugin* implementierte Werkzeug **Open Architecture Ware** (OAW) [OAW]. Um ein besseres Gesamtverständnis über die verwendeten Modelle und Transformationen im Kontext des geschäftsgetriebenen Entwicklungsvorgehens erreichen zu können, gibt Abbildung 79 einen Überblick über das gesamte Vorgehen anhand des aus den vorangegangenen Kapiteln bekannten Schaubildes.

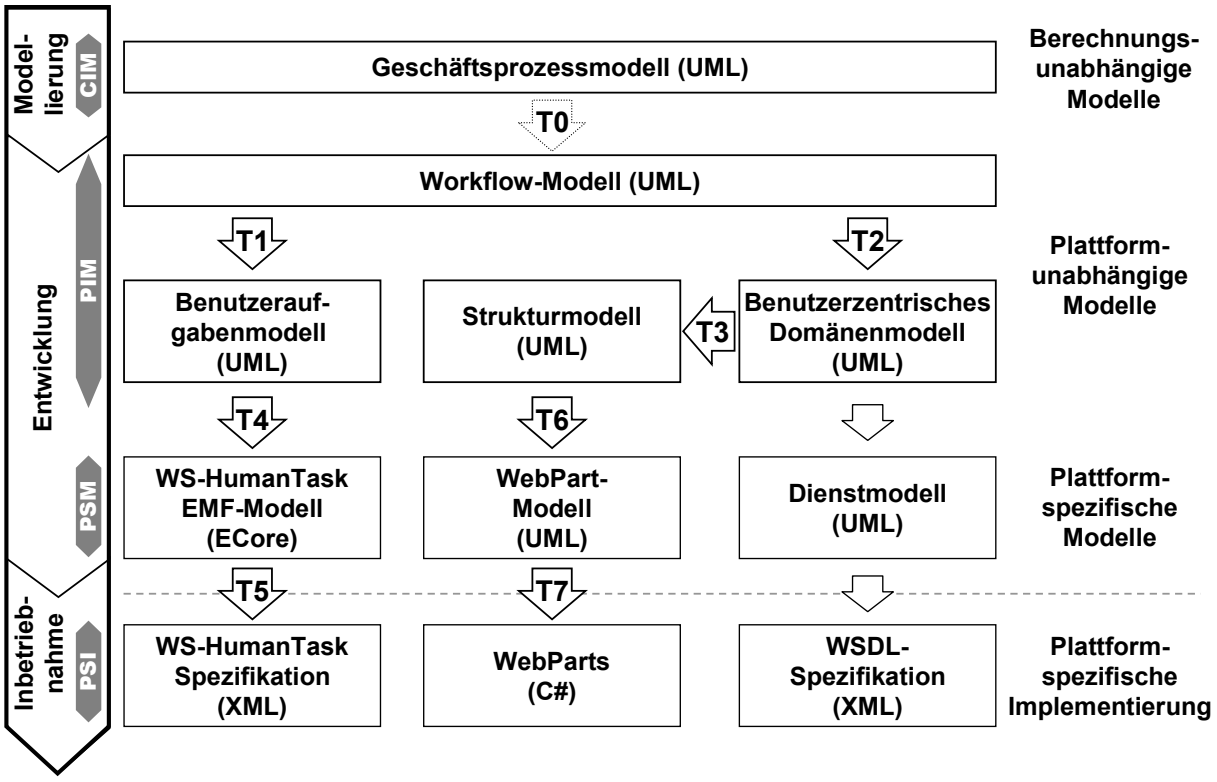

Abbildung 79: Überblick über das Entwicklungsvorgehen

Das Hauptaugenmerk des Tragfähigkeitsnachweises liegt auf der Entwicklungs- und Inbetriebnahmephase des geschäftsgetriebenen Entwicklungsvorgehens, da in diesen Phasen die in Kapitel 4 und 5 konzipierten Modelle und Transformationen zur Anwendung kommen.

Zur Umsetzung des plattformunabhängigen Benutzeraufgabenmodells auf das WS-HumanTask-EMF-Modell und die WS-HumanTask-Spezifikation kommen die in Abschnitt 5.4 entwickelten Transformationen T4 und T5 zum Einsatz. Da aktuell jedoch keine Referenzimplementierung der WS-HumanTask-Spezifikation zur Verfügung steht, zeigt Abschnitt 6.2.5 zusätzlich die Ausführung einer leicht modifizierten Transformation T5 auf, die das WS-HumanTask-EMF-Modell auf die durch IBM spezifizierte Benutzeraufgabenausführungssprache ITEL (IBM *Task Expression Language*) abbildet. Die auf diese Weise erzeugten ITEL-Spezifikationen können auf dem WebSphere Process Server von IBM [IBM-WPS] in Betrieb genommen werden.

Da aufgrund eines bisher fehlenden einheitlichen Standards zur Spezifikation von Benutzerschnittstellen in den vorangegangenen Kapiteln die Transformation eines plattformunabhängigen Strukturmodells auf ein plattformspezifisches Strukturmodell nicht beleuchtet wurde, zeigt Abschnitt 6.2.6 anhand der Anwendung der Modell-zu-Modell-Transformation T6 die Erzeugung eines plattformspezifischen Strukturmodells am Beispiel von ASP.NET auf. Im Anschluss an T6 wird das plattformspezifische Strukturmodell durch eine Modell-zu-Text-Transformation T7 in OAW auf ausführbare Softwareartefakte der Programmiersprache C# überführt, die unter anderem in dem durch Microsoft entwickelten Office SharePoint Server betrieben werden können.

6.2.2 Vorbereitende Maßnahmen

Bevor die beteiligten Entwicklungsrollen mit der Ausführung des geschäftsgetriebenen Entwicklungsvorgehens beginnen können, sind einige vorbereitende Maßnahmen durchzuführen. So müssen beispielsweise die in Kapitel 4 konzipierten UML-Profile für das gewählte Modellierungswerkzeug RSA durch qualifizierte MDSD-Experten implementiert werden, damit den Anwendungsentwicklern die entsprechenden Stereotypen zur Spezifikation der Modelle zur Verfügung stehen. In seiner aktuellen Version 7.5 bietet der RSA hierzu einen grafischen Editor an, der eine einfache und intuitive Implementierung der UML-Profile ermöglicht. Abbildung 80 zeigt beispielhaft die Implementierung des Benutzeraktionsprofils in RSA.

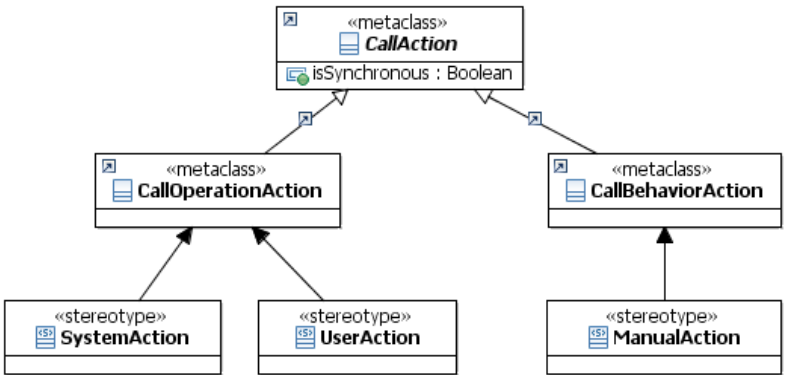

Abbildung 80: Benutzeraktionsprofil in Rational Software Architect

Wurden alle Profile für RSA implementiert, so stehen deren Stereotypen den Anwendungsentwicklern fortwährend zur Modellierung in RSA zur Verfügung. Da der RSA UML-Profile aber auch UML-Modelle in einem jeweils eigenen Dateiformat speichert, müssen alle Profile nach der Modellierung entsprechend dem OMG-eigenen Standard *XML Metadata Interchange* (XMI) [OMG-XMI] serialisiert werden, damit die Profile in anderen Entwicklungsumgebungen wie etwa zur Transformation in MediniQVT genutzt werden können. Ferner müssen die in Kapitel 5 entwickelten Transformationen für MediniQVT bzw. OAW implementiert werden, sodass diese von den Anwendungsentwicklern im

geschäftsgetriebenen Entwicklungsprozess angewendet werden können. Bei der Umsetzung der entwickelten Transformationen auf MediniQVT ist zu beachten, dass die zum Zeitpunkt der Erstellung dieser Arbeit aktuelle Version 1.6 von MediniQVT Stereotypen nicht erkennen und daher nur auf den Elementen des UML-Metamodells operieren kann. Um diese fehlende Funktionalität überbrücken zu können, stehen im Rahmen von Eclipse einige Hilfsmethoden zur Verarbeitung von Stereotypen zur Verfügung, die bei der Implementierung der Transformationen angewendet werden können. Anhang B.1 geht auf die Verwendung dieser Hilfsmethoden und die daraus resultierenden Anpassungen der Transformationen detailliert ein.

Durch die notwendigen vorbereitenden Maßnahmen kann erneut der wesentliche Unterschied zwischen traditionellen und modellgetriebenen Entwicklungsprozessen verdeutlicht werden. Während in traditionellen Softwareentwicklungsprozessen ein Großteil der Entwicklung direkt auf der Ebene des Quellcodes stattfindet, gestatten modellgetriebene Entwicklungsvorgehen eine Abstraktion von dieser Ebene durch die Verwendung von Modellen. Bis zu einem gewissen Grad kann damit die Komplexität für den Anwendungsentwickler reduziert und hinter den zur Verfügung gestellten Modellelementen verschattet werden. Gleichzeitig können durch die Spezifikation wiederverwendbarer Transformationen für ähnlich gelagerte Problemstellungen automatisiert Lösungen erzeugt werden. Dennoch müssen, wie in Kapitel 4 und 5 gezeigt, die entsprechenden Modelle und Transformationen zunächst entwickelt und den Anwendungsentwicklern zur Verfügung gestellt werden. Ein Teil des Aufwandes und der Komplexität wird damit aus den traditionellen Entwicklungsvorgehen in ein getrennt zu betrachtendes Entwicklungsvorgehen für Metamodelle und Transformationen verlagert. Es muss daher darauf geachtet werden, dass zur Anwendung der modellgetriebenen Prinzipien neben den eigentlichen Anwendungsentwicklern qualifizierte Fachkräfte im Sinne von MDSD-Experten vorhanden sind, welche die benötigten Modelle und Transformationen kontinuierlich weiterentwickeln können. Ändert sich beispielsweise ein Plattformmodell einer Zielplattform durch einen Versionswechsel des Herstellers, so muss eine entsprechend qualifizierte Entwicklungsrolle alle betroffenen Transformationen an die neue Zielplattform anpassen. Diese Änderungen haben jedoch häufig keine Auswirkung auf den Anwendungsentwickler, da für ihn viele Änderungen hinter den Modellelementen verborgen bleiben.

Nachdem durch die Implementierung aller Transformationen und die Umsetzung der benötigten Metamodelle alle vorbereitenden Maßnahmen abgeschlossen sind, kann der geschäftsgetriebene Entwicklungsprozess durch Anwendungsentwickler, die jeweils unterschiedliche Entwicklungsrollen einnehmen können, durchgeführt werden. Die nächsten Abschnitte folgen daher den eingangs erläuterten Phasen der geschäftsgetriebenen Entwicklung und zeigen unter Zuhilfenahme der entwickelten Modelle der Benutzeraktion, der bereitgestellten Transformationen und der erwähnten Werkzeuge die Umsetzung des Geschäftsprozesses „Prüfungsanmeldung“ auf eine dienstorientierte Architektur, wie sie als Realisierung der IT-Unterstützung im KIM-Projekt eingesetzt wird, auf.

6.2.3 Spezifikation des Geschäftsprozessmodells

Im Rahmen einer ersten Modellierungsphase müssen zuerst allgemeine Geschäftsanforderungen an den Geschäftsprozess „Prüfungsanmeldung“ sowie dessen wesentliche Arbeitsschritte aus einer grobgranularen Perspektive erfasst werden. Hierbei ist zu Beginn der Modellierungsphase keine formale Spezifikation des Geschäftsprozesses gefordert, weshalb die beteiligten Entwicklungsrollen, wie etwa ein Geschäftsprozessanalyst oder ein Domänenexperte, zunächst ein berechnungsunabhängiges Modell des Geschäftsprozesses mithilfe einfachster Werkzeuge wie etwa Bleistift und Papier erarbeiten können. In mehreren innerhalb der Modellierungsphase durchgeführten Iterationen muss dann das Verständnis über die Details des Geschäftsprozesses schrittweise konkretisiert und verfeinert

werden, sodass letztendlich eine formale Spezifikation im Sinne eines Geschäftsprozessmodells entwickelt werden kann. In diesem Zusammenhang sind unterschiedliche Fragestellungen, von einer geeigneten Aufteilung der einzelnen Prozessschritte auf Geschäftsrollen, bis hin zu den in der Domäne des Geschäftsprozesses bereits vorhandenen Diensten zu berücksichtigen. Diese umfangreichen Fragestellungen werden, wie bereits in Kapitel 4 angedeutet, von dieser Arbeit bewusst ausgeblendet und durch die in Abbildung 79 dargestellte manuelle Transformation T0 verschattet. Abschnitt 7.3.3 greift daher diese Fragestellungen im Ausblick der Arbeit erneut auf.

Ausgangspunkt des demonstrierten Entwicklungsvorgehens ist ein bereits formalisierter und an die Zieldomäne der Lehre und der Lehrorganisation des KIT angepasster Geschäftsprozess. Abbildung 81 zeigt den wesentlichen Ausschnitt des Geschäftsprozesses „Prüfungsanmeldung“ in Form eines UML-Aktivitätsdiagramms, das durch einen Geschäftsprozessanalysten mithilfe des RSA in Zusammenarbeit mit einem Prozessverantwortlichen des KIT als Kunde erstellt wurde und als Ausgangspunkt der Demonstration der Tragfähigkeit der in dieser Arbeit entwickelten Konzepte dienen soll.

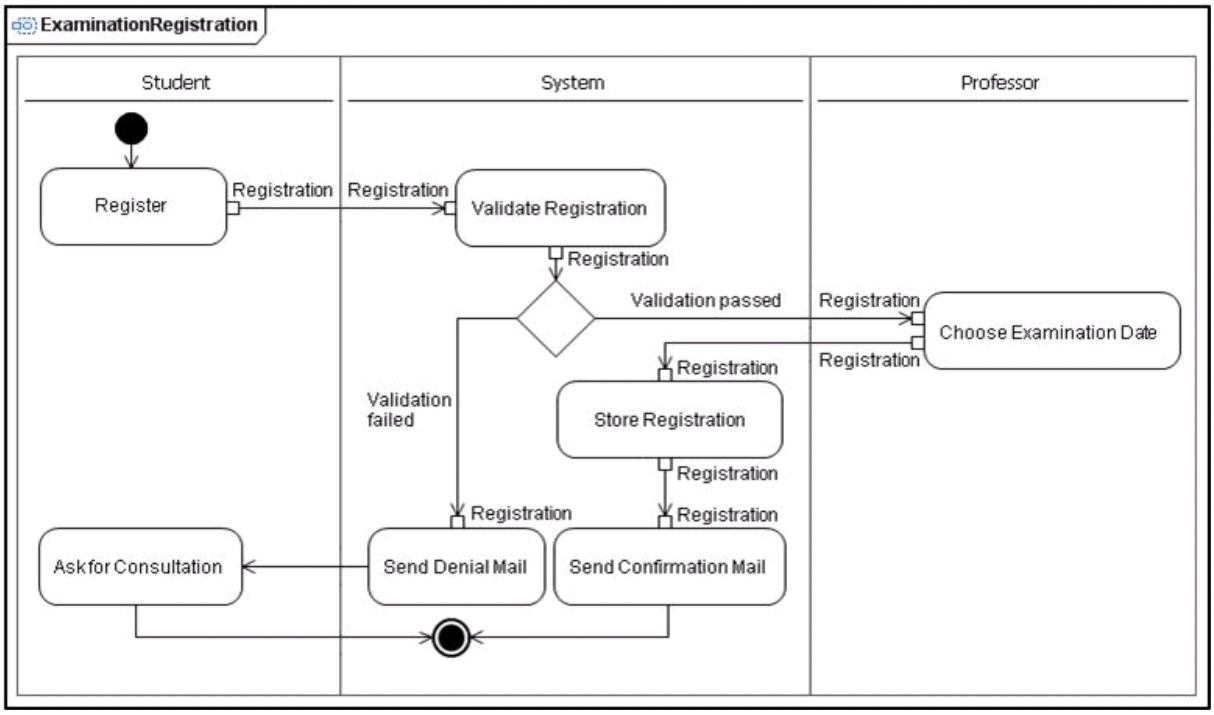

Abbildung 81: Geschäftsprozess „Prüfungsanmeldung“

Der bewusst einfach gehaltene Geschäftsprozess beginnt mit der Anmeldung eines Studierenden zu einer Prüfung (`Register`). In einem zweiten Schritt wird von einem IT-System automatisch überprüft, ob die Anmeldung des Studierenden syntaktisch vollständig und gültig ist (`Validate Registration`). Hat ein Studierender beispielsweise ein vorgeschriebenes Fachsemester oder eine benötigte Mindestpunktzahl in seinem Studium noch nicht erreicht, so kann die Anmeldung automatisch abgelehnt werden (`Validation failed`). In diesem Fall wird dem Studierenden eine E-Mail zugestellt, die ihn über die Gründe der verweigerten Anmeldung informiert (`Send Denial Mail`). Benötigt der Studierende nach der verweigerten Anmeldung weitere Informationen, so kann er anschließend eine Beratungsstelle aufsuchen, um sich über weitere Schritte zu informieren (`Ask for Consultation`). Sind alle Vorbedingungen für die Anmeldung des Studierenden zur Prüfung erfüllt (`Validation passed`), muss in einem nächsten Schritt der zuständige Hochschullehrer einen Prüfungstermin für den Studierenden wählen (`Choose Examination Date`). Sobald dies ge-

schehen ist, wird die Prüfungsanmeldung mit dem gewählten Prüfungstermin vom System persistiert (`Store Registration`) und abschließend dem Studierenden eine E-Mail mit allen relevanten Daten zur Prüfung zugestellt (`Send Confirmation Mail`).

6.2.4 Spezifikation des Workflow-Modells

Damit der im vorherigen Abschnitt spezifizierte Geschäftsprozess „Prüfungsanmeldung“ auf die am KIT vorhandenen IT-Systeme abgebildet werden kann, muss festgelegt werden, welche Teile des Geschäftsprozesses prinzipiell durch Workflows unterstützt werden können bzw. sollen. Hierbei kann der Domänenexperte, der über ein detailliertes Wissen über die am KIT vorhandenen IT-Systeme und deren Dienste verfügt, die Stereotypen des **Benutzeraktionsprofils** verwenden, um in RSA die einzelnen Arbeitsschritte des Geschäftsprozesses entsprechend als System-, Benutzer- oder manuelle Aktionen auszuzeichnen. Abbildung 82 zeigt das Ergebnis dieses Verfeinerungsschrittes in RSA.

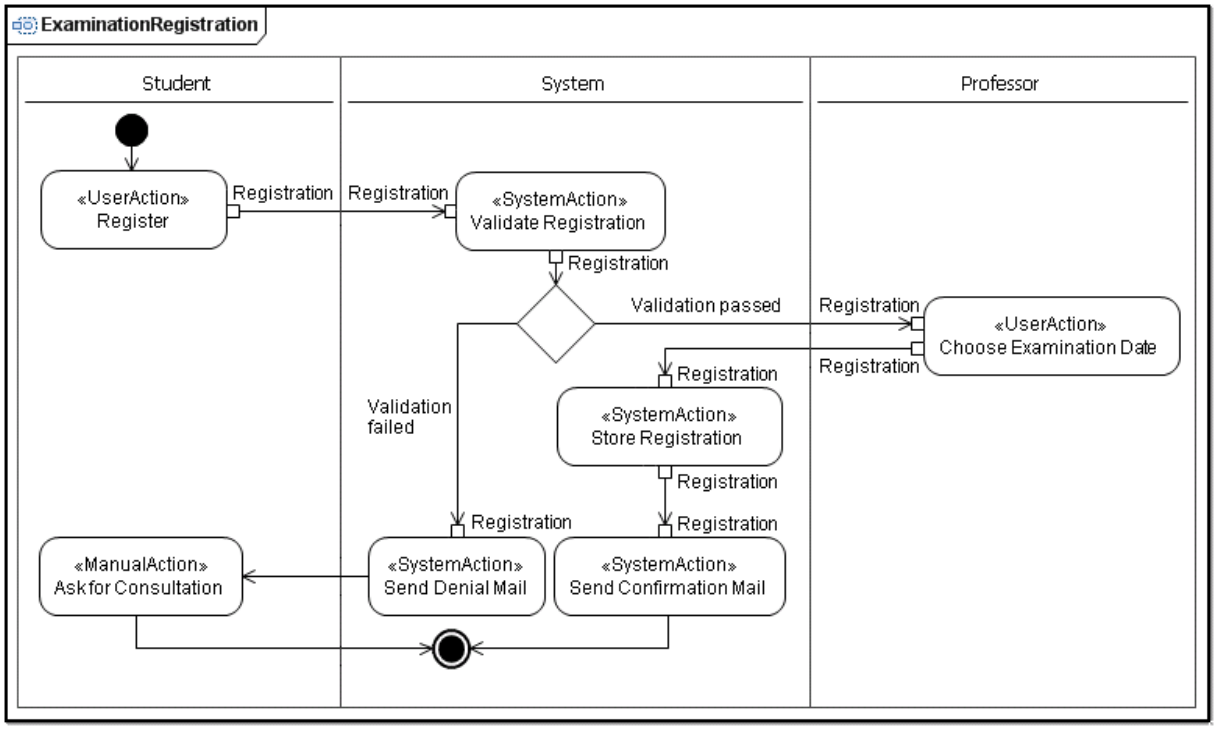

Abbildung 82: Geschäftsprozess „Prüfungsanmeldung“ mit stereotypisierten Aktionen

Anhand dieser Verfeinerung wird für den Menschen aber auch für eine Transformationsausführungsumgebung ersichtlich, welche Teile des Geschäftsprozesses auf Workflows abzubilden sind. Da manuelle Aktionen nicht bzw. noch nicht durch ein IT-System unterstützt werden können oder sollen, werden diese bei der Abbildung des Geschäftsprozessmodells auf Workflow-Modelle ausgeblendet. Im Falle des Geschäftsprozessmodells „Prüfungsanmeldung“ erfordert diese Abbildung nur einen geringen manuellen Aufwand, da der Geschäftsprozessanalyst lediglich die Kontrollflusskante der Systemaktion `Send Denial Mail` direkt mit dem Endknoten des Workflow-Modells verbinden und die manuelle Aktion `Ask for Consultation` entfernen muss. Das auf diese Weise entwickelte Workflow-Modell „Prüfungsanmeldung“ dient als Ausgangsmodell zur automatisierten Erzeugung der Benutzeraufgaben- und Benutzerschnittstellenspezifikation, womit gleichzeitig der Abschluss der Modellierungsphase und der Übergang in die Entwicklungsphase des geschäftsgetriebenen Entwicklungsvorgehens erfolgt.

Da sowohl das Benutzeraufgaben- als auch das Benutzerschnittstellenmodell jeweils einen eigenständigen Aspekt der Benutzerinteraktion konkretisieren, können vom gemeinsamen Workflow-Modell ausgehend beide Aspekte durch entsprechende Fachentwickler in parallelen Entwicklungszweigen vorangetrieben werden. Allerdings muss darauf geachtet werden, dass Änderungsanforderungen, die gegebenenfalls nachträglich in einem Entwicklungszweig entstehen und das Workflow-Modell als gemeinsames Ausgangsmodell betreffen, durch geeignete Maßnahmen in alle weiteren Entwicklungszweige propagiert werden. Die folgenden beiden Abschnitte zeigen die Erzeugung der Benutzeraufgaben- und Benutzerschnittstellenspezifikationen entlang der in Abbildung 79 aufgezeigten beiden Entwicklungszweige auf.

6.2.5 Erzeugung der Benutzeraufgabenspezifikation

Die Erzeugung der Benutzeraufgabenspezifikationen für die Benutzeraktionen des Workflows „Prüfungsanmeldung“ erfolgt durch ein dreistufiges Transformationsvorgehen. Die erste Modell-zu-Modell-Transformation T1 überführt das Workflow-Modell „Prüfungsanmeldung“ in ein plattformunabhängiges Benutzeraufgabenmodell, während die zweite Modell-zu-Modell-Transformation T4 aus dem plattformunabhängigen Benutzeraufgabenmodell das plattformspezifische WS-HumanTask-EMF-Modell erzeugt. Durch die Nutzung des *Eclipse Modeling Framework* kann, wie in Abschnitt 5.4 erörtert, die zuletzt benötigte Modell-zu-Text-Transformation T5 als gegeben hingenommen werden. Da für die WS-HumanTask-Spezifikation zum Zeitpunkt der Erstellung dieser Arbeit keine Referenzimplementierung zur Verfügung steht, wird abschließend eine alternative Transformation auf die ebenfalls XML-basierte Ausführungssprache *Inline Task Expression Language* (ITEL) [KB+07] von IBM aufgezeigt.

6.2.5.1 Erzeugung des plattformunabhängigen Benutzeraufgabenmodells

Nachdem das abzubildende Workflow-Modell nach mehreren Iterationen innerhalb der Modellierungsphase einen stabilen Zustand erreicht hat, kann der Geschäftsprozessanalyst durch die Erzeugung des plattformunabhängigen Benutzeraufgabenmodells den ersten Entwicklungsschritt der Entwicklungsphase anstoßen. Hierzu nutzt er das Transformationswerkzeug MediniQVT und führt die Modell-zu-Modell-Transformation T1 aus. Wird T1 auf das Workflow-Modell „Prüfungsanmeldung“ angewendet, so erzeugt MediniQVT ein neues UML-Modell, das die beiden Benutzeraufgaben `Register` und `Choose Examination Date` entsprechend den im Workflow-Modell spezifizierten Benutzeraktionen, enthält. Abbildung 83 zeigt einen Auszug des Benutzeraufgabenmodells für die Benutzeraktion `Choose Examination Date` in RSA als ein Ergebnis von T1. Wie an dem dargestellten Auszug zu erkennen ist, unterstützt RSA die Verwendung einer eigenen grafischen Notation über die sogenannten *Shape Images* nur bei Stereotypen, jedoch nicht bei den Metaklassen der UML. Dementsprechend ist der Akteur `Professor` in der RSA-eigenen grafischen Notation dargestellt. Ferner blendet der RSA bei der Nutzung eigener grafischer Symbole die Anzeige des Stereotyps eines Modellelements wie beispielsweise `<<UserTask>>` aus. Daher muss darauf geachtet werden, dass die Symbole für den Entwickler eindeutig und gut zu unterscheiden sind.

Wie ebenfalls an Abbildung 83 zu erkennen ist, können nicht alle für das Benutzeraufgabenmodell benötigten Informationen aus dem Workflow-Modell als Quellmodell der Transformation T1 abgeleitet werden. Daher muss das erzeugte Benutzeraufgabenmodell im Anschluss an die Transformation durch einen entsprechenden Fachentwickler, der mit der zugrunde liegenden Organisationsstruktur des KIT gut vertraut ist und daher im Folgenden als Ressourcenexperte bezeichnet wird, um zusätzliche Informationen angereichert und beispielsweise der bisher nur abstrakt spezifizierte Akteur

`Professor` durch menschliche Ressourcen im Sinne einer eine Rolle oder durch einen konkreten Benutzer ausgeprägt werden. Ferner kann der Ressourcenexperte das Benutzeraufgabenmodell in RSA um verschiedene Modellelemente, die ihm das **Benutzeraufgabenprofil** zur Verfügung stellt, erweitern und beispielsweise weitere Eskalationen zu den Benutzeraufgaben hinzufügen oder zusätzliche Aufgabenrollen wie etwa einen Delegierten oder einen Assistenten spezifizieren.

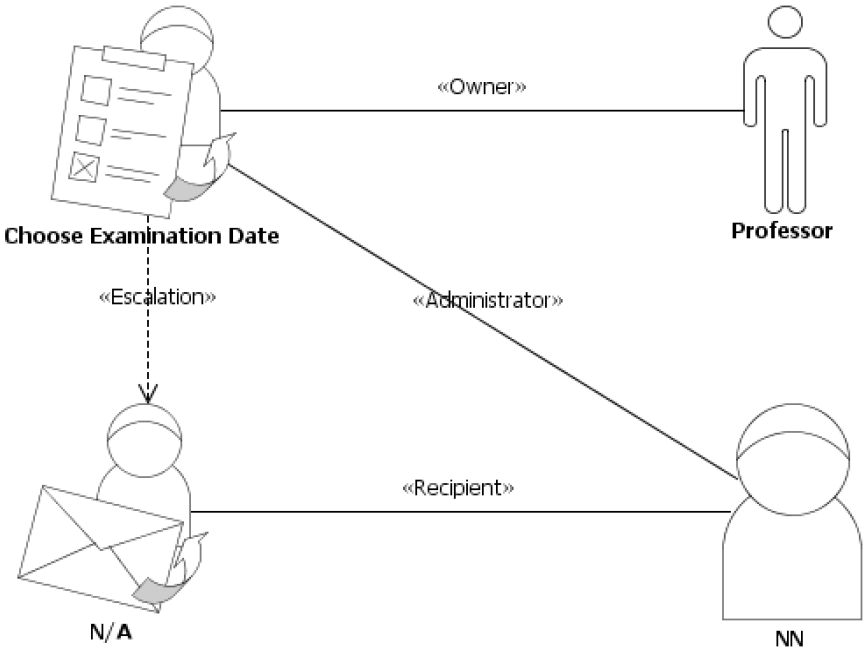

Abbildung 83: Auszug des Benutzeraufgabenmodells des Workflows "Prüfungsanmeldung"

Bei der Verfeinerung des Modells muss allerdings darauf geachtet werden, dass die durchgeführten Änderungen nicht die Konsistenz zum Quellmodell gefährden. Würde der Ressourcenexperte beispielsweise die Benutzeraufgabe `Choose Examination Date` im Benutzeraufgabenmodell durch eine andere Benutzeraufgabe ersetzen, so wäre das Benutzeraufgabenmodell nicht mehr zum Workflow-Modell als Quellmodell konsistent. Geeignete Gegenmaßnahmen zur Vermeidung solcher Inkonsistenzen müssen je nach verfolgtem Entwicklungsvorgehen und den verwendeten Entwicklungswerkzeugen ergriffen werden.

Eine sehr einfache Möglichkeit der Konsistenzprüfung bieten dabei die Transformationen selbst. Wurde ein Benutzeraufgabenmodell durch einen Ressourcenexperten verfeinert, so kann dieser im Anschluss an die Verfeinerung eine Transformation T1' ausführen, die bis auf eine Auszeichnung beider Domänen als *checkonly* zu T1 identisch ist (vgl. Abschnitt 5.2.1). T1' überprüft somit lediglich, ob das verfeinerte Benutzeraufgabenmodell noch konsistent zu seinem Quellmodell ist. Hat der Ressourcenexperte irrtümlicherweise aus dem in Abbildung 83 dargestellten Benutzeraufgabenmodell die Benutzeraufgabe `Choose Examination Date` entfernt, so liefert Transformation T1' den booleschen Wert *false* zurück und weist damit den Ressourcenexperten auf seinen Fehler hin. Je nach eingesetzter Entwicklungsumgebung ist somit ebenfalls denkbar, dass eine Überprüfung der Konsistenz eines Modells nach einer Veränderung des Quell- oder Zielmodells automatisch überprüft wird.

6.2.5.2 Erzeugung des WS-HumanTask-EMF-Modells

Aus dem plattformunabhängigen Benutzeraufgabenmodell, das durch einen Ressourcenexperten verfeinert wurde, wird zu Beginn des nächsten Entwicklungsschritts durch die Modell-zu-Modell-Transformation T4 ein plattformspezifisches WS-HumanTask-EMF-Modell erzeugt. Dementspre-

chend muss zuvor Transformation T4 in MediniQVT implementiert und das *ECore*-Modell zu WS-HumanTask in RSA bereitgestellt worden sein. Die hierzu notwendige Umsetzung der in Abschnitt 5.4.2 entwickelten Relationen von Transformation T4 sowie nähere Hinweise zur Erzeugung des WS-HumanTask-*ECore*-Modells sind in [Ho08, Ry09] zu finden. Das Ergebnis der Anwendung von T4 auf das Benutzeraufgabenmodell zum Workflow „Prüfungsanmeldung“ ist ein WS-HumanTask-EMF-Modell, das auszugsweise für die Benutzeraufgabe `Choose Examination Date` in Abbildung 84 dargestellt ist.

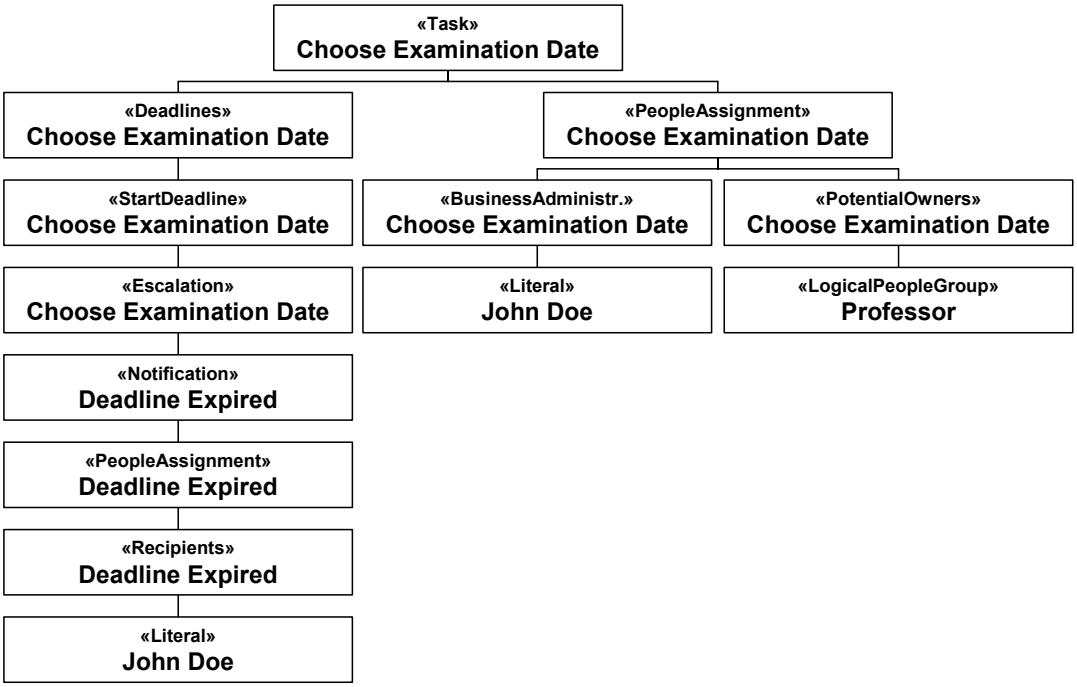

Abbildung 84: WS-HumanTask-EMF-Modell des Workflows Prüfungsanmeldung

Im erzeugten WS-HumanTask-EMF-Modell besteht nun erneut die Möglichkeit, Anpassungen oder Optimierungen der Spezifikation einer Benutzeraufgabe durchzuführen, die speziell im Kontext der Zielplattform WS-HumanTask gegebenenfalls notwendig werden. Hierzu kann ein qualifizierter WS-HumanTask-Experte als weitere am Entwicklungsvorgehen beteiligte Entwicklungsrolle das EMF-Modell in RSA erweitern und die syntaktische Korrektheit seiner Erweiterungen durch das zugrunde liegende WS-HumanTask-*ECore*-Modell ebenfalls in RSA überprüfen.

Die abschließende Transformation des WS-HumanTask-EMF-Modells auf ausführbaren Quellcode wird durch die Vorgehensweise des EMF bei der Serialisierung der EMF-Modelle stark vereinfacht. Da EMF die Beziehungen zwischen ursprünglichem XML-Schema und daraus erzeugtem *ECore*-Modell in den Metadaten des *ECore*-Modells persistiert, kann ein zu dem *ECore*-Modell konformes EMF-Modell nach dem ursprünglichen XML-Schema serialisiert werden und liegt damit als zu dem vorgegebenen XML-Schema valider XML-Quellcode vor (vgl. dazu Abschnitt 5.4.3). Eine eigenständige Modell-zu-Text-Transformation wird daher nicht benötigt. Sobald der WS-HumanTask-Experte die Bearbeitung des WS-HumanTask-EMF-Modells abgeschlossen hat und das Modell speichert, wird dieses automatisch konform zu dem XML-Schema von WS-HumanTask serialisiert. Die Serialisierung des in Abbildung 84 gezeigten EMF-Modells resultiert somit in dem nachfolgend auszugsweise gezeigten WS-HumanTask-XML-Quellcode. Der vollständige WS-HumanTask-XML-Quellcode des Workflows „Prüfungsanmeldung“ kann aus [Ry09] entnommen werden.

```
<wsht:tasks>
  <wsht:task name="Choose Examination Date">
    <wsht:peopleAssignments>
      <wsht:potentialOwners>
        <wsht:from logicalPeopleGroup="_role">
          <wsht:argument name="RoleName">Professor</wsht:argument>
        </wsht:from>
      </wsht:potentialOwners>
      <wsht:businessAdministrators>
        <wsht:from>
          <wsht:literal>
            <wsht:users>
              <wsht:user>John Doe</wsht:user>
            </wsht:users>
          </wsht:literal>
        </wsht:from>
      </wsht:businessAdministrators>
    </wsht:peopleAssignments>
  </wsht:task>
</wsht:tasks>
```

Mit der Erzeugung des WS-HumanTask-XML-Quellcodes ist die Entwicklungsphase der geschäftsgetriebenen Entwicklung für die Benutzeraufgabenmodelle abgeschlossen. In der nächsten Phase des Entwicklungsvorgehens müssen die erzeugten Softwareartefakte auf einer Ausführungsumgebung für WS-HumanTask-Spezifikationen in Betrieb genommen werden. Da zum Zeitpunkt der Erstellung dieser Arbeit jedoch keine Referenzimplementierung für eine solche Ausführungsumgebung verfügbar ist, soll im folgenden Abschnitt durch eine leicht modifizierte Transformation T4' die Erzeugung eines plattformspezifischen Benutzeraufgabenmodells für die durch IBM implementierte Benutzeraufgabenspezifikationssprache ITEL (*Inline Task Expression Language*) [KB+07] aufgezeigt werden.

6.2.5.3 Alternative Erzeugung eines ITEL-EMF-Modells

Da ITEL ebenfalls auf einem XML-Schema beruht, kann analog zur Erzeugung des WS-HumanTask-*ECore*-Modells mittels EMF ein ITEL-*ECore*-Modell erstellt und anschließend als Zielmodell der Transformation genutzt werden. Abbildung 85 zeigt die auf ITEL angepassten Abbildungsbezüge zwischen den Modellelementen des Benutzeraufgabenmodells und des ITEL-*ECore*-Modells (vgl. Abbildung 75, Seite 156). Im Unterschied zu den Abbildungsbezügen zu WS-HumanTask zeigen sich nur einige wenige Änderungen wie etwa am Modellelement `Editor`, das zur Abbildung der Aufgabenrolle Assistent (`Assistance`) verwendet werden kann. Da die Unterschiede zwischen beiden Spezifikationen nur marginal sind, sei für eine Umsetzung der Relationen auf QVT und für die Erzeugung von ausführbarem XML-Quellcode nach ITEL auf [Ho08b] und [Ry09] verwiesen.

Der Dienst der Benutzeraufgabenverwaltung wiederum nutzt die hinterlegte ITEL-Spezifikation und erzeugt auf deren Grundlage eine Instanz der Benutzeraufgabe. Ergeben sich bei der Ausführung von Benutzeraufgaben durch den Menschen Änderungsanforderungen an die Benutzeraufgabenspezifikation, da beispielsweise eine Frist für die Abarbeitung einer Benutzeraufgabe zu früh abläuft, so werden diese durch die Benutzer und die Administratoren an die Anwendungsentwickler übermittelt. Die Anwendungsentwickler überprüfen dann in der Analyse- und Anpassungsphase, inwieweit die Änderungsanforderungen der Nutzer in eine weitere Iteration der geschäftsgetriebenen Entwicklung eingehen und Anpassungen an den Modellen erfordern. Individuelle Verfeinerungen, die in einer Iteration

zuvor an den Modellen vorgenommen wurden, werden bei einer gegebenenfalls notwendig werdenden erneuten Ausführung der Transformationen von den verwendeten Entwicklungsumgebungen durch unterschiedliche Mechanismen wie etwa Markierungen als solche erkannt und bleiben daher auch in einer weiteren Iteration der geschäftsgetriebenen Entwicklung erhalten.

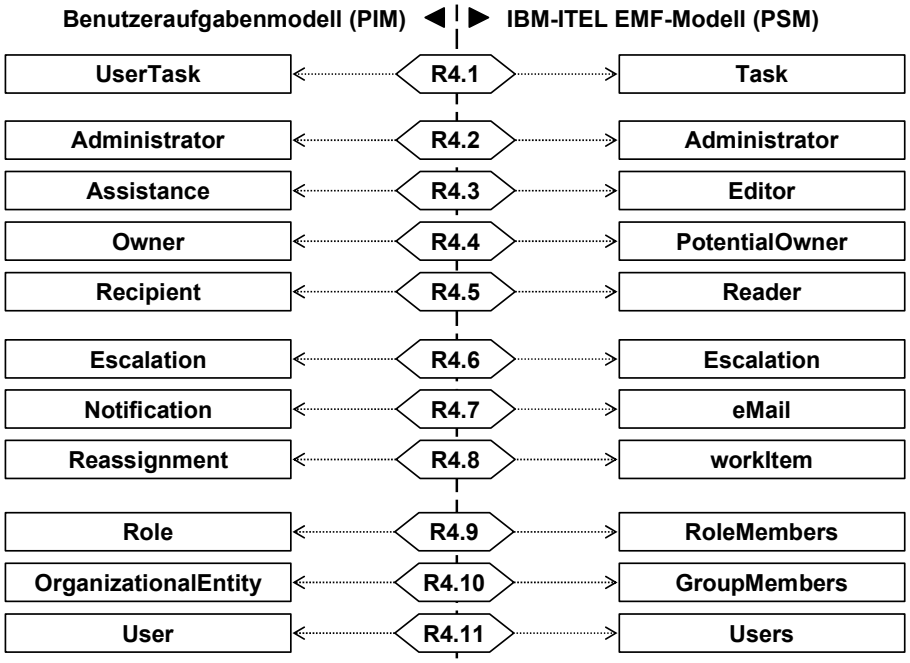

Abbildung 85: Alternative Abbildungsbezüge zu IBM-ITEL

6.2.6 Erzeugung der Benutzerschnittstellenspezifikation

Neben den Benutzeraufgabenspezifikationen können in einem zweiten Entwicklungspfad des geschäftsgetriebenen Entwicklungsvorgehens die zur Ausführung der Benutzeraktionen des Workflows „Prüfungsanmeldung“ benötigten Benutzerschnittstellen erzeugt werden. Eine erste Modell-zu-Modell-Transformation T2 (vgl. Abbildung 79) bildet hierzu das Workflow-Modell auf das benutzerzentrische Domänenmodell ab, das von einer Entwicklungsrolle, die für die Spezifikation von Benutzerschnittstellen besonders qualifiziert ist, in Zusammenarbeit mit dem Kunden um Informationen bezüglich der Benutzerinteraktion erweitert werden muss. Diese Entwicklungsrolle wird daher im weiteren Verlauf als Benutzerschnittstellenexperte bezeichnet. Die zweite Modell-zu-Modell-Transformation T3 überführt im nächsten Transformationsschritt das erweiterte benutzerzentrische Domänenmodell in das plattformunabhängige Strukturmodell der Benutzerschnittstelle, das erneut um zusätzliche Informationen erweitert werden kann. Als konkrete Zielplattform der Benutzerschnittstellen kommt der als *Active Server Pages* .NET (ASP.NET) bezeichnete Teil des .NET-Rahmenwerks von Microsoft und der als Laufzeitumgebung für ASP.NET-Spezifikationen zur Verfügung stehende Office SharePoint Server zum Einsatz, auf dessen Grundlage unter anderem das am KIT zentral angebotene Studierendenportal realisiert wurde (vgl. [AB+08]). Dementsprechend wird eine dritte Modell-zu-Modell-Transformation T6 zur Erzeugung eines plattformspezifischen Strukturmodells für ASP.NET aus dem plattformspezifischen Strukturmodell und eine abschließende Modell-zu-Text-Transformation T7 zur Erzeugung der Benutzerschnittstellenspezifikationen in ASP.NET benötigt.

Erneut kann durch die entwickelten Transformationen ein Großteil der Entwicklungsarbeit, die ursprünglich von den Anwendungsentwicklern zu erbringen gewesen wäre, in die Transformationen ausgelagert werden. Da gerade im Bereich der formularbasierten Benutzerschnittstellen viele Proble-

me eine ähnliche Lösung erfordern, steht der initial zu erbringende Aufwand der Entwicklung der Metamodelle und Transformationen einer kontinuierlichen Anwendung der neuen Modellelemente und Transformationen gegenüber. Die verwendeten Modelle gestatten außerdem, die Aspekte einer Benutzerschnittstelle wie beispielsweise Inhalt und Struktur in getrennten Modellen zu betrachten und ermöglichen damit durch die Integration spezialisierter Fachentwickler die Sicherstellung der Qualität der entwickelten Softwareartefakte. Ferner kann, wie im Folgenden gezeigt wird, nach einer ersten Erfassung der Anforderungen des Kunden an die Benutzerschnittstelle durch die Transformationen rasch ein erster funktionaler Entwurf der Benutzerschnittstelle entwickelt werden und damit die Zusammenarbeit mit dem Kunden intensiviert und dessen Vertrauen in die korrekte Umsetzung seiner Anforderungen durch konkrete Ergebnisse untermauert werden.

6.2.6.1 Erzeugung des plattformunabhängigen Strukturmodells

Hat das Workflow-Modell „Prüfungsanmeldung“ in der Modellierungsphase eine ausreichende Stabilität erreicht, so kann der Geschäftsprozessanalyst in MediniQVT Transformation T2 auf das Workflow-Modell anwenden. Als Ergebnis der Transformation wird ein benutzerzentrisches Domänenmodell für die beiden Benutzeraktionen `Register` und `Choose Examination Date` erzeugt, das an den Benutzerschnittstellenexperten übergeben und im Anschluss in mehreren Gesprächen mit dem Kunden verfeinert und um die einzelnen Eigenschaften der Benutzereingabe- und Benutzerausgabeschnittstellen erweitert werden muss. In diesem Teilabschnitt der Entwicklungsphase können der Geschäftsprozessanalyst, der Benutzerschnittstellenexperte und der Kunde anhand des Workflow-Modells die Details der einzelnen Benutzeraktionen erörtern.

Im vorliegenden Fall wünscht der Kunde, dass der Studierende bei seiner Benutzeraktion „Anmelden“ (`Register`) seinen Vornamen (`givenName`), seinen Namen (`lastName`) und einen bevorzugten Prüfungstermin (`examDate`) angeben kann. Der Hochschullehrer soll bei seiner Benutzeraktion „Prüfungstermin wählen“ (`Choose Examination Date`) die durch den Studierenden angegebenen Daten einsehen können und zusätzlich den bevorzugten Prüfungstermin des Studierenden bearbeiten sowie um eine konkrete Uhrzeit für den Beginn der Prüfung (`examStartTime`) erweitern können. Abbildung 86 zeigt im oberen Teil das durch Transformation T2 aus dem Workflow-Modell automatisiert erzeugte benutzerzentrische Domänenmodell und im unteren Teil das Resultat der aufgrund der Diskussion mit dem Kunden vom Benutzerschnittstellenexperten manuell durchgeführten Verfeinerung. Wie zu erkennen ist, hat Transformation T2 aus dem Workflow-Modell die Information übernommen, dass in beiden Benutzeraktionen jeweils das Geschäftsobjekt `Registration` bearbeitet wird und entsprechend den im Workflow-Modell spezifizierten Eingabe- und Ausgabepins (vgl. Abbildung 82, Seite 167) leere Benutzereingabe- und Benutzerausgabeschnittstellen erzeugt. Dementsprechend wird aus der Benutzeraktion `Register` heraus im Gegensatz zur Benutzeraktion `Choose Examination Date` keine Ausgabeschnittstelle abgeleitet, da der Studierende seine Daten nur eingeben muss, jedoch keine Daten angezeigt bekommt.

Sobald die erste Verfeinerung eines benutzerzentrischen Domänenmodells durch den Benutzerschnittstellenexperten unter Zuhilfenahme des **Domänenprofils** abgeschlossen ist, stehen für eine Erzeugung der Benutzerschnittstelle alle benötigten Informationen zur Verfügung. Der Benutzerschnittstellenexperte kann nun nacheinander die Transformationen T3, T6 und T7 ausführen, ohne dass eine weitere Verfeinerung der erzeugten Zwischenmodelle notwendig wäre. Auf diese Weise kann der Benutzerschnittstellenexperte dem Kunden bereits in einer sehr frühen Iteration der geschäftsgetriebenen Entwicklung eine funktionale Benutzerschnittstelle präsentieren und anhand dieser erste Anpassungswünsche des Kunden erfassen. Da die Benutzerschnittstelle den zentralen Berührungspunkt des

Benutzers zur IT-Unterstützung darstellt, ist zu erwarten, dass durch eine erste prototypische Version der Benutzerschnittstelle viele Anforderungen des Kunden konkretisiert und gegebenenfalls verfeinert werden können. Erkennt der Kunde anhand der erzeugten Benutzerschnittstelle beispielsweise nachträglich, dass der Studierende aus Gründen der Verifikation seine Matrikelnummer bei der Anmeldung angeben sollte, so muss der Benutzerschnittstellenexperte hierzu lediglich die in Abbildung 86 dargestellte Benutzereingabeschnittstelle `Registration` (`Register`) um eine zusätzliche Eigenschaft erweitern und die T2 erneut ausführen. Diese Verfeinerung kann iterativ so lange wiederholt werden, bis der Kunde alle benötigten Ein- und Ausgabeparameter auf der Benutzerschnittstelle vorfindet.

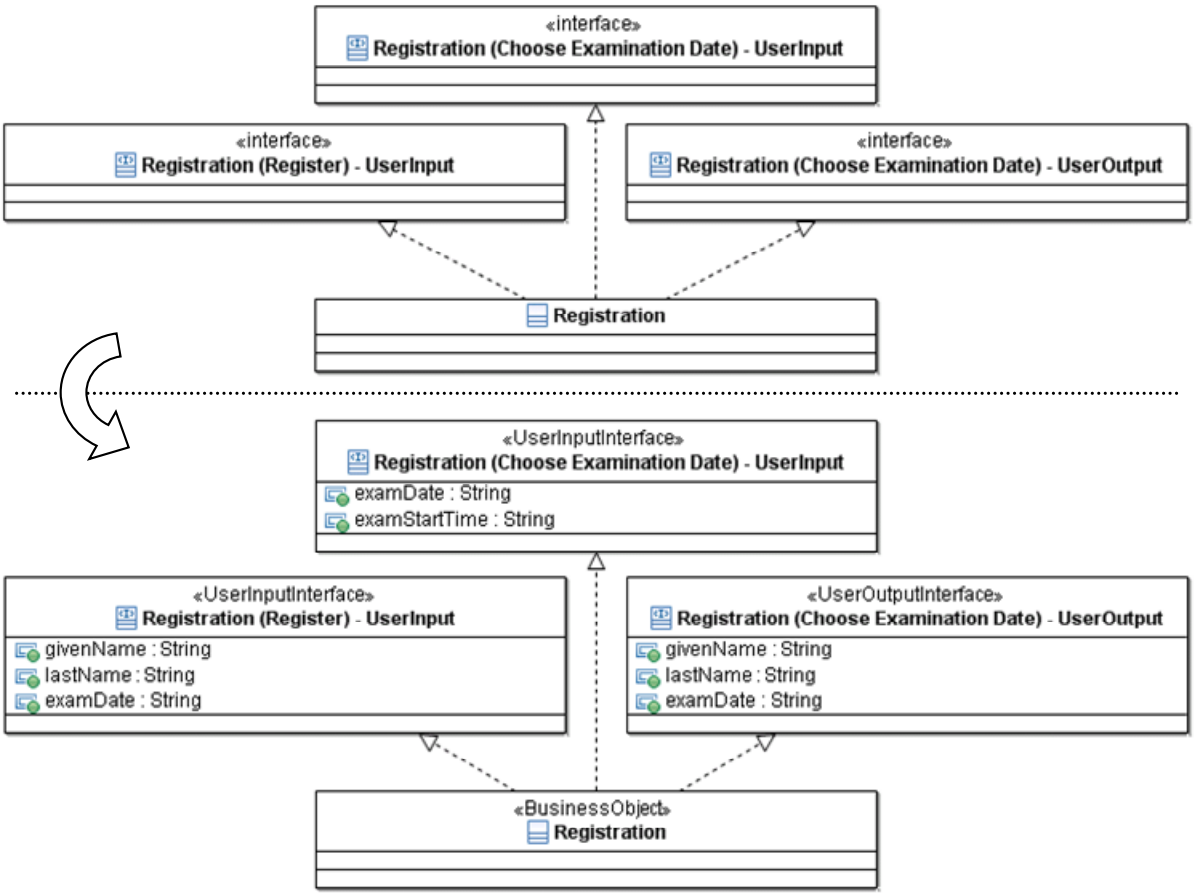

Abbildung 86: Verfeinerung des benutzerzentrischen Domänenmodells

Im Anschluss an die Spezifikation des Inhalts kann die Struktur der Benutzerschnittstelle auf Modellebene verfeinert werden. Hierzu führt der Benutzerschnittstellenexperte auf dem fertiggestellten benutzerzentrischen Domänenmodell die Modell-zu-Modell-Transformation T3 aus und nutzt das erzeugte plattformunabhängige Strukturmodell zur Verfeinerung der Struktur der Benutzerschnittstelle in RSA. Abbildung 87 zeigt am Beispiel des plattformunabhängigen Strukturmodells `Choose Examination Date` das Ergebnis der Anwendung von T3.

Das dargestellte Strukturmodell wurde durch den Benutzerschnittstellenexperten bereits hinsichtlich einer besseren Aufteilung manuell nachbearbeitet und das Containerelement zur Auswahl des Prüfungstermins (`[Container]Registration:examDate`) aus dem Containerelement `Registration` nach rechts als eigenständiges Element ausgelagert. Zusätzlich wünscht der Kunde, den Prüfungstermin über ein für ihn möglichst intuitives Benutzerschnittstellenelement auswählen zu können. Daher wurde im rechten Containerelement das zuvor vorhandene Eingabeelement (`InputElement`) für `examDate` vom Benutzerschnittstellenexperten durch ein Auswahlelement

(`SelectElement`) ersetzt. Der Benutzerschnittstellenexperte muss dementsprechend mit den Modellelementen des **Strukturprofils**, die in RSA zur Spezifikation eines Strukturmodells zur Verfügung stehen, hinreichend vertraut sein.

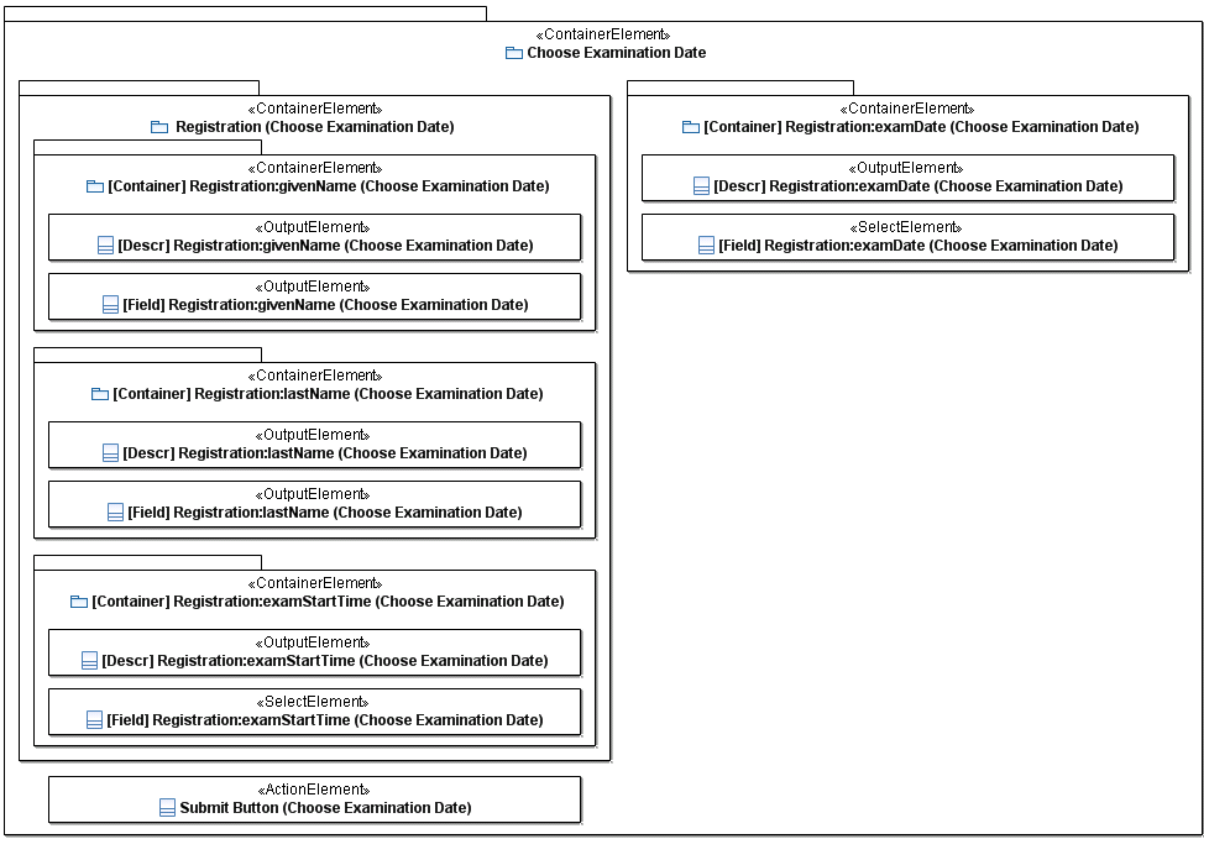

Abbildung 87: Plattformunabhängiges Strukturmodell Choose Examination Date

Um dem Kunden die Auswirkung der Verfeinerung der Struktur der Benutzerschnittstelle präsentieren zu können, werden im Anschluss der Bearbeitung des Strukturmodells erneut die Transformationen T6 und T7 zur Erzeugung einer funktionalen Benutzerschnittstelle ausgeführt. Ist der Kunde mit dem Ergebnis nicht zufrieden, kann der Benutzerschnittstellenexperte das Strukturmodell iterativ so lange verändern, bis die Anforderungen des Kunden erfüllt sind. Ergeben sich bei der Bearbeitung des Strukturmodells weitere Änderungswünsche des Kunden am Inhalt der Benutzerschnittstelle, so kann der Benutzerschnittstellenexperte ebenfalls das benutzerzentrische Domänenmodell erneut aufgreifen und die Änderungswünsche des Kunden einarbeiten, da beide Modelle in einem eigenständigen Entwicklungszweig bearbeitet werden und daher die weiteren Entwicklungsaktivitäten nicht beeinflussen. Im Strukturmodell bereits vorgenommene Veränderungen bleiben dank der vorhandenen Werkzeugunterstützung auch bei einer nachträglichen Anpassung des benutzerzentrischen Domänenmodells erhalten.

Ein zentraler Vorteil des erzeugten Strukturmodells liegt in dessen Plattformunabhängigkeit. Obwohl das Strukturmodell für eine erste Demonstration des Entwurfs einer Benutzerschnittstelle durch T6 und T7 auf eine konkrete Plattform abgebildet werden muss, kann die Benutzerschnittstelle durch eine Anpassung von T6 und T7 oder durch die Spezifikation weiterer Transformationen T6’ und T7’ für beliebige Plattformen ausgeprägt werden. Die Anpassungen der Transformationen fällt dabei nicht in das Aufgabengebiet der Anwendungsentwickler, sondern muss von entsprechenden MDSD-Experten übernommen werden.

Nachdem die wesentlichen Anforderungen des Kunden im Hinblick auf Struktur und Inhalt der Benutzerschnittstellen erfasst wurden, kann das Entwicklungsvorgehen aufgrund der in den folgenden Entwicklungsschritten behandelten technischen Spezifika der Zielplattformen ohne den Kunden weiterverfolgt werden. Eine mögliche Zielplattform für Benutzerschnittstellen stellt der im Kontext des KIM-Projekts eingesetzte Teil ASP.NET des .NET-Rahmenwerks dar, durch den unter anderem als *WebParts* bezeichnete Präsentationskomponenten für ein Portal einer dienstorientierten Architektur spezifiziert werden können. Um die Benutzerschnittstellen des Workflows „Prüfungsanmeldung“ für ASP.NET implementieren zu können, zeigt der folgende Abschnitt die notwendigen Schritte zur Erzeugung eines plattformspezifischen Strukturmodells auf, das im weiteren Verlauf als ***WebPart*-Modell** bezeichnet wird.

6.2.6.2 Erzeugung des WebPart-Modells

Die Benutzerschnittstelle einer dienstorientierten Architektur wird im Kontext dieser Arbeit durch ein Portal realisiert. Einzelne, als *Portlets* bezeichnete Präsentationskomponenten, die in Portalen zu Portalseiten aggregiert werden, können im Wesentlichen gegen zwei Spezifikationen entwickelt werden. Auf der einen Seite steht hierzu die Java-*Portlet*-Spezifikation in Version 1.0 [JCP-JSR168] und Version 2.0 [JCP-JSR286] zur Verfügung, auf der anderen Seite existiert die durch Microsoft vorangetriebene Spezifikation von ASP.NET (*Active Server Pages* .NET), welche die Entwicklung von Webanwendungen auf der Basis des .NET-Rahmenwerks ermöglicht. In ASP.NET findet sich das Konzept der *WebParts*, das die Erstellung von personalisierbaren Portalseiten ermöglicht[6]. Beide Spezifikationen haben für die Anforderungen dieser Arbeit einen ausreichend mächtigen Umfang zur Erstellung von *Portlets* als Grundlage der Spezifikation formularbasierter Benutzerschnittstellen für ein Portal. Da der durch Microsoft vertriebene Office SharePoint Server und das durch diesen implementierte .NET-*Framework* im Kontext des KIM-Projekts eingesetzt werden, wird im Folgenden die Abbildung des plattformunabhängigen Strukturmodells auf ASP.NET und die dort spezifizierten *WebParts* demonstriert. Ferner bietet der Office SharePoint Server über eine Erweiterung die Möglichkeit, *WebParts* als WSRP-*Portlets* gemäß dem WSRP-Standard in Betrieb zu nehmen. Durch diese Möglichkeit der Bereitstellung eines *WebParts* als präsentationsorientierter Webservice bleibt dem späteren Dienstnehmer der interne Aufbau der diensterbringenden Komponente verborgen. Für ihn ist somit transparent, ob das *Portlet* gegen die ASP.NET- oder Java-*Portlet*-Spezifikation entwickelt wurde. Die damit realisierte Plattformunabhängigkeit führt zu einer guten Portabilität der *Portlets*, die in unterschiedlichen Anwendungsszenarien zum Einsatz kommen können.

Um ein *WebPart* entsprechend der ASP.NET-Spezifikation durch ein Modell spezifizieren zu können, müssen die Konzepte der Spezifikation auf ein geeignetes Metamodell umgesetzt und die Modellelemente den Anwendungsentwicklern zur Verfügung gestellt werden. Anhang A.3 zeigt daher alle notwendigen Schritte und Entwurfsentscheidungen zur Erzeugung eines ***WebPart*-Profils** auf, dessen Modellelemente einer als *WebPart*-Experte bezeichneten Entwicklungsrolle in RSA zur Modellierung eines *WebPart*-Modells zur Verfügung gestellt werden. Die Anwendung der vertikalen Modell-zu-Modell-Transformation T6 auf ein plattformunabhängiges Strukturmodell erzeugt, wie Abbildung 88 zeigt, ein gleichnamiges *WebPart*-Modell, das die grundlegende Struktur einer Benutzerschnittstelle übernimmt, dabei jedoch um plattformspezifische Details der Zielplattform ASP.NET wie etwa der Spezifikation eines `Panels` für jedes Containerelement spezialisiert wird.

---

[6] WebPart ist die Microsoft-spezifische Bezeichnung für eine Präsentationskomponente für den Office SharePoint Server. Eine umfassende Einführung in diese Technologie ist beispielsweise in [Bo07] zu finden.

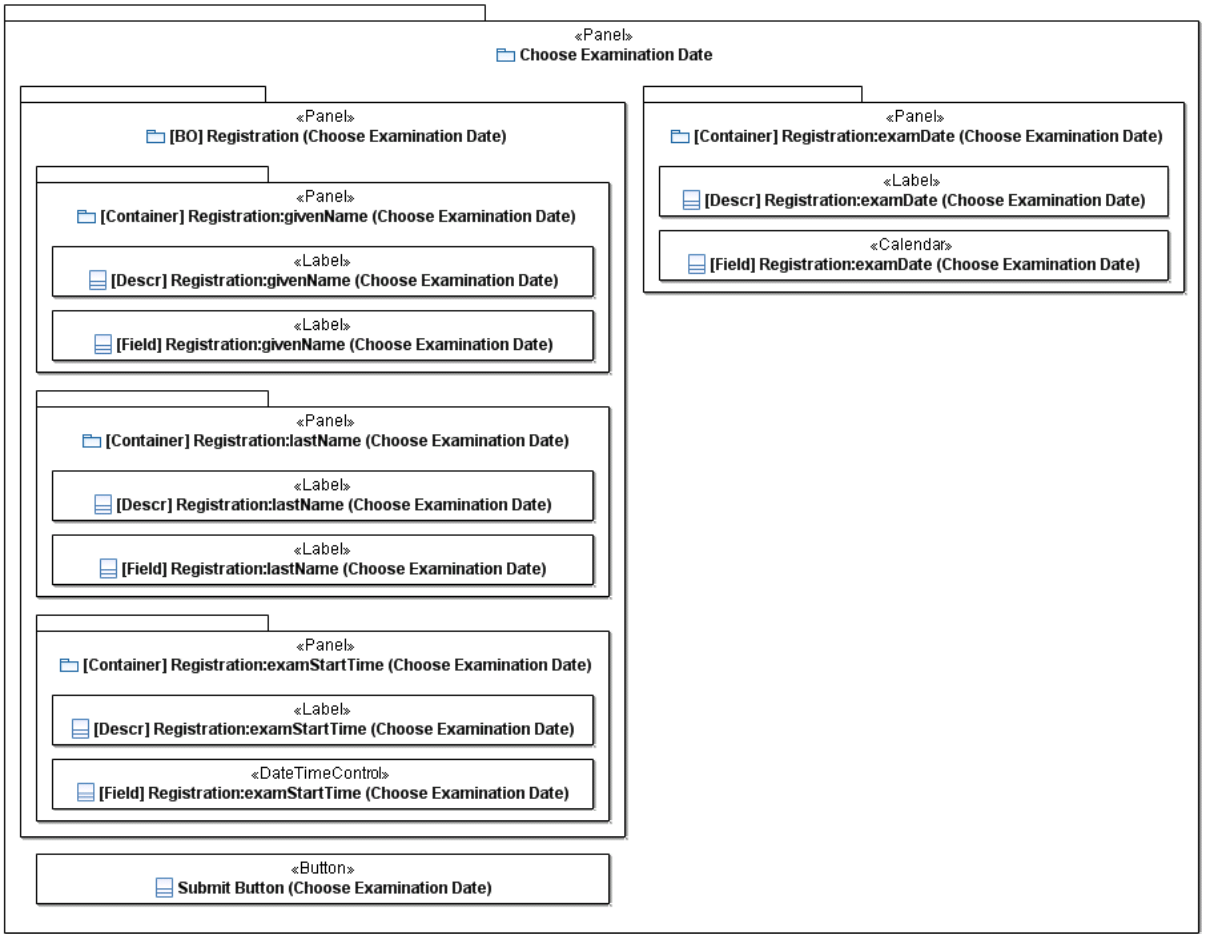

Abbildung 88: WebPart-Modell Choose Examination Date

Das erzeugte *WebPart*-Modell steht im Anschluss für eine erneute Verfeinerung oder Erweiterung durch einen *WebPart*-Experten zur Verfügung, der somit eine weitere strukturelle Ausgestaltung der Benutzerschnittstellen direkt am Modell vornehmen kann. Er muss beispielsweise entscheiden, ob das Kalenderelement, das zur Auswahl des Prüfungstermins automatisch durch Transformation T6 erzeugt wurde, den Anforderungen des Kunden entspricht oder ob ein anderes Modellelement des *WebPart*-Profils geeigneter ist.

Hat der *WebPart*-Experte die Verfeinerung der *WebPart*-Modelle abgeschlossen, so müssen diese in einem letzten Entwicklungsschritt in der Entwicklungsphase durch Transformation T7 auf die plattformspezifische Implementierung für ASP.NET abgebildet und im Anschluss die erzeugten Softwareartefakte in Betrieb genommen werden.

6.2.6.3 Erzeugung der Benutzerschnittstellenspezifikation

Obwohl im Kontext der modellgetriebenen Softwareentwicklung die Vision einer durchgängigen Spezifikation aller Softwareartefakte durch Modelle existiert, erfolgt deren Implementierung bisher noch durch höhere Programmiersprachen wie beispielsweise Java oder C#. Für diese Programmiersprachen liegen Metamodelle im Sinne der modellgetriebenen Entwicklung nicht vor, weshalb in einem letzten Transformationsschritt die plattformspezifischen Modelle durch eine Modell-zu-Text-Transformation auf Quellcode einer Programmiersprache überführt werden müssen. Gleiches gilt somit auch für das *WebPart*-Modell, das auf die konkrete textuelle Syntax der Programmiersprache C# abgebildet werden muss.

Durch das nicht vorhandene Metamodell der Zielplattform C# kann das bisher verwendete Metamodell-basierte Transformationsvorgehen nicht weiter angewendet werden. Derzeitig verfügbare Transformationsausführungsumgebungen für Modell-zu-Text-Transformationen nutzen zur Umsetzung dieser Transformationen in der Regel einen besucherbasierten (engl. *Visitor-based*) oder vorlagenbasierten (engl. *Template-based*) Ansatz [CH06]. Ein Beispiel für einen vorlagenbasierten Ansatz liefert die Entwicklungsumgebung *Open Architecture Ware* (OAW) [OAW], welche die vorlagenbasierte Transformationssprache *Xpand* [BG+07] nutzt. Die benötigten Vorlagen enthalten dabei statischen Programmcode, der um sogenannte *Metacode*-Fragmente entsprechend der Sprachreferenz von *Xpand* erweitert wird. Bei der Ausführung der Transformation werden durch die *Metacode*-Fragmente Teile des Quellmodells selektiert und die Vorlage iterativ um den dynamisch erzeugten Programmcode (etwa C#-Quellcode) ergänzt [Cl88]. Der nachfolgend gezeigte Ausschnitt einer Vorlagen-Spezifikation in *Xpand* definiert ein Attribut `CheckBoxList` für ein *WebPart*-Modell und setzt hier den aus dem Quellmodell gewonnenen Bezeichner ein.

```
«DEFINE AttributeDeclaration FOR WebPartModel::CheckBoxList»
  CheckBoxList «this.name.getValidVarName()»;
«ENDDEFINE»
```

Zur Spezifikation der Transformation T7 nutzt diese Arbeit die durch OAW implementierte Transformationssprache *Xpand*, weshalb entsprechende Vorlagen entwickelt werden müssen, die das *WebPart*-Modell als Quelle nutzen. Bei der Ausführung der Transformation werden diese Vorlagen dann schrittweise mit den aus dem Quellmodell gewonnenen Werten gefüllt, sodass nach der Ausführung der Quellcode der *WebParts* in C# als Ergebnis von T7 vorliegt. Dementsprechend werden qualifizierte Fachentwickler benötigt, die vor der Ausführung des geschäftsgetriebenen Entwicklungsvorgehens die notwendigen Vorlagen entwickeln und den *WebPart*-Experten zur Ausführung der Transformation zur Verfügung stellen. Anhang A.4 geht auf diese vorbereitenden Maßnahmen im Detail ein und zeigt die Spezifikation der für Transformation T7 benötigten Vorlagen.

Stehen die benötigten Transformationsvorlagen für den *WebPart*-Experten in OAW bereit, so kann dieser Transformation T7 auf den *WebPart*-Modellen des Workflows „Prüfungsanmeldung“ ausführen und dadurch Quellcode für die beiden Benutzerschnittstellen `Register` und `Choose Examination Date` jeweils in Form eines *WebParts* generieren. Sofern eine weitere Verfeinerung des erzeugten Quellcodes notwendig ist, dient dem *WebPart*-Experten das Produkt Visual Studio von Microsoft als Entwicklungsumgebung, das im KIM-Projekt ebenfalls durchgängig zur Entwicklung von Lösungen für den Office SharePoint Server eingesetzt wird. Sind abschließend alle Arbeiten an den *WebParts* abgeschlossen, dann kann die Entwicklungsphase für die Benutzerschnittstellen beendet und die erzeugten Softwareartefakte der Inbetriebnahmephase übergeben werden.

Um die generierten *WebParts* `Choose Examination Date` und `Register` in den Betrieb zu überführen, müssen beide durch einen Administrator auf einem Office SharePoint Server installiert und für die Integration in eine Portalseite freigegeben werden. Dabei können die erzeugten *WebParts* auch als präsentationsorientierte Dienste für verschiedene Anwendungsszenarien nutzbar gemacht werden, indem sie über eine als WSRP-*Toolkit* [Microsoft-WSRP] bezeichnete Erweiterung des Office SharePoint Servers als WSRP-*Portlet* bereitgestellt werden und damit von verschiedenen, entfernten WSRP-Konsumenten verwendet werden können. Auf diese Weise ist es möglich, den durch ein *WebPart* generierten Inhalt auf einem entfernten Portal-Server wie beispielsweise den durch IBM entwickelten WebSphere Portal Server [IBM-WPoS] einzubinden, der die Ausführung von *WebParts*

per se nicht unterstützt. Eine detaillierte Beschreibung der Inbetriebnahme eines *WebParts* als WSRP-*Portlet* ist in [Ry09] zu finden.

Abbildung 89 zeigt schließlich als ein Ergebnis des geschäftsgetriebenen Entwicklungsvorgehens das *WebPart* zur Benutzeraktion `Choose Examinaton Date` als Bestandteil des KIT-Studierendenportals. Hierbei finden sich die im benutzerzentrischen Domänenmodell spezifizierten Eigenschaften der Benutzerschnittstelle, wie etwa der Name und Vorname des Studierenden (`Given Name`, `Last Name`) und das Kalenderelement zur Auswahl des Prüfungstermins, aber auch die im Strukturmodell erfassten Spezifikationen, beispielsweise die Positionierung des Kalenderelements im rechten Teil der Benutzerschnittstelle, wieder.

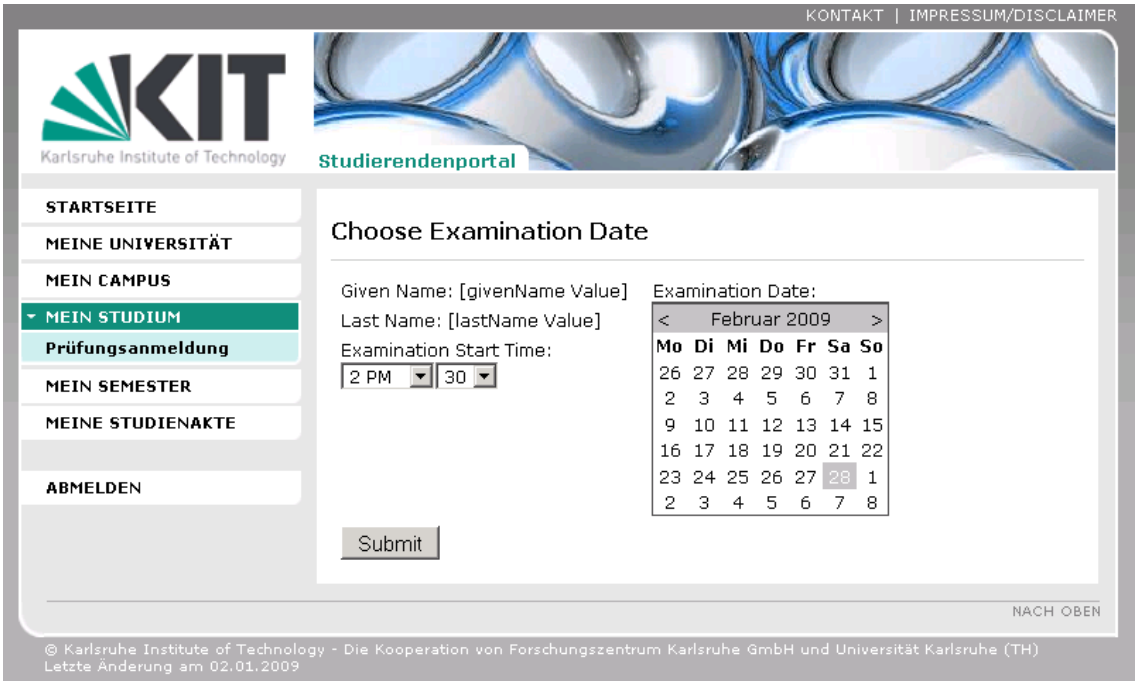

Abbildung 89: Benutzerschnittstelle zu Choose Examination Date im Office SharePoint Server

Das erzeugte *WebPart* muss in einem weiteren Entwicklungsschritt durch einen weiteren Experten hinsichtlich seines Layouts beispielsweise durch den Einsatz von *Cascading Style Sheets* [W3C-CSS2.1] überarbeitet und optimiert werden, um den Benutzern des KIT-Studierendenportals eine optisch und ergonomisch ansprechende Benutzerschnittstelle präsentieren zu können. Diese Überarbeitung ist jedoch nicht mehr Bestandteil dieser Arbeit (siehe Prämisse P2).

Wie angeführt können die erzeugten *WebParts* über den WSRP-*Provider* des Office SharePoint Servers als präsentationsorientierte Dienste angeboten werden. Wird dabei die Integration aller Dienstbeschreibungen der präsentationsorientierten Dienste in ein gemeinsames, KIT-weites Dienstverzeichnis verfolgt, so kann das zuvor aufgezeigte Entwicklungsvorgehen angepasst werden. Sobald die Anforderungen eines Kunden im Hinblick auf die benötigten Ein- und Ausgabeparameter einer Benutzerschnittstelle erfasst wurden, kann der Benutzerschnittstellenexperte zunächst das Dienstverzeichnis des KIT durchsuchen, ob in diesem bereits ein präsentationsorientierter Dienst verzeichnet ist, der den geforderten Ansprüchen genügt. Falls ein *WSRP*-Provider einen entsprechenden Dienst bereits anbietet, kann die Entwicklung der Benutzerschnittstellenspezifikation auf die Integration des geeigneten WSRP-*Portlet* in das Zielportal reduziert werden. Für eine umfassende Behandlung der in diesem Zusammenhang auftretenden Fragestellungen, beispielsweise nach einer hinreichenden semantischen

Beschreibung der Dienste oder nach geeigneten Verfahren zur Suche von Diensten, sei auf [Kr07] verwiesen.

Neben den erzeugten Benutzeraufgaben- und Benutzerschnittstellenspezifikationen muss für eine vollständige Umsetzung des Workflows „Prüfungsanmeldung“ auch die Abbildung des Kontrollflusses des Workflows auf eine Prozessausführungssprache verfolgt werden. Wird hierbei das Workflow-Modell entsprechend eines der in unterschiedlichen Forschungsansätzen wie etwa [EW+06] untersuchten Vorgehen auf eine Prozessausführungssprache abgebildet, so stehen alle Softwareartefakte zur Ausführung des Workflows „Prüfungsanmeldung“ inklusive der darin enthaltenen Benutzerinteraktionen zur Verfügung. Werden ferner, wie in Abschnitt 1.2 erläutert, alle weiteren benötigten fachfunktionalen Dienste zur Abbildung des Workflows als gegeben angenommen, so kann der Workflow durch die vorhandenen IT-Systeme ausgeführt werden. Das in dieser Arbeit zur Abbildung des Workflows verfolgte geschäftsgetriebene Entwicklungsvorgehen ist dabei nur ein mögliches Vorgehensmodell, in dem die Konzepte dieser Arbeit angewendet werden können. Da die modellgetriebene Architektur und die in deren Kontext verwendeten Modelle und Transformationen an sich an keinen Entwicklungsprozess gebunden sind, können unterschiedliche Entwicklungsvorgehen als Grundlage der Entwicklung und damit einhergehend auch unterschiedliche Rollen verwendet werden. Einzige Voraussetzung der direkten Anwendbarkeit der Konzepte dieser Arbeit ist die Ausrichtung des Entwicklungsvorgehens an einem Geschäftsprozess als Ausgangsmodell. Abschnitt 7.3.4 gibt einen Ausblick auf mögliche Anpassungen zur Aufhebung dieser Voraussetzung.

Um die Anwendbarkeit der in dieser Arbeit entwickelten Konzepte auch im Umfeld der Industrie erproben zu können, wurde die Umsetzung der Konzepte im Rahmen einer Industriekooperation angestrebt. Die Ergebnisse dieser Kooperation werden im folgenden Abschnitt diskutiert.

6.3 Umsetzung der Konzepte in der Industrie

Ein weiterer Tragfähigkeitsnachweis der Konzepte dieser Arbeit wurde im Rahmen einer Industriekooperation mit einem mittelständischen Unternehmen erbracht, das sich auf die Entwicklung von *Enterprise Resource Planning* (ERP)-Systemen spezialisiert hat und seine ERP-Lösungen an verschiedene mittelständische Unternehmen ausliefert. Da viele der über Jahre gewachsenen ERP-Systeme unterschiedlichster Hersteller zum Großteil noch funktions- und datenorientiert sind, können die für ERP-Systeme typischen Geschäftsprozesse, wie beispielsweise der Verkauf von Lagerteilen, nicht direkt auf diese ERP-Systeme umgesetzt werden. Daher muss der Ablauf dieser Geschäftsprozesse den Sachbearbeitern im Voraus bekannt sein und von diesen eigenständig durch die Verwendung der benötigten Anwendungen oder Funktionen ausgeführt werden. Diese implizite Steuerung der Geschäftsprozesse über verschiedene Benutzerschnittstellen und Transaktionen hinweg steht einer schnellen und flexiblen IT-Unterstützung im Wege. Die Herausforderung in der mit dem ERP-Unternehmen eingegangenen Kooperation bestand daher in der Erweiterung des bestehenden ERP-Systems des Unternehmens um zusätzliche fachfunktionale Komponenten, welche die Spezifikation und die Ausführung von Geschäftsprozessen mit Benutzerinteraktion ermöglichen sollten. Ziel war es, die Architektur und Funktionalität des ERP-Systems dahin gehend zu erweitern, dass die Erfassung von Geschäftsprozessen durch Modelle und die Umsetzung dieser Modelle durch eine vollautomatisierte Abbildung auf die fachfunktionalen Komponenten des ERP-Systems ermöglicht wird.

Um die Ausführung von Geschäftsprozessen im ERP-System unterstützen zu können, wurde zunächst eine Möglichkeit zur expliziten Spezifikation der Geschäftsprozesse innerhalb des ERP-Systems benötigt. Hierbei bestand eine Anforderung des Auftraggebers darin, alle zusätzlich benötigten

fachfunktionalen Komponenten vollständig in das zu entwickelnde ERP-System zu integrieren und keine Anwendungen von kommerziellen Drittanbietern zu nutzen. Wie der Autor in [LJ+08] publiziert hat, wurde daher das als Open-Source-Implementierung zur Verfügung stehende *Eclipse Modeling Framework* zur Modellierung der Geschäftsprozesse gewählt und in das ERP-System integriert. Als Erweiterung zu EMF existiert das *Graphical Modeling Framework* (GMF) [EF-GMF], mithilfe dessen voll funktionsfähige grafische Editoren für eigenständig definierte Metamodelle generiert werden können. Aus diesem Grund konnte GMF im vorliegenden Fall dazu genutzt werden, eine auf die Domäne des ERP-Systems exakt zugeschnittene domänenspezifische Sprache zu entwerfen. Da in ERP-Systemen die Unterscheidung, ob eine Arbeitseinheit in einem Geschäftsprozess durch einen Menschen bzw. ein IT-System alleine oder durch eine Interaktion zwischen Mensch und IT-System ausgeführt werden muss, von zentraler Bedeutung ist, wurde als erster Teil der domänenspezifischen Sprache das in Abschnitt 4.2 eingeführte **Benutzeraktionsprofil** durch ein *ECore*-Metamodell umgesetzt und damit eine detaillierte Spezifikation der einzelnen Arbeitseinheiten der Geschäftsprozesse ermöglicht. Abbildung 90 zeigt einen Ausschnitt eines repräsentativen Geschäftsprozesses „Lagerteile verkaufen“, in dem die einzelnen Arbeitseinheiten bereits durch die konkreten Stereotypen der Systemaktion (Zahnrad-Symbol), der Benutzeraktion (Monitor-Symbol) und der manuellen Aktion (Figur-Symbol) aus dem Benutzeraufgabenprofil ausgezeichnet wurden.

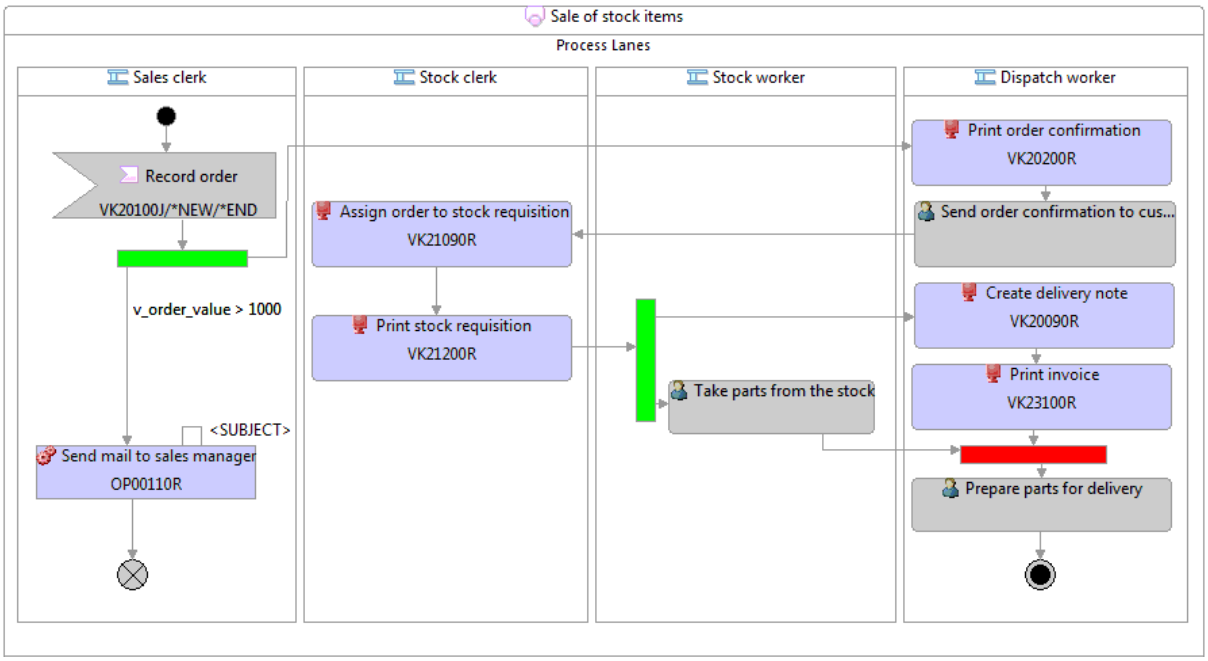

Abbildung 90: Plattformspezifisches Geschäftsprozessmodell im ERP-System

Den Prinzipien der modellgetriebenen Architektur folgend, sollten in dem auf Basis des GMF entwickelten Geschäftsprozesseditor gleichsam dem in dieser Arbeit verfolgten Vorgehen zunächst plattformunabhängige Geschäftsprozesse modelliert werden können. Der Vorteil einer plattformunabhängigen Spezifikation der Geschäftsprozesse liegt zum einen in der Möglichkeit, verschiedene für ERP-Systeme typische Geschäftsprozesse bereits mit dem Produkt als Referenzmodelle auszuliefern. Dadurch muss der Käufer des ERP-Systems diese Referenzmodelle gegebenenfalls nur minimal an sein Anwendungsszenario adaptieren und kann dann im Anschluss an die Anpassung den Referenzprozess direkt auf die Laufzeitumgebung des ERP-Systems umsetzen. Zum anderen liegt der Vorteil der plattformunabhängigen Betrachtung in der möglich werdenden Berücksichtigung der verschiedenen fachfunktionalen Komponenten, die das ERP-System zur Verfügung stellt. Für eine Benutzeraktion wie beispielsweise „Kunde anlegen“ existieren unter Umständen verschiedene fachfunktionale

Komponenten, welche die gewünschte Funktionalität erbringen können. Wird die Wahl der zu verwendenden Komponente auf die plattformspezifische Ebene verlagert, so behält der allgemein spezifizierte Geschäftsprozess unabhängig von seiner Umsetzung Gültigkeit und kann jederzeit angepasst werden.

Dementsprechend wird im nächsten Schritt der Entwicklung eines Geschäftsprozesses im ERP-System eine Modell-zu-Modell-Transformation des plattformunabhängigen Modells auf das plattformspezifische Modell eines Geschäftsprozesses ausgeführt. Im erzeugten plattformspezifischen Modell kann dann ein Berater des ERP-Unternehmens (ein Domänenexperte) dem Kunden bei der Auswahl der geeigneten Funktionsbausteine zur Unterstützung der einzelnen Arbeitseinheiten behilflich sein. Abbildung 90 zeigt einen Ausschnitt eines bereits verfeinerten plattformspezifischen Geschäftsprozesses „Lagerteile verkaufen“, in dem die in den Aktionen dargestellten Akronyme wie etwa „VK21090R“ den Bezeichnungen der anvisierten Funktionsbausteine entsprechen. Hierbei steht im Geschäftsprozesseditor durch das Benutzeraktionsprofil eine an die UML angelehnte konkrete Syntax zur Verfügung, die durch eine möglichst klare und strukturierte Darstellung sowie aussagekräftige Modellelemente den Kunden und Berater gleichermaßen bei der Spezifikation der Geschäftsprozesse unterstützen soll.

Wurde ein plattformspezifischer Geschäftsprozess durch den Kunden und den Berater hinreichend verfeinert, so wird dieser Geschäftsprozess durch eine ebenfalls im ERP-System implementierte Modell-zu-Text-Transformation auf die Prozessausführungssprache des ERP-Systems abgebildet. Da aus den spezifizierten Benutzeraktionen auch in diesem Anwendungsszenario verschiedene Benutzeraufgaben resultieren, wurden die Konzepte des **Benutzeraufgabenprofils** und der zur Überwachung von Benutzeraufgaben notwendigen **Benutzeraufgabenverwaltung** ebenfalls in das ERP-System integriert.

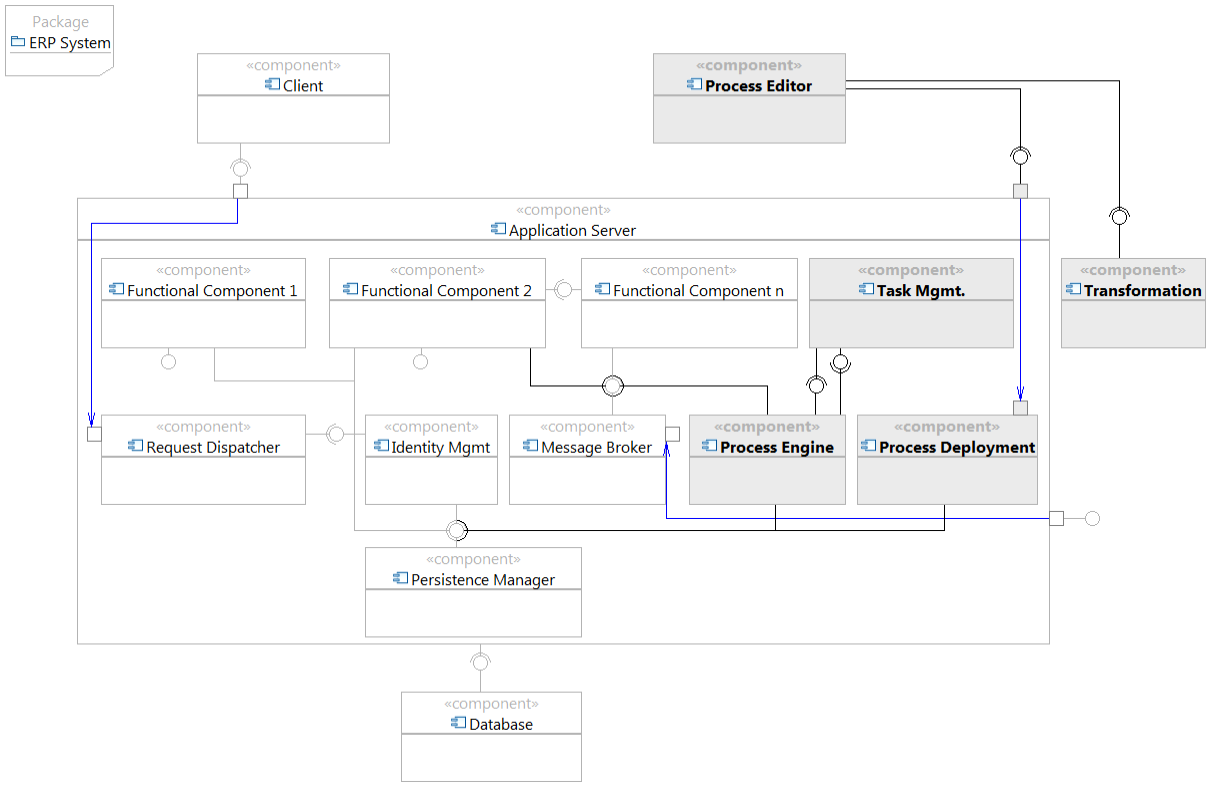

Abbildung 91: Architektur des erweiterten ERP-Systems

Aufgrund der Anforderung des ERP-Unternehmens, möglichst auf die Integration von Implementierungen Dritter in das eigene ERP-System zu verzichten, wurde die Architektur des ERP-Systems um mehrere eigenständig implementierte fachfunktionale Komponenten wie beispielsweise einen Geschäftsprozesseditor, eine Prozessausführungsumgebung, aber auch eine Transformationsausführungsumgebung erweitert. Abbildung 91 gibt einen Überblick über die Architektur des ERP-Systems, in der sich ebenfalls die in dieser Arbeit vorgeschlagenen Erweiterungen einer Benutzeraufgabenverwaltung (`Task Management`) oder einer Identitätsverwaltung (`Identity Management`) wiederfinden. Den Zugriff des Benutzers auf seine aktuellen Benutzeraufgaben wird durch die Komponente `Client` ermöglicht, die für jeden Benutzer eine eigene ***Tasklist*** mit den offenen Benutzeraufgaben vorhält. Für eine vollständige Beschreibung der verwendeten fachfunktionalen Komponenten, deren Kommunikationsbeziehungen und der verwendeten Ausführungssprachen sei auf [Ja08] verwiesen.

Die im Geschäftsprozesseditor spezifizierten Geschäftsprozesse können durch die in der Transformationsausführungsumgebung (`Transformation`) hinterlegten Transformationen auf Softwareartefakte für das ERP-System abgebildet werden. In dieser wurden unter anderem die in Abschnitt 5.2 konzipierten Transformationen umgesetzt und damit eine Erzeugung von Benutzeraufgabenspezifikationen aus Benutzeraktionen ermöglicht. Eine durchgängige Unterstützung von Geschäftsprozessen mit Benutzerinteraktion wurde durch die Benutzeraufgabenspezifikationen und deren Ausführung in der Benutzeraufgabenverwaltung somit ermöglicht.

Das erfolgreich erweiterte ERP-System befindet sich heute als eigenständiges Produkt im Einsatz, mit dessen Hilfe die Anwender des ERP-Systems ihre Geschäftsprozesse entlang eines modellgetriebenen Entwicklungsvorgehens in einem GMF-basierten Geschäftsprozesseditor erfassen und vollautomatisch auf die vorhandenen Funktionsbausteine des ERP-Systems abbilden können. Ein manueller Eingriff in die Implementierung der Geschäftsprozesse ist nicht notwendig. Auch in diesem Anwendungsszenario konnte somit gezeigt werden, dass eine durchgängige Umsetzung von Geschäftsprozessen mit Benutzerinteraktion entlang eines modellgetriebenen Vorgehens ermöglicht werden kann. Als zentraler Mehrwert gestattet der modellgetriebene Ansatz eine flexible Anpassung der Geschäftsprozesse eines Unternehmens an die sich aufgrund des Wettbewerbs rasch ändernden Anforderungen.

6.4 Resümee

In diesem Kapitel konnte die Tragfähigkeit der entwickelten Beiträge anhand zweier konkreter Szenarien demonstriert werden. Im ersten Fall wurde ein aus dem Szenario der Forschung und Lehre entnommener Geschäftsprozess in den Mittelpunkt des Nachweises gerückt und die Umsetzung dieses Geschäftsprozesses auf eine dienstorientierte Architektur als Realisierung der IT-Unterstützung aufgezeigt. Hierbei konnte demonstriert werden, dass die konzipierten Metamodelle eine in das gesamte modellgetriebene Entwicklungsvorgehen integrierte Berücksichtigung der betrachteten Aspekte der Benutzerinteraktion ermöglichen. Dementsprechend standen die Benutzerschnittstelle als zentraler Berührungspunkt des Menschen zur IT-Unterstützung und die durch einen Menschen auszuführenden Benutzeraufgaben im Fokus des verfolgten geschäftsgetriebenen Entwicklungsvorgehens. Von einem Workflow-Modell ausgehend konnte unter Verwendung bestehender Entwicklungsumgebungen gezeigt werden, dass die konzipierten Metamodelle eine genügend große Ausdrucksmächtigkeit aufweisen, um zunächst abstrahiert von plattformspezifischen Details die Aspekte der Benutzerinteraktion durch Modelle spezifizieren zu können. So konnten aus einem plattformunabhängigen Workflow-Modell in einem ersten Transformationsschritt für die vorhandenen Benutzeraktionen je ein Benutzeraufgabenmodell und ein benutzerzentrisches Domänenmodell sowie ein Strukturmodell der Benutzerschnittstelle automatisiert erzeugt werden. Nachdem diese Modelle durch entsprechende

Fachentwickler erweitert und verfeinert wurden, dienten sie im nächsten Entwicklungsschritt als Ausgangsmodelle zur automatisierten Erzeugung plattformspezifischer Modelle der Benutzeraufgaben und Benutzerschnittstellen. Entsprechend der im konkreten Szenario vorhandenen Plattformen wurde beispielhaft eine automatisierte Erzeugung und Ausprägung der Modelle für die Plattformen ASP.NET und WS-HumanTask aufgezeigt. Als Ergebnis der beispielhaften Ausführung des geschäftsgetriebenen Entwicklungsvorgehens lagen Benutzerschnittstellen- und Benutzeraufgabenspezifikationen im Sinne ausführbarer Softwareartefakte vor, die in der aufgezeigten dienstorientierten Architektur ausgeführt und damit zur Unterstützung von Geschäftsprozessen mit Benutzerinteraktion verwendet werden können.

Anhand eines zweiten konkreten Szenarios aus der Industrie konnte gezeigt werden, dass die in dieser Arbeit entwickelten Konzepte zur Abbildung von Geschäftsprozessen mit Benutzerinteraktion auch für die in der Industrie vorzufindenden Anwendungsszenarien eine hohe Relevanz haben. Am Beispiel eines ERP-Unternehmens wurde daher demonstriert, wie die Konzepte dieser Arbeit auf bestehende IT-Systeme gewinnbringend angewendet werden können. Als Resultat lag ein erweitertes ERP-System vor, das sowohl die Spezifikation von Geschäftsprozessen mit Benutzerinteraktion durch Modelle als auch deren Transformation auf Softwareartefakte und deren Ausführung vollständig unterstützt.

7 Bewertung und Ausblick

In diesem abschließenden Kapitel werden die Beiträge der Arbeit zusammengefasst und hinsichtlich der in Abschnitt 3.1 erfassten Anforderungen bewertet. Ferner wird ein Ausblick auf mögliche weiterführende Fragestellungen gegeben.

7.1 Beiträge der Arbeit

Ziel der vorlegenden Arbeit war es, Geschäftsprozesse unter Berücksichtigung der Interaktion eines Menschen in einem modellgetriebenen Verfahren zu entwickeln und automatisiert auf Softwareartefakte einer dienstorientierten Architektur abzubilden. Um den Entwicklern hierzu geeignete Modellelemente zur Spezifikation der verschiedenen Aspekte der Benutzerinteraktion zur Verfügung stellen zu können, wurden im ersten Beitrag dieser Arbeit plattformunabhängige Metamodelle konzipiert und auf das bestehende Metamodell der UML umgesetzt. Hierdurch konnte eine Integration der Metamodelle der Benutzerinteraktion in bestehende Entwicklungsvorgehen erreicht werden. Zusätzlich abstrahieren die Metamodelle von plattformspezifischen Details und tragen damit der Heterogenität heutiger IT-Landschaften Rechnung. Den Prinzipien der modellgetriebenen Softwareentwicklung folgend standen eine automatisierte Erzeugung der Modelle der Benutzerinteraktion und deren Abbildung auf lauffähige Softwareartefakte für eine dienstorientierte Architektur durch Transformationen im Mittelpunkt des zweiten Beitrags. Zwei Tragfähigkeitsnachweise demonstrierten die Umsetzbarkeit und Anwendbarkeit der entwickelten Konzepte.

7.1.1 Metamodelle zur Spezifikation der Benutzerinteraktion

Die zur Erreichung der Geschäftsziele eines Unternehmens notwendigen Geschäftsprozesse bildeten den Ausgangspunkt des im ersten Beitrag konzipierten modellgetriebenen Entwicklungsvorgehens. Um bereits in einem abstrakten Modell eines Geschäftsprozesses die Anforderungen der Benutzerinteraktion verankern zu können, wurden durch eine Erweiterung bestehender Metamodelle zusätzliche Modellelemente für eine verfeinerte Spezifikation eines Geschäftsprozesses konzipiert und mittels einer Umsetzung auf das Metamodell der UML durch das **Benutzeraktionsprofil** zur Modellierung bereitgestellt. Von einem entsprechend verfeinerten Geschäftsprozessmodell aus konnte in einem manuellen Abbildungsschritt die Ableitung eines Workflow-Modells gezeigt werden, das dann als neuer Ausgangspunkt zur Erzeugung der folgenden plattformunabhängigen Modelle der Benutzerinteraktion diente:

Das **Benutzeraufgabenmodell** stellt den Bezug zwischen einer in der Kontrollflussperspektive eines Workflows erfassten Benutzeraktion und einer daraus entstehenden Benutzeraufgabe in der Ressourcenperspektive eines Workflows her. Um eine detaillierte Betrachtung eines Workflows aus der Ressourcenperspektive zu ermöglichen, werden durch das als **Benutzeraufgabenprofil** umgesetzte Benutzeraufgabenmetamodell umfangreiche Modellelemente zur Spezifikation menschlicher Ressourcen und deren Beziehungen zu unterschiedlichen Benutzeraufgaben bereitgestellt. Über die ebenfalls vorgesehenen Indikatoren und Eskalationen kann im Benutzeraufgabenmodell zusätzlich die Perspektive der Ausnahmebehandlung eines Workflows berücksichtigt werden.

Das **benutzerzentrische Domänenmodell** greift als Erstes der beiden plattformunabhängigen Benutzerschnittstellenmodelle die im Workflow-Modell enthaltene Datenperspektive auf und ermöglicht eine präzise Spezifikation der Anforderungen dieser Perspektive. Die durch das **Domänenprofil**

umgesetzten Metamodellelemente rücken dabei den Auftraggeber als Anwender in den Mittelpunkt der Modellierung und gestatten eine an ihm ausgerichtete Spezifikation der inhaltlichen Eigenschaften der Benutzerschnittstelle. Diese Eigenschaften bilden die Grundlage des zweiten als **Strukturmodell** bezeichneten Benutzerschnittstellenmodells. Das **Strukturprofil** stellt zu dessen Spezifikation unterschiedlichste Benutzerschnittstellenelemente bereit, wodurch der strukturelle Aufbau einer Benutzerschnittstelle anhand der zuvor spezifizierten Inhalte erfasst werden kann.

Durch das **Dienstmodell** können die im Kontext der Ausführung eines Workflows benötigten Dienste näher spezifiziert werden, ohne dass dabei ein direkter Bezug zu einer konkreten Realisierung dieser Dienste hergestellt werden muss. Da auch die Benutzeraktionen eines Workflows durch Dienste unterstützt werden, kann das Workflow-Modell und insbesondere das benutzerzentrische Domänenmodell zur Ableitung der Spezifikation der benötigten Dienstnachrichten und Dienstparameter verwendet werden.

7.1.2 Transformation der Modelle der Benutzerinteraktion auf Softwareartefakte einer dienstorientierten Architektur

Um die Fehlerwahrscheinlichkeit und den Zeitbedarf einer manuellen Abbildung eines Workflow-Modells auf die angeführten Folgemodelle reduzieren zu können, wurde im zweiten Beitrag das Ziel einer automatisierten Erzeugung der plattformunabhängigen Modelle der Benutzeraktion verfolgt. Hierzu wurden zunächst die benötigten Abbildungsrelationen zwischen den einzelnen Elementen der Metamodelle entworfen und diese anschließend entsprechend der konkreten Syntax des Sprachteils QVT-*Relations* der standardisierten Transformationssprache *Query, View, Transformation* umgesetzt. Die auf diese Weise automatisiert erzeugbaren Modelle der Benutzerinteraktion stellen noch keinen Bezug zu den spezifischen Details einer Plattform her und können daher für unterschiedliche Plattformen ausgeprägt werden. Dementsprechend lag ein weiteres Ziel des zweiten Beitrags in der Konzeption einer als Ausführungsumgebung für Geschäftsprozesse mit Benutzerinteraktionen geeigneten dienstorientierten Architektur. Im Vergleich zu bestehenden dienstorientierten Referenzarchitekturen wurde eine Erweiterung um Dienste zur Verwaltung von Benutzeraufgaben und Identitäten vorgenommen. Zusätzlich wurde darauf geachtet, dass die durch ein Portal realisierte Benutzerschnittstelle der dienstorientierten Architektur die notwendige Flexibilität aufweist, um den Menschen bei der Abarbeitung verschiedenster Benutzeraufgaben im Rahmen der Geschäftsprozesse geeignet unterstützen zu können. Um die zuvor erzeugten plattformunabhängigen Modelle der Benutzeraktion auf die konzipierte dienstorientierte Architektur abbilden zu können, wurde am Beispiel des Benutzeraufgabenmodells die Transformation des plattformunabhängigen Modells bis auf ausführbare Benutzeraufgabenspezifikationen präsentiert.

7.1.3 Tragfähigkeitsnachweise

Die Tragfähigkeit der in dieser Arbeit entwickelten Beiträge wurde einerseits durch die Ausführung des konzipierten modellgetriebenen Vorgehens anhand eines aus dem Szenario der Lehre und Forschung am Karlsruher Institut für Technologie (KIT) entnommenen Geschäftsprozesses nachgewiesen. Hierbei kamen die konzipierten Metamodelle und die entwickelten Transformationen zur automatisierten Erzeugung der jeweiligen Zielmodelle der Benutzerinteraktion zum Einsatz. Mit deren Hilfe konnte der Geschäftsprozess „Prüfungsanmeldung“ unter Berücksichtigung der Aspekte der Benutzerinteraktion bis auf Softwareartefakte für die im Rahmen des KIT-weiten Projekts „Karlsruher Integriertes InformationsManagement“ (KIM) [UKA-KIM] etablierte dienstorientierte Architektur abgebildet werden.

Andererseits konnte die Tragfähigkeit der entwickelten Beiträge dieser Arbeit im Rahmen einer Kooperation mit der Industrie nachgewiesen werden, indem sie als Grundlage für eine Erweiterung eines bestehenden ERP-Systems dienten. Durch diese Erweiterung wurden die modellbasierte Spezifikation und die Ausführung von Geschäftsprozessen mit Benutzerinteraktion in dem bestehenden ERP-System ermöglicht.

7.2 Diskussion der Ergebnisse

Die folgenden Abschnitte unterziehen die entwickelten Beiträge dieser Arbeit einer detaillierten Bewertung, die auf der Grundlage des in Abschnitt 3.1 festgehaltenen Anforderungskatalogs vorgenommen wird.

7.2.1 Spezifikation der Benutzerinteraktion

Die an die Spezifikation der Benutzerinteraktion gestellten Anforderungen nehmen einerseits Bezug auf die Eigenschaften der benötigten Metamodelle und adressieren andererseits deren automatisierte Umsetzung mittels Transformationen. Dieser Abschnitt bewertet daher die in Kapitel 4 konzipierten Metamodelle und die in Kapitel 5 entwickelten Transformationen im Hinblick auf diese Anforderungen.

7.2.1.1 Integration in ein gemeinsames Vorgehensmodell

Ohne ein zugrunde liegendes Vorgehensmodell kann die Qualität der zu entwickelnden Softwareartefakte aufgrund der Komplexität heutiger IT-Systeme nicht gewährleistet werden. Daher sieht das in dieser Arbeit konzipierte modellgetriebene Entwicklungsvorgehen explizit die Integration der Entwicklungsaktivitäten vor, die durch die Berücksichtigung der Benutzerinteraktion zusätzlich notwendig werden. Wie im Tragfähigkeitsnachweis gezeigt wurde, können die konzipierten Metamodelle der Benutzerinteraktion entweder auf ein bestehendes Metamodell wie beispielsweise das Metamodell der UML oder durch ein eigenständiges *ECore*-Metamodell ausgeprägt und damit in bestehenden Entwicklungsumgebungen im Rahmen existierender modellgetriebener Entwicklungsvorgehen verwendet werden. Die hierdurch erreichte nachvollziehbare Entwicklung gewährleistet die Qualität der Softwareartefakte der Benutzerinteraktion.

7.2.1.2 Plattformunabhängigkeit

Eine wesentliche Herausforderung bei der Spezifikation der Benutzerinteraktion lag in der Bewältigung der Heterogenität der vorhandenen Plattformen, die insbesondere im Bereich der Benutzerschnittstellen durch unterschiedlichste Entwicklungsrahmenwerke, Softwarearchitekturen und Endgeräte stark ausgeprägt ist. Die in Kapitel 4 entwickelten Metamodelle der Benutzerinteraktion abstrahieren daher von dieser Heterogenität, indem sie – den Prinzipien der modellgetriebenen Architektur folgend – plattformspezifische Details zunächst ausblenden und eine rein fachliche Spezifikation der Benutzerinteraktion ermöglichen. Wie der Tragfähigkeitsnachweis am Beispiel des Benutzeraufgabenmodells zeigte, können diese plattformunabhängigen Modelle für unterschiedliche Zielplattformen durch eine entsprechende Anpassung der Transformationen ausgeprägt werden. Die Gültigkeit der plattformunabhängigen Modelle bleibt hiervon jedoch unberührt, wodurch insgesamt die geforderte Steigerung der Portabilität der entwickelten Softwareartefakte erreicht wird.

7.2.1.3 Spezifikation der Geschäftsanforderungen

Um die unterschiedlichen Anforderungen an einen Geschäftsprozess präzise auf die IT-Unterstützung in Form von Workflows abbilden zu können, müssen die in Abschnitt 2.2.3 eingeführten Anforderungsperspektiven bei der Spezifikation von Workflows beachtet werden. Während durch das Benutzeraktionsprofil bereits im Geschäftsprozessmodell eine verfeinerte Spezifikation der Kontrollflussperspektive im Hinblick auf die Interaktion eines Menschen ermöglicht wird, gestattet das konzipierte Metamodell für Benutzeraufgaben eine detaillierte Spezifikation der Ressourcenperspektive in Bezug auf die zur Ausführung eines Workflows benötigten menschlichen Ressourcen. Zusätzlich können durch das Benutzeraufgabenmodell mögliche Ausnahmebehandlungen im Zusammenhang mit der Ausführung von Benutzeraufgaben durch menschliche Ressourcen erfasst werden. Im Gegensatz zu bestehenden Spezifikationssprachen können so durch die konzipierten Metamodelle unter anderem die in Abschnitt 2.2.3.1 eingeführten Ressourcenmuster Delegation und Eskalation spezifiziert werden. Eine aus [Ho08b] entnommene Bewertung der in [RA+05] erfassten Ressourcenmuster zeigt, dass durch die Erweiterungen des Benutzeraufgabenprofils (BAP) eine verbesserte Unterstützung der Ressourcenmuster und der damit korrelierten Ressourcenperspektive eines Workflows erreicht werden kann. Ein + zeigt die vollständige Unterstützung eines Ressourcenmusters an, ein nicht unterstütztes Ressourcenmuster wird mit - gekennzeichnet. Für eine umfassende und detaillierte Ausführung der Bewertung sei auf [Ho08b] verwiesen.

Ressourcenmuster	UML	BAP
1. Direct Allocation	+	+
2. Role-based Allocation	+	+
3. Deferred Allocation	–	+
4. Authorisation	–	+
5. Separation of Duties	–	+
6. Case Handling	–	+
7. Retain Familiar	–	+
8. Capability-based Distribution	–	+
9. History-based Distribution	–	+
10. Organisational Distribution	–	+
11. Automatic Execution	+	+
27. Delegation	–	+
28. Escalation	–	+
34. Redo	–	–
43. Additional Resources	–	+

Nach [Ho08, RA+05] BAP = Benutzeraufgabenprofil

Tabelle 5: Bewertung des Benutzeraufgabenprofils

7.2.1.4 Aspekte der Benutzerschnittstelle

Zur Spezifikation einer funktionalen Benutzerschnittstelle müssen, wie in Abschnitt 2.3.2 festgehalten, im Wesentlichen die drei Aspekte Inhalt, Präsentation und Navigation berücksichtigt werden. Das aus einem Workflow-Modell erzeugte benutzerzentrische Domänenmodell fokussiert den inhaltlichen Aspekt der Benutzerschnittstelle, wobei das Hauptaugenmerk dieser Spezifikation auf dem Kunden und dessen Anforderungen als späterem Anwender liegt. Sind die Wünsche des Kunden im Modell erfasst, so dient das aus dem benutzerzentrischen Domänenmodell erzeugte Strukturmodell dem

Aspekt der Präsentation, indem es die grundlegende Struktur der Benutzerschnittstelle spezifiziert. Wie im Tragfähigkeitsnachweis gezeigt, abstrahieren das benutzerzentrische Domänenmodell und das Strukturmodell dabei von plattformspezifischen Details, weshalb beide Modelle je nach Anwendungsszenario für unterschiedliche Zielplattformen ausgeprägt werden können. Im Kontext der Zielplattform wird dann auch das ebenfalls zum Aspekt der Präsentation gehörende *Layout* einer Benutzerschnittstelle beachtet, indem das eigentliche Aussehen der Benutzerschnittstelle durch die gegebenen Möglichkeiten der Zielplattform konkretisiert wird. Der Aspekt der Navigation konnte aufgrund der Tatsache, dass einer Benutzeraufgabe als abgeschlossene Arbeitseinheit immer genau eine formularbasierte Benutzerschnittstelle zugeordnet wurde, minimalistisch betrachtet werden. Durch die im Strukturmodell vorgesehenen Container- und Aktionselemente kann zwar eine beliebige Unterteilung der Benutzerschnittstelle vorgenommen und ein als im Englischen als *GUI-Flow* bezeichnete Abfolge zwischen den einzelnen Teilen der Benutzerschnittstelle realisiert werden. Um jedoch die auf diese Weise möglichen Anwendungsszenarien über die Anforderungen dieser Arbeit hinaus erweitern und die Navigationspfade komplexer Benutzerschnittstellen erfassen zu können, wird ein eigenständiges Navigationsmodell benötigt. Im Ausblick zu dieser Arbeit zeigt Abschnitt 7.3.1 eine rasch umsetzbare Erweiterungsmöglichkeit des modellgetriebenen Entwicklungsvorgehens im Hinblick auf ein detailliertes Navigationsmodell auf.

7.2.1.5 Integration der Fachentwickler

Der in dieser Arbeit verfolgte Ansatz, die verschiedenen Aspekte der Benutzerinteraktion in jeweils eigenständigen Modellen zu betrachten, ermöglicht nach dem Prinzip der getrennten Zuständigkeiten (engl. *Separation of Concerns*) die Integration von Fachentwicklern, die sich auf die Anforderungen jeweils eines Aspektes der Benutzerinteraktion spezialisiert haben. Durch diese eigenständige Betrachtung kann die Komplexität der Modelle auf das für den Fachentwickler notwendige Maß reduziert werden. Im Umkehrschluss gestattet die Fokussierung des Experten auf sein Fachgebiet die Entwicklung qualitativ hochwertiger Ergebnisse. Gleichzeitig kann durch den verfolgten Ansatz eine Parallelität in einigen Teilen des Entwicklungsvorgehens erreicht werden. Während beispielsweise ein Domänenexperte das benutzerzentrische Domänenmodell um systeminterne Eigenschaften anreichert, kann parallel dazu ein weiterer Fachentwickler die Strukturierung der Benutzerschnittstelle vorantreiben. Ein dritter Fachentwickler kann mit der Spezifikationsverfeinerung der benötigten Benutzeraufgabenmodelle betraut werden. Somit gestattet es der Ansatz, einzelne Aspekte wie beispielsweise die Benutzerschnittstelle unabhängig von den weiteren Entwicklungszweigen auf ausführbare Softwareartefakte abzubilden und ermöglicht dadurch eine frühzeitige Verifikation der korrekten Umsetzung einzelner Teilanforderungen entsprechend den Wünschen des Auftraggebers. Das Risiko einer Fehlentwicklung kann damit reduziert werden.

7.2.1.6 Automatisierte Ausführung der Abbildung

Da eine manuelle Abbildung der entwickelten Modelle auf die jeweiligen Folgemodelle durch den Menschen eine erhebliche Fehlerwahrscheinlichkeit mit sich bringt und zusätzlich einen nicht unerheblichen Zeitaufwand einfordert, wurden in Kapitel 5 Metamodell-basierte Transformationen entwickelt. Diese ermöglichen eine automatisierte Erzeugung der Folgemodelle aus den Quellmodellen und reduzieren damit das Risiko einer fehlerhaften Interpretation eines Modells durch den Menschen sowie den zur Abbildung benötigten Zeitaufwand. Ferner wurden die benötigten Transformationen gegen die standardisierte Transformationssprache *Queries, Views, Transformations* entwickelt, um somit die Einsetzbarkeit und Wiederverwendbarkeit der Transformationsregeln in einer möglichst breiten Anzahl von Entwicklungswerkzeugen zu ermöglichen. Wie im Tragfähigkeitsnachweis gezeigt wurde,

können durch die automatisierten Transformationen rasch funktionale Softwareartefakte erzeugt und die Anforderungen des Kunden an diesen frühzeitig verifiziert werden.

7.2.2 Dienstorientierte Architektur zur Unterstützung von Benutzerinteraktion

Die Modelle der Benutzerinteraktion müssen auf Softwareartefakte einer dienstorientierten Architektur abgebildet werden, die zur Unterstützung von Benutzerinteraktion und hinsichtlich der daraus resultierenden Anforderungen geeignet ist. Dieser Abschnitt bewertet daher die in Kapitel 5 konzipierte dienstorientierte Architektur im Hinblick auf diese Anforderungen.

7.2.2.1 Unterstützung von Benutzerinteraktion

Die Unterstützung von Benutzerinteraktion bedingt zum einen die Bereitstellung einer Benutzerschnittstelle, über die der Benutzer mit dem Anwendungssystem interagieren kann. Die in Abschnitt 5.3 konzipierte dienstorientierte Architektur sieht hier ein Portal als Realisierung der Benutzerschnittstelle vor, über das der Benutzer Zugriff auf die unterschiedlichen Dienste einer dienstorientierten Architektur erhält. Zum anderen wurde durch die zusätzliche Fachkomponente der Benutzeraufgabenverwaltung an die bestehenden Konzepte aus dem Bereich der Workflow-Management-Systeme angeknüpft, um eine Ausführung von Benutzeraufgaben zu ermöglichen. Als zusätzliche Erweiterung wurde eine Identitätsverwaltung vorgesehen, die neben der Authentifizierung und Autorisierung der Benutzer die Abbildung abstrakter menschlicher Ressourcen auf konkrete Benutzer ermöglicht. Mit den vorgenommenen Erweiterungen der dienstorientierten Architektur wird letztendlich die angestrebte Abbildung und Ausführung von Geschäftsprozessen mit Benutzerinteraktion ermöglicht.

7.2.2.2 Interoperable Dienstschnittstellen

Da interoperable Schnittstellen eine wesentliche Grundvoraussetzung für den Erfolg der dienstorientierten Architekturen darstellen, wurden die zur Unterstützung von Geschäftsprozessen mit Benutzerinteraktion zusätzlich benötigten fachfunktionalen Komponenten ebenfalls als Dienste konzipiert, die ihre Fachfunktionalität über standardisierte Webservice-Schnittstellen bereitstellen. Hierzu wurden die zentralen Spezifikationen WS-HumanTask und *Web Services for Remote Portlets* (WSRP) herangezogen und bei der Konzeption der dienstorientierten Architektur entsprechend berücksichtigt. Eine flexible Unterstützung von Geschäftsprozessen mit Benutzerinteraktion unter Berücksichtigung der heute vorhandenen Heterogenität der IT-Landschaften wird hierdurch erreicht.

7.2.2.3 Integrationsfähigkeit und Flexibilität

Die konzipierte dienstorientierte Architektur ermöglicht eine bedarfsgerechte Anpassung der bereitgestellten Fachfunktionalität an das jeweils betrachtete Anwendungsszenario, indem sie die zur Unterstützung von Benutzerinteraktion zusätzlich benötigte Fachfunktionalität als jeweils eigenständige Dienste auffasst. Sind in einem konkreten Anwendungsszenario beispielsweise unterschiedliche IT-Systeme für die Ausführung von Benutzeraufgabenspezifikationen vorhanden, die als fachfunktionale Komponenten zur Verwaltung von Benutzeraufgaben herangezogen werden können, so können diese unter der Voraussetzung der vorhandenen interoperablen Dienstschnittstellen in die dienstorientierte Architektur je nach Bedarf integriert werden. Ist hingegen im Anwendungsszenario nur ein IT-System vorhanden, das sowohl die Fachfunktionalität einer Workflow-*Engine* als auch einer Benutzeraufgabenverwaltung bereitstellt, so kann diese Fachfunktionalität ebenfalls in die dienstorientierte Architektur entsprechend der benötigten Dienstschnittstellen integriert werden.

7.2.3 Resümee

Die im vorangegangenen Abschnitt durchgeführte Bewertung der Beiträge dieser Arbeit anhand des aufgestellten Anforderungskatalogs belegt, dass der in Kapitel 3 für diese Arbeit motivierte Handlungsbedarf und die in Abschnitt 1.3 aufgezeigten Problemstellungen erfolgreich bearbeitet wurden.

Durch die entwickelten plattformunabhängigen Metamodelle und deren Umsetzung auf bestehende Metamodelle stehen im Rahmen des verfolgten modellgetriebenen Entwicklungsvorgehens vielfältige Modellelemente zur Berücksichtigung der spezifischen Anforderungen der Benutzerinteraktion zur Verfügung. Hierzu können bereits bestehende Entwicklungsumgebungen zur Spezifikation der Modelle der Benutzerinteraktion herangezogen werden, die aufgrund der bekannten Metamodelle den Entwickler zusätzlich bei der Überprüfung der syntaktischen Korrektheit der von ihm erstellten Modelle unterstützen. Ferner kann aufgrund der in das Gesamtvorgehen integrierten Berücksichtigung dieser Modelle eine nachvollziehbare Entwicklung von einem Geschäftsprozessmodell als Ausgangspunkt bis hin zu den erzeugten, ausführbaren Softwareartefakten gewährleistet werden. Diese Nachvollziehbarkeit wird durch die spezifizierten Abbildungsbezüge zwischen den verschiedenen Modellelementen zusätzlich verstärkt und gestattet in letzter Konsequenz die Entwicklung automatisierbarer Abbildungen zwischen den unterschiedlichen Modellen.

Der gewählte Ansatz, die betrachteten Aspekte der Benutzerinteraktion durch jeweils eigenständige Modelle zu berücksichtigen, erweist sich – wie im Tragfähigkeitsnachweis demonstriert werden konnte – als vorteilhaft, da auf diese Weise qualifizierte Fachentwickler einzelne Teilaspekte unabhängig vom Fortschritt der Entwicklung der weiteren Teilaspekte rasch bis auf ausführbare Softwareartefakte abbilden können. Anhand dieser Softwareartefakte kann im Gespräch mit dem Kunden als Auftraggeber die Korrektheit der Umsetzung der Anforderungen des Auftraggebers verifiziert und das Risiko einer nicht rechtzeitig entdeckten Fehlentwicklung reduziert werden. Zusätzlich ermöglicht die eigenständige Betrachtung der einzelnen Aspekte der Benutzerinteraktion eine deutliche Komplexitätsreduktion für den Entwickler, da dieser folglich nur mit den für ihn relevanten Aspekten konfrontiert werden muss.

Ein weiterer zentraler Vorteil der vorgestellten Konzepte liegt, wie im Rahmen der Tragfähigkeitsnachweise belegt werden konnte, in der Plattformunabhängigkeit der spezifizierten Metamodelle. Durch die Abstraktion von plattformspezifischen Details müssen die Modelle der Benutzerinteraktion nur einmal entwickelt werden, um dann für unterschiedliche Zielplattformen ausgeprägt werden zu können. Die somit erreichte Portabilität der Modelle wirkt der heute vorhandenen Heterogenität der IT-Systeme entgegen. Dieser Heterogenität wird zusätzlich durch die im Rahmen der konzipierten dienstorientierten Architektur verwendeten Standards Rechnung getragen, wodurch ferner die Interoperabilität der als Dienste bereitgestellten Softwareartefakte sichergestellt werden kann.

Obwohl durch die vorgenommene Abstraktion von plattformspezifischen Details und die Anwendung des Ansatzes der getrennten Zuständigkeiten die Komplexität für den einzelnen Entwickler reduziert werden kann, muss immer im Kontext eines konkreten Anwendungsszenarios entschieden werden, ob der sich durch die Berücksichtigung der Benutzerinteraktion ergebenden Komplexitätssteigerung in ausreichendem Maße begegnet werden kann. Da die Umsetzung der entwickelten Metamodelle und Transformationen einen nicht unerheblichen initialen Aufwand erfordert, muss eine gewisse kritische Masse an umzusetzenden Geschäftsprozessen im Anwendungsszenario vorhanden sein, um den entstehenden Mehraufwand zu rechtfertigen.

7.3 Ausblick

Bei der Bearbeitung der spezifizierten Problemstellungen ergaben sich weiterführende Fragestellungen und Möglichkeiten zur Erweiterung der entwickelten Konzepte. Die folgenden beiden Abschnitte diskutieren zwei Erweiterungsmöglichkeiten, die kurzfristig umgesetzt werden können bzw. mit deren Umsetzung bereits begonnen wurde. Die sich anschließenden Abschnitte 7.3.3 und 7.3.4 hingegen diskutieren Einschränkungen, die im betrachteten Szenario dieser Arbeit aus Gründen der Fokussierung vorgenommen wurden und zeigen auf, inwieweit die entwickelten Konzepte ohne diese Einschränkungen angewendet werden können. Im letzten Abschnitt wird schließlich ein Ausblick auf eine mögliche mittelfristige Anwendung der entwickelten Konzepte im Rahmen der industriellen Entwicklung von Software gegeben.

7.3.1 Navigationsmodelle für Benutzerschnittstellen

Die Interaktion eines Menschen mit einem IT-System kann mittels eine Benutzerschnittstelle von einer einfachen Bestätigung einer Anfrage bis hin zur Erfassung umfassender Daten reichen. Während für die Bestätigung einer einfachen Anfrage in der Regel eine ebenso einfache Benutzerschnittstelle ausreichend ist, kann die Eingabe umfassender Daten eine komplexe Benutzerschnittstelle erfordern. Um den Benutzer hierbei geeignet zu unterstützen, wird in bestehenden Anwendungen die Benutzerschnittstelle häufig in verschiedene Masken unterteilt, die in einer bestimmten Reihenfolge abgearbeitet werden können. Durch diese im Englischen als *GUI-Flow* bezeichnete Unterteilung einer Benutzerschnittstelle kann die Komplexität für den Benutzer reduziert werden. Um die Spezifikation eines *GUI-Flow* bereits auf der Modellebene berücksichtigten zu können, muss die Benutzerschnittstelle über ein umfassendes Navigationsmodell verfügen, das alle Navigationsmöglichkeiten des Benutzers innerhalb eines festgelegten Zustandsraumes umfasst. Durch das im Strukturprofil vorgesehene Containerelement und Aktionselement kann in einem Strukturmodell zwar eine beliebige Unterteilung der Benutzerschnittstelle vorgenommen und eine sequenzielle Abfolge einzelner Containerelemente als Repräsentation von Masken realisiert werden. Eine komplexe, nicht sequenzielle Abfolge vieler Masken sieht das Strukturmodell aufgrund seiner statischen Natur aber nicht vor. Daher wird zur Spezifikation dieses dynamischen Aspekts der Benutzerschnittstelle ein eigenständiges Navigationsmodell benötigt.

Eine Erweiterungsansatz des konzipierten modellgetriebenen Entwicklungsvorgehens, der die Integration eines eigenständigen Navigationsmodells ermöglicht, sei an einem weiteren Beispiel aus dem bereits im Tragfähigkeitsnachweis bemühten Aufgabenbereich der Lehre und Lehrorganisation erläutert. Muss eine Fachkraft im Zuge der Ausführung des Geschäftsprozesses „Studierenden immatrikulieren“ die Daten eines Studierenden vollständig erfassen, so werden hierzu die Personalien des Studierenden, seine Adressdaten und seine Fachrichtung benötigt. Dementsprechend sieht das in Abbildung 92 dargestellte Strukturmodell je ein Containerelement für die Personalien, die Adressdaten und die Fachrichtung des Studierenden vor. Aus dem Strukturmodell kann nun mit einer Modell-zu-Modell-Transformation ein Zustandsautomat abgeleitet werden, der zunächst alle Containerelemente in Form von Zuständen enthält und im Anschluss an die Transformation durch die Entwickler mit Kanten im Sinne von Navigationspfaden verfeinert werden muss. Durch die Kanten kann spezifiziert werden, in welcher Art und Weise der Benutzer über dem Zustandsraum der Benutzerschnittstelle navigieren kann. Im gegebenen Beispiel in Abbildung 92 kann der Benutzer dementsprechend von der als Zustand dargestellten Maske `Personalien` zu den Zuständen `Adresse` oder `Fachrichtung` navigieren. Vom Zustand `Adresse` aus besteht die Möglichkeit, die Bearbeitung der Masken und damit die Benutzeraufgabe abzuschließen.

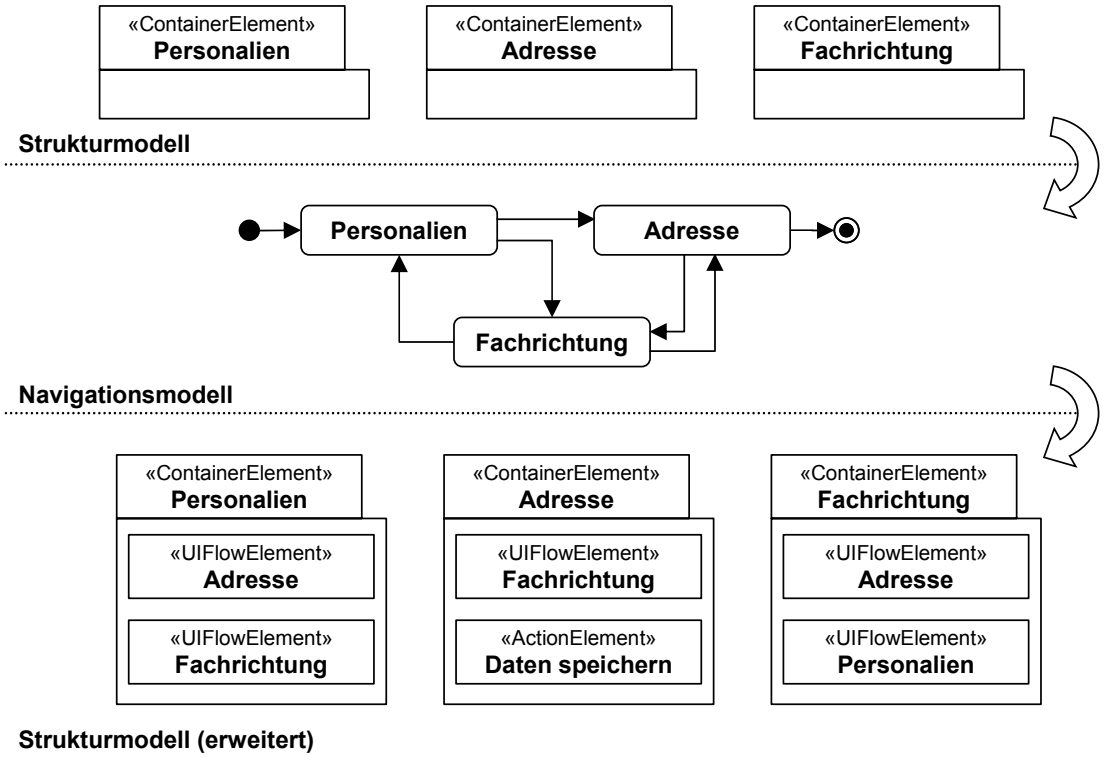

Abbildung 92: Navigationsmodell einer Benutzerschnittstelle

Ist das Navigationsmodell vollständig spezifiziert, so können die neu verfügbaren Informationen entweder ergänzend in das plattformunabhängige Strukturmodell zurück oder unter Zuhilfenahme des Plattformmodells der Zielplattform direkt in das plattformspezifische Strukturmodell transformiert werden. Um die unterschiedliche Auswirkung der Navigation eines Benutzers im Strukturmodell erfassen zu können, sollte das Aktionselement des Strukturprofils um die Spezialisierung eines `UIFlowElements` erweitert werden. Ein `UIFlowElement` löst im Gegensatz zu einem Aktionselement nicht den Abschluss einer Benutzeraufgabe aus, sondern zeigt dem Benutzer lediglich das in einer Eigenschaft des `UIFlowElements` festgehaltene Containerelement an. Ferner kann es sinnvoll sein, Eigenschaftswerte, die durch den Benutzer in einer ersten Maske eingegeben wurden, in einer zweiten Maske erneut anzuzeigen. Hat die Fachkraft in der ersten Maske `Personalien` beispielsweise bereits den Namen und Vornamen des Studierenden erfasst, so können diese Werte als Orientierungshilfe in allen Folgemasken aufgelistet werden. Hierzu können an den Kanten des Navigationsmodells Eigenschaften bzw. Eigenschaftswerte annotiert werden, die zwischen den einzelnen Masken der Benutzerschnittstelle weitergegeben werden sollen, ohne dass in einem Zwischenschritt eine Persistierung der zuvor eingegebenen Daten durch die Workflow-*Engine* veranlasst werden müsste.

7.3.2 Überwachbarkeit der Benutzerinteraktion

Eine zusätzliche Erweiterungsmöglichkeit der entwickelten Konzepte ergibt sich im Hinblick auf die angestrebte Überwachbarkeit eines Menschen bei der Ausführung einer Benutzeraufgabe. Da Unternehmen eine kontinuierliche Verbesserung ihrer Geschäftsprozesse anstreben, um im globalen Wettbewerb konkurrenzfähig zu bleiben, muss die Interaktion des Menschen in einem Geschäftsprozess hinsichtlich möglicher Optimierungspotenziale untersucht werden. Grundlage für eine Identifizierung dieser Optimierungspotenziale ist zunächst die Bereitstellung aussagekräftiger Informationen bezüglich der Interaktion eines Menschen. Das in dieser Arbeit konzipierte Metamodell sieht hier als einfachen Indikator der Überwachung die Zeitspanne vor, innerhalb welcher Benutzeraufgaben erfüllt

werden müssen. Wie der Autor in [FH+09] publiziert hat, kann diese Überwachung im Rahmen des verfolgten modellgetriebenen Vorgehens konsequent ausgebaut und um zusätzliche Indikatoren erweitert werden. Die Benutzerschnittstelle als zentraler Berührungspunkt des Menschen mit den IT-Systemen der IT-Unterstützung kann hier wichtige Informationen liefern, wie an einem Beispiel verdeutlicht werden soll. In einer Benutzeraktion „Studierendendaten eingeben“ eines Workflows muss eine Fachkraft den Namen und Vornamen eines Studierenden eingeben. Daher verfügt die zugeordnete Benutzerschnittstelle über zwei entsprechende Eingabefelder Name und Vorname. Bestünde nun die Möglichkeit, die Zeit, die unterschiedliche Fachkräfte zum Befüllen der beiden Eingabefelder benötigten, über einen längeren Zeitraum zu messen, so könnten aus den erhaltenen Daten verschiedenste Rückschlüsse gezogen und Informationen gewonnen werden. Benötigen beispielsweise viele Fachkräfte zur Eingabe des Vornamens im Vergleich zur Eingabe des Namens unverhältnismäßig mehr Zeit, so kann für den Workflow gegebenenfalls der Rückschluss gezogen werden, dass entweder der Vorname der Fachkraft nicht in geeigneter Form vorliegt und sie diesen aus einer anderen Quelle erst heraussuchen muss, oder dass das zur Eingabe bereitgestellte Eingabefeld nicht den Anforderungen der Fachkraft entspricht.

Ein möglicher, in [FH+09] publizierter Ansatz zur Gewinnung von Daten über das Eingabeverhalten der Benutzer besteht, wie in Abbildung 93 exemplarisch an einem Strukturmodell dargestellt, in der Auszeichnung der zu untersuchenden Benutzerschnittstellenelemente durch einen zusätzlichen Stereotyp `observable`.

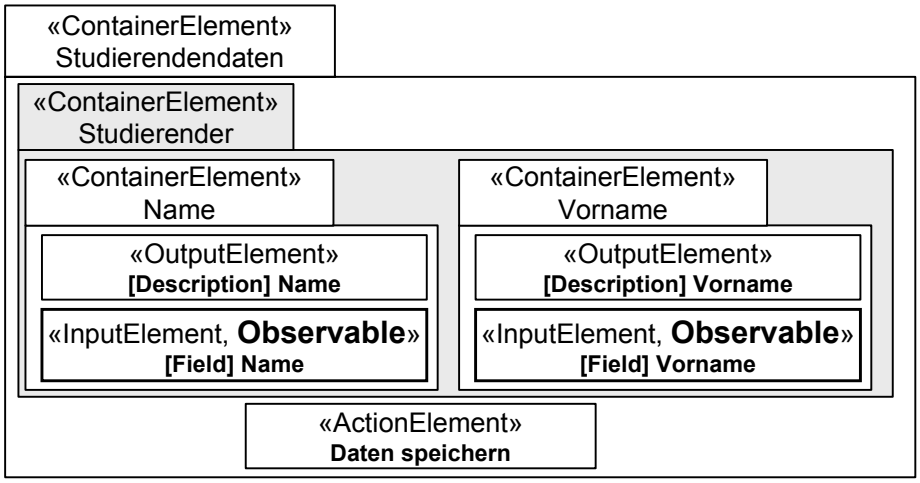

Abbildung 93: Beispielhafte Erweiterung eines Strukturmodells

Ein als `observable` stereotypisiertes Benutzerschnittstellenelement wird bei der Umsetzung auf ein plattformspezifisches Modell, wie beispielsweise das im Tragfähigkeitsnachweis verwendete *Web-Part*-Modell, auf ein weiteres als „Haken“ (engl. *Hook*) bezeichnetes Modellelement abgebildet, das geeignete Operationen und Attribute für die Überwachung des Benutzerschnittstellenelements bereitstellt. Die in [FH+09] bisher prototypisch umgesetzten Konzepte zur Überwachung von Benutzeraktionen können unter Berücksichtigung der in [Mo09] und [Me07] entwickelten Ansätze für eine umfassende Überwachung der Interaktion eines Benutzers ausgebaut und im Hinblick auf die Optimierung eines Geschäftsprozesses gewinnbringend angewendet werden. Auf diese Weise können zeitintensive Benutzeraktionen in einem Workflow erkannt und durch eine Anpassung der Benutzerschnittstelle oder eine veränderte Ressourcenzuordnung der zugeordneten Benutzeraufgabe optimiert werden. Wird diese Erweiterungsmöglichkeit konsequent weiterverfolgt, so ist eine automatisierte Adaption der Benutzerschnittstelle, aber auch der Benutzeraufgaben nach klar zu definierenden Richtlinien denkbar.

Während in den vorangegangenen Abschnitten kurzfristig umsetzbare Erweiterungen diskutiert wurden, greifen die folgenden beiden Abschnitte Annahmen aus dem betrachteten Szenario dieser Arbeit (vgl. Abschnitt 1.2) auf und zeigen, durch welche Maßnahmen eine Aufhebung dieser Annahmen zugunsten einer breiteren Anwendbarkeit der entwickelten Konzepte erreicht werden kann.

7.3.3 Entwicklung der benötigten Dienste

Im betrachteten Szenario wurde die zur Unterstützung der Benutzerinteraktion benötigte Fachfunktionalität, wie etwa eine Benutzeraufgabenverwaltung, aber auch ein dienstorientiertes Portal, in Form von Diensten als gegeben angenommen. Durch diese Fokussierung konnten die entwickelten Modelle und Transformationen entlang des verfolgten geschäftsgetriebenen Entwicklungsvorgehens im Sinne eines sogenannten *Top-Down*-Ansatzes eingesetzt und auf diese Dienste abgebildet werden. In vielen Anwendungsszenarien sind die zur Unterstützung von Benutzerinteraktion benötigten Dienste jedoch nicht vorhanden. Gerade die in dieser Arbeit als Dienst aufgefasste Benutzeraufgabenverwaltung kann in bestehenden dienstorientierten Architekturen heute nicht als gegeben betrachtet werden (vgl. Abschnitt 3.3). Dementsprechend muss das zur Umsetzung eines Geschäftsprozesses auf die IT-Unterstützung verfolgte Entwicklungsvorgehen auch die Bereitstellung und gegebenenfalls die Entwicklung der benötigten Dienste mit berücksichtigen.

In diesem Zusammenhang ist ein reiner *Top-Down*-Ansatz, wie er in dieser Arbeit verfolgt wird, nicht ausreichend. Vielmehr muss der nicht näher betrachtete Übergang von einem auf hohem und abstraktem Niveau erfassten Geschäftsprozess hin zu einem auf die IT umsetzbaren Workflow-Modell (vgl. T0, Abbildung 79) näher beleuchtet und die bei diesem Übergang entstehenden Fragestellungen detailliert untersucht werden. Um beispielsweise feststellen zu können, welche Arbeitseinheiten eines Geschäftsprozesses prinzipiell durch die IT unterstützt werden können, muss ein Experte, der mit den in einem Unternehmen vorhandenen IT-Systemen vertraut ist, die Anforderungen der Geschäftsebene den vorhandenen Diensten gegenüberstellen. Um den Experten bei dieser Aufgabe geeignet unterstützen zu können, nimmt das in Abschnitt 5.3 eingeführte Dienstverzeichnis eine zentrale Stellung ein. Die hierbei auftretenden Fragestellungen, beispielsweise nach einer effizienten und semantischen Suche in einem Dienstverzeichnis, werden in [Kr07] detailliert behandelt.

Stehen in einem Dienstverzeichnis eines Unternehmens alle zur Umsetzung eines Geschäftsprozesses notwendigen Dienste zur Verfügung, so müssen diese Dienste im Entwicklungsvorgehen lediglich integriert werden. Allerdings ist die zur Umsetzung eines Geschäftsprozesses benötigte Fachfunktionalität häufig nicht oder nur teilweise vorhanden und muss daher zunächst entwickelt werden. Dementsprechend muss das verfolgte Vorgehensmodell auch eine Entwicklungsphase zur Entwicklung der benötigten Dienste vorsehen. Hierbei treten neben reinen technischen Fragestellungen verschiedene konzeptionelle Fragestellungen auf, beispielsweise nach einer geeigneten Granularität der zu entwickelnden Dienste. Je nachdem, wie detailliert die Spezifikation eines Geschäftsprozesses bzw. Workflows vorgenommen wird, müssen den spezifizierten Aktionen passende atomare oder komponierte Dienste zugeordnet werden. Vielversprechende Hinweise auf mögliche Vorgehensweisen zur Bestimmung geeigneter Dienste sind in [Er05] zu finden.

Um das in dieser Arbeit verfolgte Entwicklungsvorgehen somit auch auf Anwendungsszenarien ausweiten zu können, in denen die benötigten Dienste nicht vorhanden und gegebenenfalls erst zu entwickeln sind, muss der Übergang zwischen einem Geschäftsprozess und einem Workflow-Modell nach einem systematischen Vorgehen erfolgen, das die Granularität der Workflow-Modelle in Relation zu den vorhandenen bzw. den zu entwickelnden Diensten setzt. In einem weiteren Entwicklungsschritt muss nach der Spezifikation des Workflow-Modells auch die Entwicklung der benötigten

Dienste vorangetrieben werden. Das ebenfalls aus dem Workflow-Modell abgeleitete Dienstmodell, in dem die benötigten Dienstparameter im Kontext von Dienstnachrichten bereits erfasst sind, kann hierfür als Ausgangspunkt eines weiteren Entwicklungspfades dienen (vgl. Abschnitt 4.5).

7.3.4 Einsatz der Konzepte in unterschiedlichen Entwicklungsprozessen

Wie in Abschnitt 2.1 näher beleuchtet wurde, unterstützen die Prinzipien der modellgetriebenen Softwareentwicklung die Entwicklung komplexer IT-Systeme. Dabei gibt die modellgetriebene Softwareentwicklung an sich kein eigenes Vorgehensmodell zur Softwareentwicklung vor. Vielmehr können ihre Prinzipien auf beliebige Vorgehensmodelle zur Softwareentwicklung angewendet werden, solange diese den Einsatz von Modellen vorsehen. Als ein Beispiel für ein solches Entwicklungsvorgehen wurde im betrachteten Szenario die geschäftsgetriebene Entwicklung verfolgt, die einen besonderen Schwerpunkt auf die Geschäftsprozesse eines Unternehmens und deren Umsetzung auf die IT legt. Zusätzlich kamen im Rahmen der geschäftsgetriebenen Entwicklung die Konzepte der modellgetriebenen Architektur der OMG zum Einsatz, in dem die entwickelten Metamodelle und deren Modellinstanzen anhand der durch die modellgetriebene Architektur vorgeschlagenen Modellhierarchie ausgerichtet wurden.

Die Anwendbarkeit der Konzepte dieser Arbeit ist jedoch weder an die geschäftsgetriebene Entwicklung, noch an die modellgetriebene Architektur gebunden. Vielmehr wurden die betrachteten Metamodelle, wie etwa das Benutzeraufgabenmetamodell oder die Metamodelle der Benutzerschnittstelle, bewusst als jeweils eigenständige Modelle entwickelt, um deren Anwendbarkeit in unterschiedlichen Entwicklungsvorgehen gewährleisten zu können. Da hierbei immer das konzeptionelle Metamodell im Vordergrund stand, muss die Umsetzung der Metamodelle auch nicht zwingend an der Modellhierarchie der MDA ausgerichtet werden. Daher können neben einem Workflow-Modell, das im Rahmen der geschäftsgetriebenen Entwicklung als Quellmodell für die Erzeugung der Benutzerinteraktionsmodelle verwendet wurde, auch andere Modelle als Quellmodelle verwendet werden, sofern diese über den gleichen Informationsgehalt verfügen. Verschiedene Forschungsansätze, die in Kapitel 3 diskutiert wurden, sehen hier anstatt eines Geschäftsprozessmodells andere Modelle, die aus dem Forschungsgebiet der Anforderungserhebung (engl. *Requirements Engineering*) bekannt sind, als Ausgangspunkt der Entwicklung (vgl. dazu u. a. [RF+04], [Ko06], [NT+99]). Um die in dieser Arbeit entwickelten Konzepte in diesen Entwicklungsvorgehen einsetzen zu können, müssen die horizontalen Transformationen zur Erzeugung der Modelle der Benutzerinteraktion an die jeweils zur Verfügung stehenden Quellmodelle angepasst werden.

Neben den zuvor erörterten Erweiterungen des betrachteten Szenarios dieser Arbeit soll abschließend ein mittelfristig möglicher Einsatz der entwickelten Konzepte in einem größeren industriellen Kontext diskutiert und damit ein Ausblick auf die Anwendbarkeit der modellgetriebenen Entwicklung von Geschäftsprozessen mit Benutzerinteraktion gegeben werden.

7.3.5 Einsatz der entwickelten Konzepte zur Softwareindustrialisierung

Zur Unterstützung von Geschäftsprozessen kommen heute häufig Individuallösungen zum Einsatz, deren Entwicklung langwierig und teuer ist. Daher werden im Forschungsfeld der Softwaretechnik unterschiedliche Konzepte und Methoden untersucht, die eine Reduktion von Zeit und Kosten bei der Softwareherstellung ermöglichen sollen. Ein vielversprechender Ansatz findet sich hierbei in der sogenannten Softwareindustrialisierung, deren Ziel in einer industriellen Fertigung von Software nach dem Vorbild industrieller Güter wie etwa Maschinen oder Werkzeugen liegt (vgl. [HO07, Ta05]). Mit dem Aufkommen der modellgetriebenen Entwicklungsverfahren stehen neue Methoden und Werkzeu-

ge bereit, die es gestatten, Software nicht mehr in Form einzelner Unikate herstellen zu müssen, sondern eine industrielle Produktion von Software unterstützen [GS03]. In [Ta05] wird hierbei der Vergleich zur Automobilindustrie bemüht, in der die industrielle Produktion seit vielen Jahren zum Einsatz kommt. Auf der Grundlage identischer Fahrzeugplattformen können unterschiedliche Fahrzeugmodelle gefertigt werden, die jeweils auf die Gegebenheiten verschiedener Anwendungsszenarien und die Bedürfnisse entsprechender Kundenkreise angepasst sind. Ein weiteres klassisches Prinzip der Industrialisierung liegt in der Verringerung der Fertigungstiefe und in einem daraus resultierenden Zukauf einzelner Bauteile von hochspezialisierten und -qualifizierten Zulieferern. Auf diese Weise konnten in der Vergangenheit die Kosten der Herstellung von Fahrzeugen gesenkt und gleichzeitig deren Qualität gesteigert werden. In gleicher Weise können die modellgetriebenen Entwicklungsverfahren zu einer Kostenreduktion und Qualitätssteigerung in der Softwareentwicklung durch eine Spezialisierung beitragen [AC+07].

Aufgrund der Ausrichtung an einem modellgetriebenen Entwicklungsvorgehen können auch die in dieser Arbeit entwickelten Konzepte im Rahmen eines Ansatzes zur Softwareindustrialisierung vorteilhaft eingesetzt werden. Da gerade im Bereich der Benutzerinteraktion die Heterogenität der IT-Systeme, unter anderem aufgrund einer Vielzahl von Endgeräten, wie etwa Desktop-Rechner, Mobiltelefone oder Anzeigetafeln, besonders ausgeprägt ist, steht der initial zu erbringende Aufwand zur Entwicklung der benötigten Modelle und Transformationen deren Wiederverwendung zur Erzeugung ähnlicher Lösungen für verschiedene Zielplattformen gegenüber. Wie gezeigt wurde, können die spezifischen Anforderungen der Benutzerinteraktion im Sinne einer fachlichen Spezifikation erfasst und durch Transformationen auf unterschiedliche Zielplattformen ausgeprägt werden. Ferner ermöglicht die getrennte Betrachtung der verschiedenen Aspekte der Benutzerinteraktion eine Arbeitsteilung bei der Entwicklung der verschiedenen Softwareartefakte der Benutzerinteraktion, sodass die Integration von spezialisierten Teillösungen ermöglicht wird. Im Hinblick auf diese Teillösungen spielt auch das Paradigma der Dienstorientierung eine wesentliche Rolle. Fachfunktionalität, die aufgrund ihrer Spezialisierung einen hohen Ressourcenaufwand in der Entwicklung fordert, kann durch die Dienstorientierung in Form eines Dienstes in unterschiedlichen Softwareprodukten als Softwarebaustein wiederverwendet werden (vgl. [PW07, AC+07]). Der Dienst einer Benutzeraufgabenverwaltung, der zur Unterstützung von Geschäftsprozessen in einer dienstorientierten Architektur benötigt wird, oder eine spezielle Benutzerschnittstelle können als Beispiele genannt werden, die im Sinne der Softwareindustrialisierung Softwarebausteine darstellen, die in unterschiedliche Softwareprodukte integriert werden können. Insgesamt ist durch die Anwendung der Konzepte dieser Arbeit im Rahmen einer Softwareindustrialisierung somit eine verbesserte Qualität der erzeugten Softwareartefakte und unter der Annahme einer hinreichend häufigen Anwendbarkeit der Modelle und Transformationen gleichzeitig eine Reduktion der Kosten zu erwarten [GS03].

Allerdings muss auch beachtet werden, dass für eine Umsetzung der in dieser Arbeit entwickelten Konzepte im Rahmen einer Softwareindustrialisierung in einem Unternehmen Fachkräfte benötigt werden, die nicht nur über technische Qualifikationen im Bereich der modellgetriebenen Entwicklung und der dienstorientierten Architekturen verfügen, sondern auch den damit einhergehenden organisatorischen Wandel in einem Unternehmen stützen und vorantreiben können. Während in der Praxis die technischen und organisatorischen Herausforderungen und Auswirkungen der Etablierung der dienstorientierten Architekturen gerade intensiv diskutiert und bewertet werden, können sich die modellgetriebenen Ansätze, unter anderem aufgrund der damit einhergehenden tief greifenden Veränderungen in der Softwareentwicklung, nur langsam in der Industrie etablieren (vgl. dazu u. a. [PM06, HT06]). Wie in [HO07] konstatiert, trägt der akute Mangel an entsprechend qualifizierten Fachkräften zusätzlich zu einer nur schleppend vorangehenden Etablierung neuer Konzepte und Geschäftsfelder, wie

etwa der Softwareindustrialisierung, bei. Dementsprechend bleibt abzuwarten, inwieweit der im sogenannten Dagstuhl-Manifesto [BJ+06] publizierte Aufruf namhafter Forscher nach einem Ausbau des *Software Engineering* in Deutschland zur einer nachhaltigen Verbesserung der Verfügbarkeit qualifizierter Fachkräfte und damit zu einer rascheren Konsolidierung und Umsetzung moderner Konzepte der Informatik führen kann.

Anhang

A. Vollständige Transformationsspezifikationen

Dieser Abschnitt des Anhangs greift einige der in Kapitel 5 entwickelten Transformationen erneut auf und führt im Hinblick auf vollständige Transformationsspezifikationen die zuvor ausgeblendeten Relationen und Hilfsmethoden ein.

1. Transformation T3: Benutzerzentrisches Domänenmodell – Strukturmodell

Um die vertikale Position eines Containerelements in einem Strukturmodell überschneidungsfrei bestimmen zu können, muss aus einer gegebenen Menge an Integern der größte Wert ermittelt werden. Die folgende Hilfsmethode liefert diese Funktionalität.

Query getMaximum

Gibt das Maximum aus einer Liste von Integern zurück.

```
query getMaximum(positions : Sequence(Integer)) : Integer
{
  positions -> iterate(position; maximum : Integer = 0 |
    if(maximum > position)
    then
      maximum
    else
      position
    endif)
}
```

Query getPositionValues

Diese Hilfsmethode durchläuft eine Liste von Elementen (`elements`) und fügt die Positionsangaben (`positionParameter`) jedes Containerelements in eine Liste ein, die abschließend zurückgegeben wird.

```
query getPositionValues(elements : OrderedSet(uml::Element),
     positionParamenter : String) : Sequence(Integer)
{
  elements->iterate(element; positions : Sequence(Integer) = Sequence{} |
    if(element.oclIsTypeOf(DomainProfile::ContainerElement))
    then
     if(positionParamenter = 'positionTop')
        then positions->including(
         element.oclAsType(DomainProfile::ContainerElement).positionTop)
        else positions->including(
         element.oclAsType(DomainProfile::ContainerElement).positionLeft)
        endif
    else true
    endif
    )
  )
}
```

Query determineInputType

Anhand des Typs des übergebenen Parameters muss der für ein Eingabeelement benötigte Eingabetyp (`InputType`) bestimmt werden. Ist der Typ des Parameters ein boolescher Wert (`Boolean`), so wird der Wert `InputTypes::Boolean` zurückgegeben. Im Falle eines `Integer` oder `UnlimitedNatural` wird `InputTypes::Number` ausgegeben, `InputTypes::Text` in allen weiteren Fällen.

```
query determineInputType(attribute : uml::Property)
     :StructureProfile::InputTypes {
  if(attribute.type = Boolean.oclAsType(uml::Type))
  then
    InputTypes::Boolean
  else
    if(attribute.type = Integer.oclAsType(uml::Type) or
       attribute.type = UnlimitedNatural.oclAsType(uml::Type))
    then
      InputTypes::Number
    else
      InputTypes::Text
    endif
  endif
}
```

Query elementExistsInContainer

Diese Hilfsmethode bekommt ein Containerelement (`container`) und den Bezeichner eines zweiten Elements (`elemName`) übergeben. Daraufhin wird über die im Containerelement bereits enthaltenen Elemente iteriert und deren Bezeichner auf Gleichheit mit `elemName` geprüft. Ist ein Element mit dem gleichen Bezeichner vorhanden, liefert die Hilfsmethode `True` zurück, ansonsten `False`.

```
query elementExistsInContainer(
     container : StructureProfile::ContainerElement, elemName : String)
     : Boolean {
  container.packagedElement -> exists(pElem | pElem.name = elemName)
}
```

2. Transformation T6: Strukturmodell – WebPart-Modell

Um das plattformunabhängige Strukturmodell einer Benutzerschnittstelle mittels einer Metamodell-basierten Transformation auf ein *WebPart*-Modell abbilden zu können, muss zunächst ein Plattformmodell für die Zielplattform ASP.NET bereitgestellt werden. Da ASP.NET insgesamt ein mächtiges Entwicklungsrahmenwerk für Webanwendungen repräsentiert, wird für die zu entwickelnde Transformation nur der zur Spezifikation von *WebParts* relevante Teil der Klassenbibliothek von ASP.NET in Form eines Plattformmodells benötigt, der in Abbildung 94 dargestellt ist.

Das zentrale und als abstrakt definierte Element `Control` definiert grundlegende Eigenschaften eines Benutzerschnittstellenelements, die von allen weiteren Elementen des Plattformmodells vererbt werden. Das Element `Panel` als Spezialisierung des abstrakten Elements `WebControl` kann analog zum Containerelement des Strukturprofils weitere Spezialisierungen des `Control`-Elements enthal-

ten und dient daher der Strukturierung einer Benutzerschnittstelle. Alle im weiteren Verlauf benötigten Spezialisierungen des Elements `WebControl` sind in Abbildung 94 entsprechend ihrer Funktionalität analog zur Ordnung des in Abbildung 47 (Seite 116) dargestellten Strukturprofils aufgeführt. Im linken Teil finden sich Benutzerschnittstellenelemente zur Spezifikation von Aktionen, die durch den Benutzer ausgelöst werden können. Im mittleren Teil finden sich Benutzerschnittstellenelemente, die zur Spezifikation von Eingaben von und Ausgaben an den Benutzer verwendet werden können. Bei den Eingabeelementen nehmen die Klassen `CheckBoxList` bis `RadioButtonList` eine Sonderstellung ein, da bei diesen Elementen die Eingabemöglichkeiten des Benutzers durch eine `ListItemCollection`, die über beliebig viele `ListItems` verfügen kann, näher spezifiziert werden müssen. Eine detaillierte Beschreibung aller Elemente des Plattformmodells und deren in Abbildung 94 ausgeblendeten Eigenschaften ist in [Ge09] zu finden.

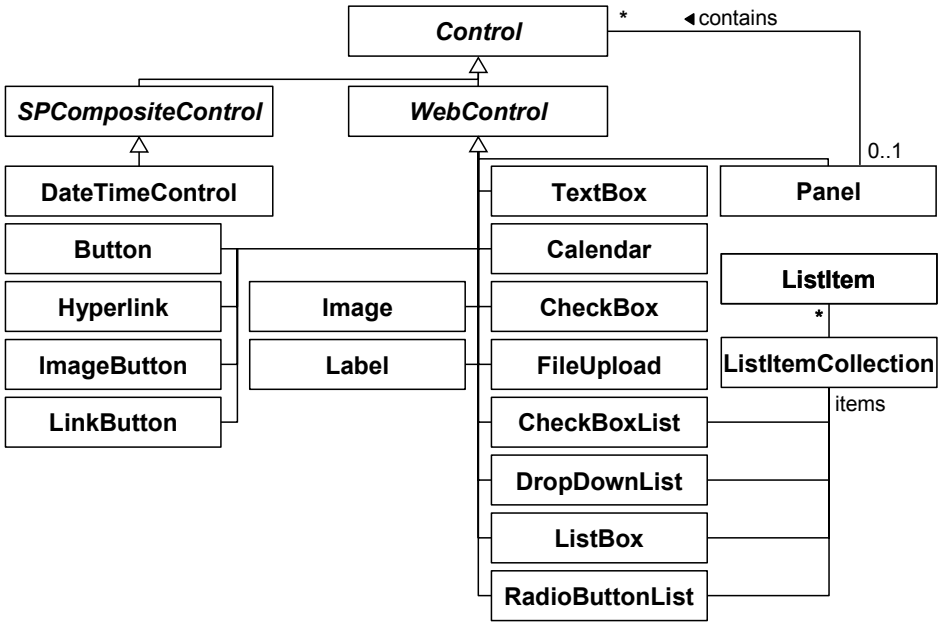

Abbildung 94: Relevanter Ausschnitt des Plattformmodells von ASP.NET

Um das in Abbildung 94 auszugsweise dargestellte Plattformmodell von ASP.NET zur Spezifikation im modellgetriebenen Entwicklungsvorgehen verwenden zu können, muss das Plattformmodell durch ein konkretes Metamodell umgesetzt werden. Neben der Möglichkeit, das Plattformmodell durch ein *ECore*-Modell umzusetzen, kann auch die Erweiterung des UML-Metamodells durch ein UML-Profil in Erwägung gezogen werden. Obwohl ein UML-Profil die Komplexität des UML-Metamodells übernimmt, kann dieser Faktor für das *WebPart*-Modell ausgeblendet werden. Der Grund hierfür liegt in der Konzeption des plattformunabhängigen Strukturmodells. Wie Abschnitt 5.2.4 zeigt, nutzt das Strukturprofil das UML-Metamodellelement `Package`, um die Enthaltenseinbeziehung der einzelnen Benutzerschnittstellenelemente auszudrücken. Hierdurch sind weitere Assoziationen zwischen Benutzerschnittstellenelementen unnötig und können daher durch eine einzige in OCL formalisierte Beschränkung unterbunden werden. Die Komplexität der UML wird auf diese Weise zwar drastisch eingeschränkt, die verbleibenden Spezifikationsmöglichkeiten sind für das Strukturmodell jedoch völlig ausreichend. Wird dieser Ansatz für das *WebPart*-Modell übernommen, so kann die Komplexität der UML auch hier stark reduziert werden, womit der Aufwand für die Umsetzung des Plattformmodells von ASP.NET auf ein UML-Profil vertretbar wird. Der wesentliche Mehrwert eines UML-Profils liegt in der breiten Werkzeugunterstützung und der konkreten Syntax der UML, welche die Verfeinerung eines *WebPart*-Modells erleichtert.

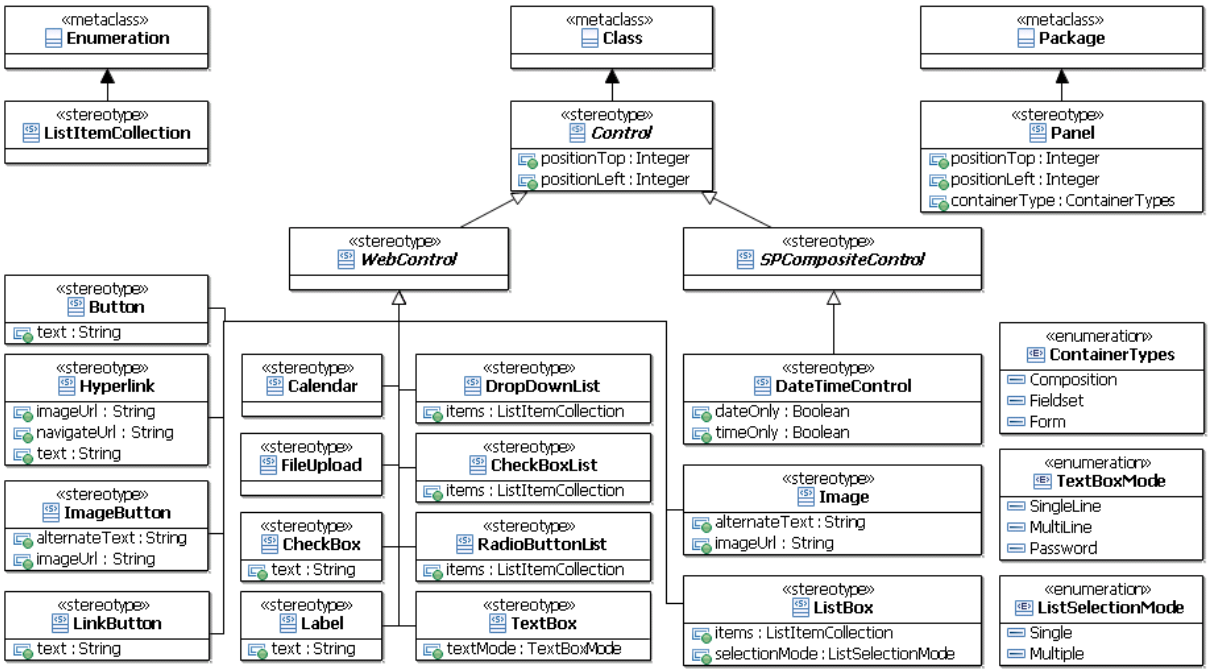

Abbildung 95: Umsetzung der WebPart-Elemente als UML-Profil in RSA

Zur Umsetzung des Plattformmodells vom ASP.NET auf das UML-Metamodell und ein im weiteren Verlauf als ***WebPart*-Profil** bezeichnetes UML-Profil müssen, wie in Abbildung 95 dargestellt, die drei UML-Metaklassen `Class`, `Enumeration` und `Package` durch entsprechende Stereotypen erweitert werden. Das Metamodellelement `Panel`, das für die Strukturierung der Benutzerschnittstelle verwendet wird, ist daher gleichsam dem Containerelement des Strukturprofils als Erweiterung der UML-Metaklasse `Package` spezifiziert. Eine `ListItemCollection` umfasst mehrere Auswahlmöglichkeiten des Benutzers und erweitert daher die UML-Metaklasse `Enumeration`. Alle weiteren Elemente repräsentieren einfache Benutzerschnittstellenelemente und werden daher als Spezialisierung der UML-Metaklasse `Class` entsprechend der durch die Klassenbibliothek von ASP.NET vorgegebenen Hierarchie umgesetzt.

Durch das *WebPart*-Profil stehen in RSA alle zur Spezifikation eines *WebPart* relevanten Konzepte als Stereotypen zur Verfügung. Somit kann ein *WebPart* durch ein Modell spezifiziert werden, das vom eigentlichen Quellcode eines *WebPart* abstrahiert. Um im nächsten Entwicklungsschritt eine automatisierte Erzeugung des *WebPart*-Modells (PSM) auf der Grundlage des Strukturmodells (PIM) zu ermöglichen, müssen zuerst unterschiedliche Abbildungsbezüge zwischen den Stereotypen beider Modelle hergestellt und je nach konkreter Ausprägung eines PIM-Elements ein geeignetes PSM-Element als Ziel der Transformation gewählt werden. Abbildung 96 gibt daher einen Überblick über die benötigten Relationen der zu entwickelnden Modell-zu-Modell-Transformation T6.

Während ein als Stereotyp `ContainerElement` ausgeprägtes Modellelement des PIM direkt auf ein Modellelement mit Stereotyp `Panel` des PSM abgebildet werden kann, ist bei allen weiteren Stereotypen des PIM auch der gewünschte Typ und die gewünschte Visualisierung des PIM-Elements zur Herstellung eines Bezuges mit einem Stereotyp des PSM ausschlaggebend. Ist beispielsweise ein Ausgabeelement (`OutputElement`) im plattformunabhängigen Strukturmodell als Bild (`Image`) spezifiziert, so muss dieses PIM-Element auf den Stereotyp `Image` des *WebPart*-Modells abgebildet werden. Ist das Ausgabeelement hingegen vom Typ `Text`, so muss ein auf dem *WebPart*-Modell ein Modellelement mit Stereotyp `Label` erzeugt werden.

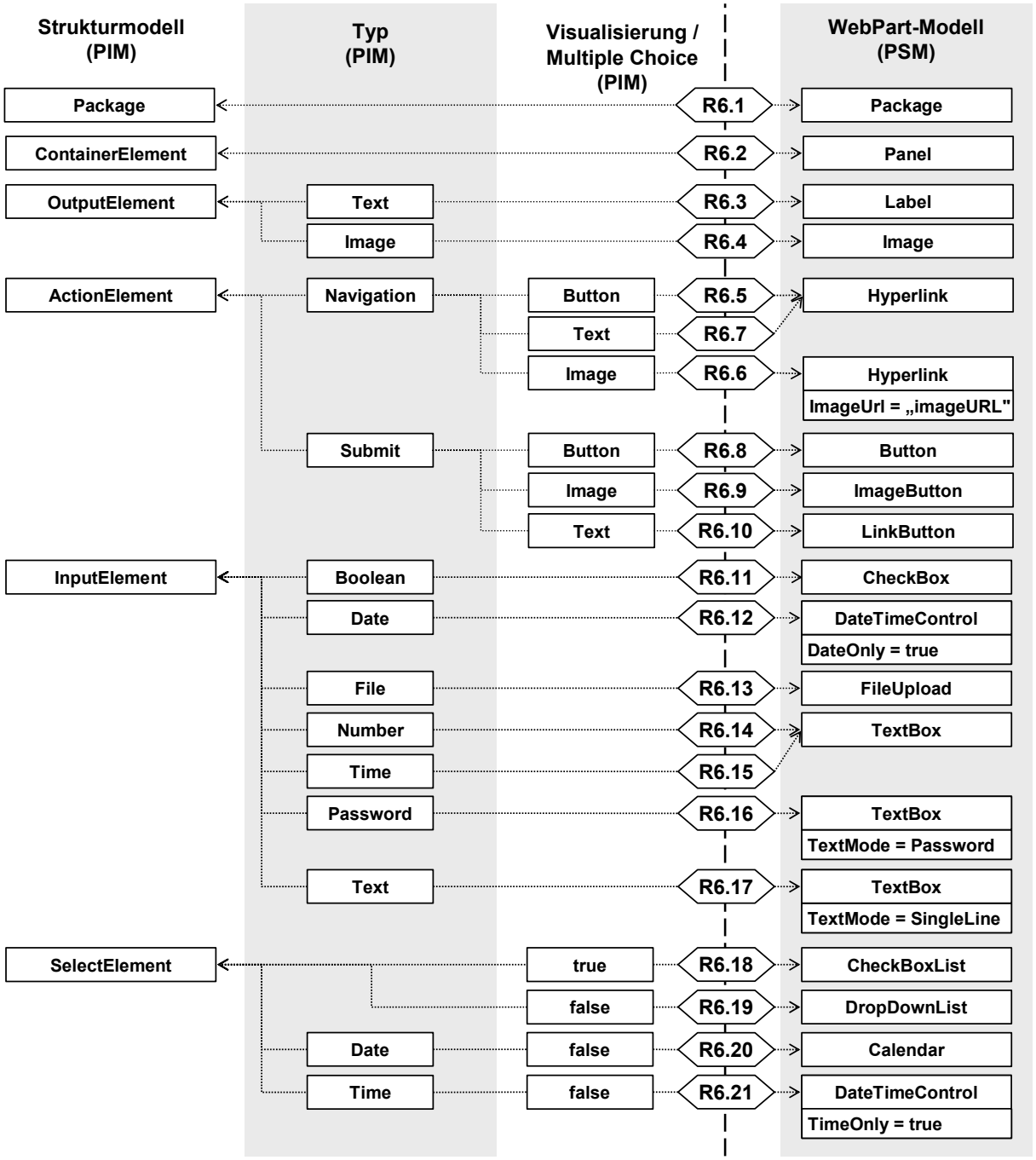

Abbildung 96: Abbildungsbezüge der Modellelemente der Strukturmodelle

Für eine automatisierte Erzeugung des *WebPart*-Modells sollen nun alle Relationen durch QVT-*Relations* formalisiert und durch MediniQVT implementiert werden. Da die Relationen wie im Überblick in Abbildung 96 dargestellt immer nach dem gleichen Schema aufgebaut sind, zeigen die folgenden Abschnitte auszugsweise die Umsetzung der Relationen R6.1, R6.2 und R6.3.

Relation R6.1 – Package_To_Package

Da in einem Workflow-Modell mehrere Benutzerinteraktionen enthalten sein können, sind die daraus indirekt erzeugten Strukturmodelle in einem gemeinsamen Paket zusammengefasst. Somit bekommt die Relation `Package_To_Package` (R6.1) zwei UML-Pakete als Eingabeparameter übergeben und erzeugt für jedes Strukturmodell, das im ersten Paket (`PIM_UIPackage`) enthalten ist, ein neues

WebPart-Modell im Paket der Zieldomäne (`PSM_UIPackage`). Ferner generiert R6.1 für das im Strukturmodell als `ContainerElement` enthaltene Wurzelelement (`contElement`) der Benutzerschnittstelle den Stereotyp `Panel` im *WebPart*-Modell. Die Eigenschaften des Containerelements werden dabei vollständig durch das erzeugte `Panel` übernommen. R6.1 baut somit das Grundgerüst des *WebPart*-Modells auf und fordert anschließend in der *where*-Klausel die Gültigkeit aller zusätzlich benötigten Relationen ein.

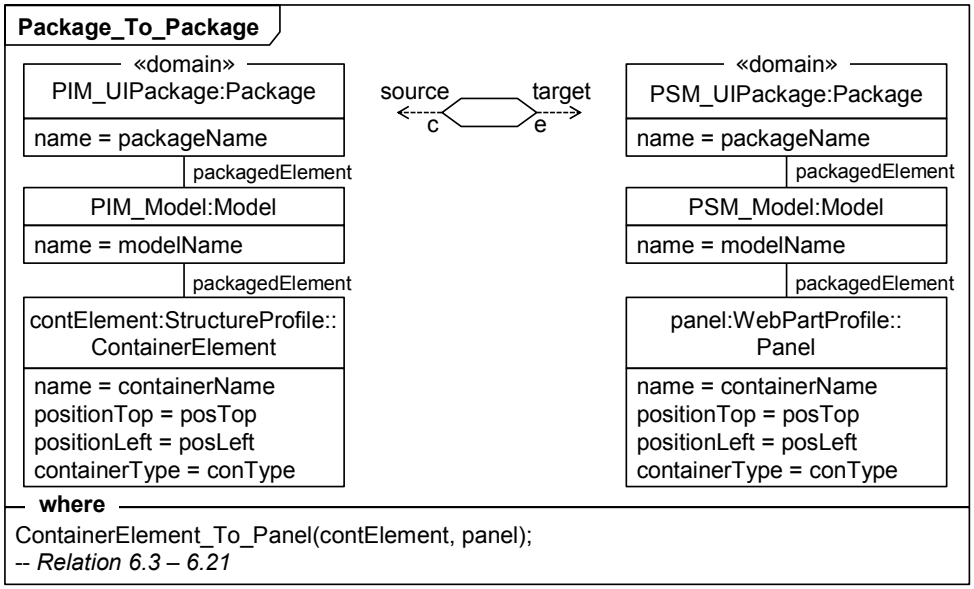

Abbildung 97: Package_To_Package-Relation in QVT-Relations

Relation R6.2 – ContainerElement_To_Panel

Die Gültigkeit der Relation `ContainerElement_To_Panel` (R6.2) wird von R6.1 eingefordert. Hierzu erstellt die Relation für jedes Containerelement (`innerContElement`), das in einem als Parameter der Relation übergebenen Containerelement (`contElement`) enthalten ist, ein neues als `innerPanel` bezeichnetes Modellelement `Panel` auf dem Zielmodell. Im nächsten Schritt ruft die Relation sich selbst auf, um auf diese Weise alle weiteren verschachtelten Containerelemente beliebiger Tiefe bei der Transformation berücksichtigen zu können. Abschließend fordert R6.2 ebenfalls die Gültigkeit aller weiteren Transformationen von T6 ein.

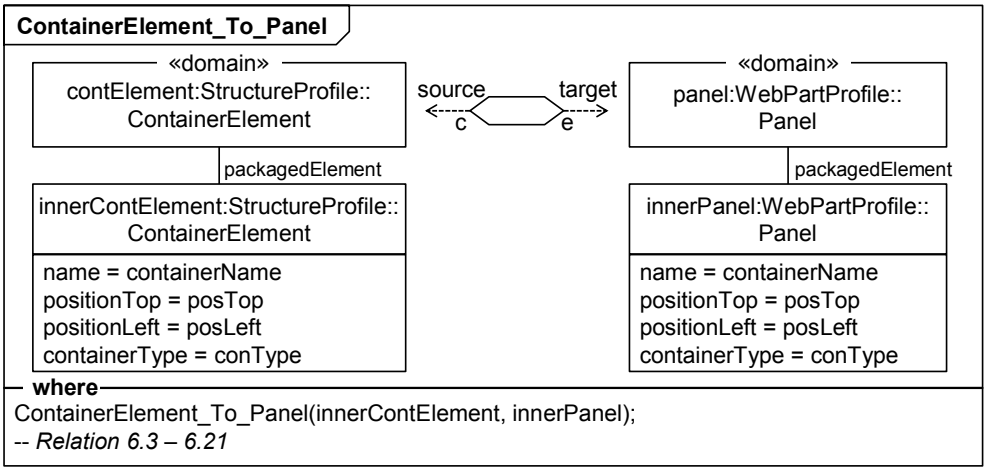

Abbildung 98: ContainerElement_To_Panel-Relation in QVT-Relations

Relation R6.3 – OutputElement_To_Label

Jedes weitere Benutzerschnittstellenelement des Strukturmodells (PIM) muss anhand seines Typs und seiner *Tagged Values* in ein Element des *WebPart*-Modells (PSM) transformiert werden. Der Aufbau der hierzu benötigten Relationen R6.3 - R6.21 ist sehr ähnlich: Den Relationen werden als Eingabeparameter immer ein Containerelement aus dem PIM und ein *Panel* aus PSM übergeben. Vom Containerelement aus wird nach einem Modellelement gesucht, das je nach Relation eine bestimmte Wertbelegung seiner *Tagged Values* aufweist. Wird ein entsprechendes Modellelement gefunden, so wird entsprechend den in Abbildung 96 dargestellten Abbildungsbezügen auf dem Zielmodell im *Panel* ein neues PSM-Element angelegt und dieses mit den adäquaten *Tagged Values* belegt. Ein Ausgabeelement (`OutputElement`) beispielsweise, das durch R6.3 transformiert wird, kann abhängig vom Ausgabetyp (`outputType`) entweder auf ein *Label* oder ein *Image* im PSM abgebildet werden.

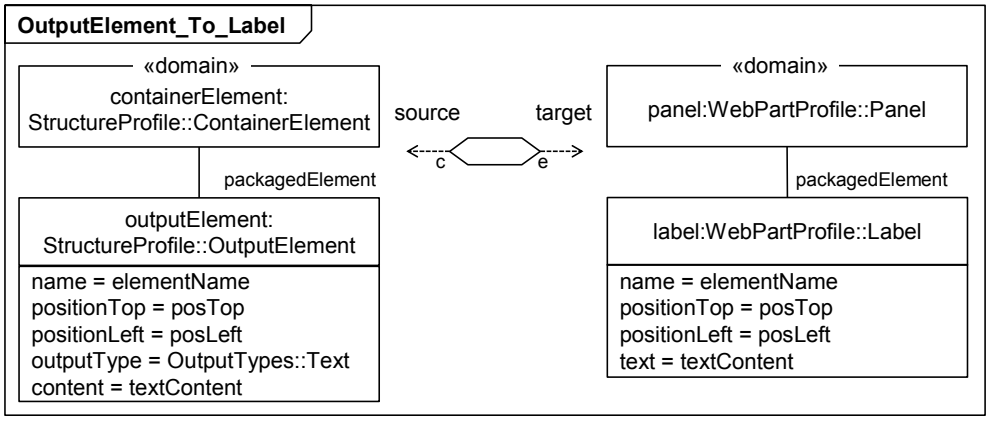

Abbildung 99: OutputElement_To_Label-Relation in QVT-Relations

Die in Abbildung 99 dargestellte Relation 6.3 greift für den Fall, dass ein Ausgabeelement vom Typ Text (`OutputTypes::Text`) ist und erzeugt deswegen im *WebPart*-Modell ein *Label* (`label`), das den Inhalt (`content`) des Ausgabeelements in einem *Tagged Value* `text` speichert. Dem gleichen Schema folgend müssen für alle Stereotypen Relationen entwickelt und anschließend für eine automatisierte Erzeugung der *WebPart*-Modelle für MediniQVT umgesetzt werden.

3. Transformation T7: WebPart-Modell – WebPart-Spezifikation

Der nachfolgend gezeigte Ausschnitt *Xpand*-Vorlage dient zur Generierung des *WebPart*-Quellcodes aus dem *WebPart*-Modell. Hierzu werden zunächst das Metamodell der UML, das *WebPart*-Profil und einige Hilfsmethoden, die aus Gründen der Übersichtlichkeit ausgelagert wurden, importiert. Im Anschluss daran werden die zur Transformation benötigten Vorlagen definiert. Die erste Vorlage `main`, die in Analogie zu höheren Programmiersprachen wie etwa Java den Einsprungpunkt in die Vorlagen darstellt, ruft für jedes übergebene Modell über `EXPAND` eine weitere Vorlage `WebPart` auf. Der Aufruf der gesamten Transformation an sich erfolgt dabei aus einem sogenannten OAW-Workflow heraus, der die gesamte Konfiguration der Transformation, inklusive des Einstiegpunkts in die Transformation enthält. In der *WebPart*-Vorlage wird dann zunächst über die eingebundene Hilfsmethode `getFileName` eine neue Datei mit dem Namen des Modells erzeugt und anschließend das Grundgerüst eines *WebPart* nach der Spezifikation von ASP.NET aufgebaut. Hierzu gehören einige *using*-Direktiven, der Namensraum des *WebPart*, aber auch die Klasse des *WebPart* an sich. Im Anschluss wird eine weitere Vorlage `AttributeDeclaration` aufgerufen, die anhand des übergebenen Pakets über alle Attribute iteriert, die durch den Benutzer über die Benutzerschnittstelle

eingegeben werden können, und diese im *WebPart* deklariert. Die Erzeugung der eigentlichen Benutzerschnittstelle selbst erfolgt dann in einer weiteren als `CreateChildControlsFor` bezeichneten Vorlage, die für jeden Benutzerschnittstellencontainer (`uiContainer`) der Übersichtlichkeit halber eine eigene Methode im *WebPart* erstellt. Die Rümpfe dieser Methoden werden durch eine weitere Vorlage `UIElement` gefüllt, welche die einzelnen Benutzerschnittstellenelemente in die dazugehörenden Panels einhängt. Die vollständige Spezifikation der Transformationsvorlagen, des OAW-Workflows sowie aller benötigter Hilfsmethoden kann aus [Ge09] und [Ji09] entnommen werden.

```
«IMPORT uml»
«IMPORT WebPartProfile»
«EXTENSION template::HelperMethods»

«DEFINE main FOR Package»
  «EXPAND WebPart FOREACH this.packagedElement.typeSelect(Model)»
«ENDDEFINE»

«DEFINE WebPart FOR Model»
  «FILE this.getFileName()-»
  using Microsoft.SharePoint;
  using Microsoft.SharePoint.WebControls;
  using Microsoft.SharePoint.WebPartPages;

  namespace WebPart_T7
  {
    public class WebPart_T7 : System.Web.UI.WebControls.WebParts.WebPart
    {
      «EXPAND AttributeDeclaration FOREACH
       this.packagedElement.typeSelect(Package)»

      public WebPart_T7() {
        this.ExportMode = WebPartExportMode.All;
      }

      protected override void CreateChildControls() {
         base.CreateChildControls();

    «FOREACH this.packagedElement.typeSelect(Package) AS uiContainer-»
      CreateChildControlsFor«uiContainer.name.getValidVarName()»();
    «ENDFOREACH»

    «FOREACH this.packagedElement.typeSelect(Package) AS uiContainer»
      private void
        CreateChildControlsFor«uiContainer.name.getValidVarName()»() {
          «EXPAND UIElement("this", 0) FOR uiContainer-»
      }
    «ENDFOREACH»
    …
  «ENDFILE»
«ENDDEFINE»
```

B. Implementierung der Transformationen

1. Benötigte Anpassungen der Transformationen für MediniQVT

Die in Kapitel 5 entwickelten Transformationen müssen, um in einem Entwicklungsvorgehen eingesetzt werden zu können, für MediniQVT bzw. OAW implementiert werden. Bei der Umsetzung der entwickelten Transformationen auf MediniQVT ist dabei zu beachten, dass die zum Zeitpunkt der Erstellung dieser Arbeit aktuelle Version 1.6 von MediniQVT Stereotypen nicht als eigene Typen erkennen und daher nur auf den Modellelementen des UML-Metamodells operieren kann. Um diese fehlende Funktionalität bereitzustellen und Stereotypen erkennen und setzen zu können, stehen im Rahmen der Entwicklungsumgebung Eclipse allerdings einige Hilfsmethoden zur Verarbeitung von Stereotypen zur Verfügung. So gibt beispielsweise die nachfolgend dargestellte Hilfsmethode `isStereotypeApplied` die booleschen Werte `True` oder `False` zurück, je nachdem, ob ein als Bezeichner übergebener Stereotyp aus einem ebenfalls als Bezeichner übergebenen Profilnamen auf das übergebene Element (`element`) angewandt wurde oder nicht. Die Hilfsmethode bedient sich dabei der durch das EMF bereitgestellten Methode `getApplied-Stereotype`, die somit in MediniQVT ebenfalls zur Verfügung steht.

```
query isStereotypeApplied(element : uml::Element, profileName : String,
stereotypeName : String) : Boolean {
  not element.getAppliedStereotype(profileName + '::' + stereotypeName)
    .oclIsUndefined()
}
```

Eine zweite Hilfsmethode `applyProfile` widmet sich der Anwendung eines Profils auf ein Modell. Die Hilfsmethode nimmt ein Paket als Generalisierung eines Modells sowie einen Profil-Bezeichner entgegen und liefert mithilfe der durch das EMF bereitgestellten Methode `getAppliedProfile` `True` oder `False` zurück, je nachdem, ob das Profil bereits auf das Modell angewandt wurde oder nicht. Allerdings kann EMF nur diejenigen Profile erkennen, die in der Werkzeugumgebung bereits zur Verfügung gestellt wurden. Daher werden durch `applyProfile` alle verfügbaren Instanzen von UML-Profilen (`uml::Profile.allInstances`) anhand des Profilnamens (`profileName`) durchsucht.

```
query applyProfile(aPackage:uml::Package, profileName:String):Boolean {
  if (aPackage.getAppliedProfile(profileName).oclIsUndefined())
  then
    not aPackage.applyProfile(uml::Profile.allInstances()
      -> select(profile | profile.name = profileName)
      -> asSequence()
      -> first()).oclIsUndefined()
  else true
  endif
}
```

2. Transformation T1: Workflow-Modell – Benutzeraufgabenmodell

Der folgende Ausschnitt zeigt den wesentlichen Kern der Transformation T1, deren *Top-Relation* `WorkflowModel_To_UserTask-Model` zuerst ein Modell auf der Zieldomäne, das alle Benut-

zeraufgaben enthalten soll, erzeugt. Die zweite Relation `UserAction_To_UserTask` fügt für jede Benutzeraktion des Quellmodells eine Benutzeraufgabe auf dem Zielmodell hinzu. Nach Ausführung von T1 müssen die Stereotypen des Benutzeraktionsprofils auf dem Zielmodell angewendet werden. Die vollständige Spezifikation der Transformationen T1 ist in [Ji09] zu finden.

```
transformation WorkflowModel_To_UserTaskModel(workflowModel:uml,
               userTaskModel:uml) {

  /* first relation generates a model for each workflow */
  top relation Model_To_Model {

    / * workflowModel is source of transformation */
    checkonly domain workflowModel sourceModel : uml::Model {
                    name = modelName };

    / * userTaskModel is the target of transformation */
    enforce domain userTaskModel targetModel : uml::Model {
                  name = modelName };

    where {
      /* helper method to apply profile */
      profile = applyProfile(targetModel, 'UserTaskProfile');

      /* second relation */
      UserAction_To_UserTask(profile, sourceModel, targetModel);
    }
  }

  / * second relation addresses user actions in workflowModel */
  relation UserAction_To_UserTask { ... }
}
```

3. Transformation T3: Workflow-Modell – Benutzerzentrisches Domänenmodell

Um eine automatisierte Erzeugung des benutzerzentrischen Domänenmodells aus dem Workflow-Modell zu erreichen, muss die in Abschnitt 5.2.3 entwickelte Transformation T2 in MediniQVT und das konzipierte Domänenprofil in RSA bereitgestellt werden. Erneut bedarf es hierzu einer Adaption der entwickelten Relationen entsprechend den Spezifika des EMF, weshalb Abbildung 100 beispielhaft die angepasste Relation R2.1 `InputPin_To_UserOutputInterface` zeigt. Im Gegensatz zu der in Abbildung 61 (Seite 137) dargestellten Relation nutzt die adaptierte Relation keine Stereotypen in den Domänenmustern, sondern lediglich die aus dem UML-Metamodell bekannten Modellelemente. Der Zugriff auf die Stereotypen erfolgt erst in der *where*- und *when*-Klausel der Relation über die Hilfsmethoden wie beispielsweise `applyUIProfile`. In der *when*-Klausel muss in diesem Fall explizit geprüft werden, ob die im Quellmodell gesuchte Aktion (`userAction`) mit dem Stereotyp `UserAction` des Benutzeraktionsprofils ausgezeichnet ist, bevor die Transformation durchgeführt wird. Die *where*-Klausel hält fest, dass nach der Erzeugung der Elemente auf dem Zielmodell (das benutzerzentrische Domänenmodell) das Domänenprofil angewendet werden muss, bevor dann die Stereotypen (`applyUIStereotype`) und *Tagged Values* der Stereotypen gesetzt (`setTaggedValue`) werden können. Die vollständig angepasste Transformation T2 ist ausführlich in [Ge09] beschrieben.

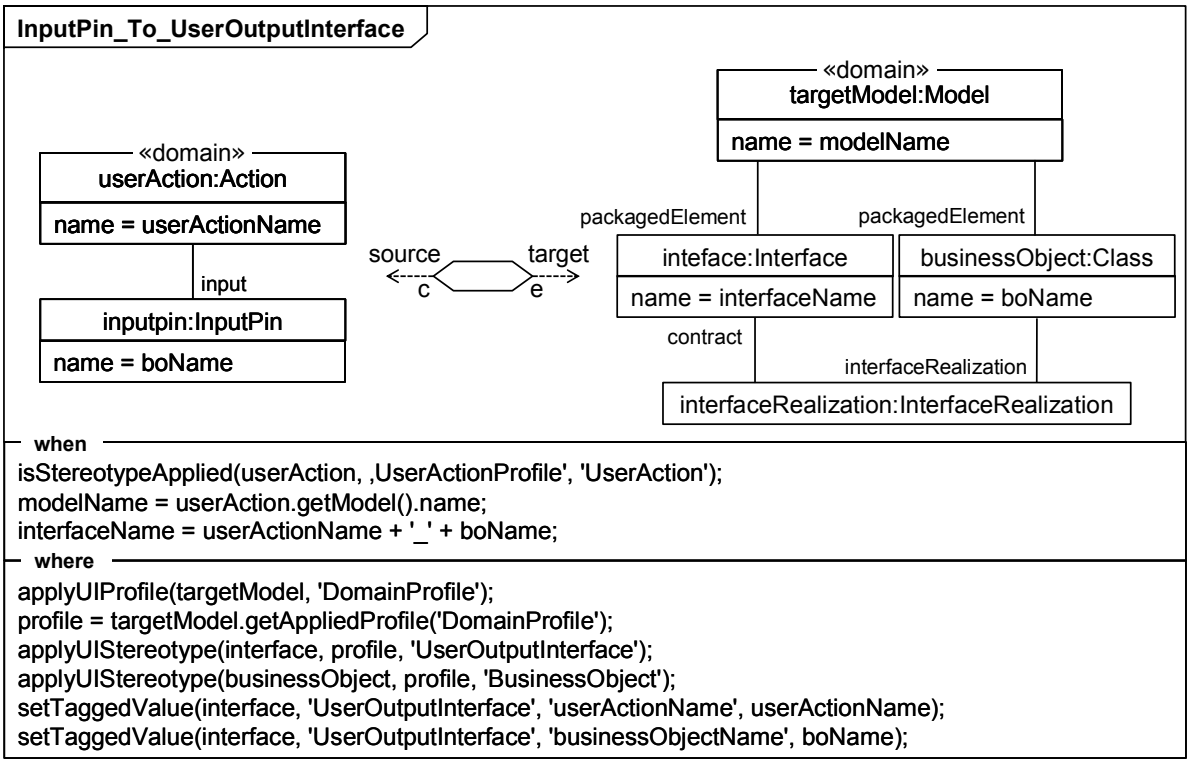

Abbildung 100: Angepasste QVT-Relation InputPin_To_UserOutputInterface

4. Transformation T6: Strukturmodell – WebPart-Modell

Bei dieser Umsetzung der Transformation T6 auf MediniQVT ergibt sich allerdings eine Verlagerung der Komplexität in dem Sinne, dass aufgrund der Tatsache, dass EMF die Stereotypen der Modellelemente nicht erkennt, alle Relationen R6.3 bis R6.21 zu einer einzigen Relation R6.3a zusammengefasst werden können. Die Unterscheidung, welcher Stereotyp auf dem Quell- und auf dem Zielmodell vorliegt, wird daher in die Hilfsmethoden der Relation hineinverlagert. Der folgende Abschnitt zeigt Relation R6.3a als eine Spezialisierung der Relationen R6.3 bis R6.21 für die Implementierung in MediniQVT.

Relation R6.3a – PIM_UIElement_To_PSM_UIElement

Die aufgrund der Spezifika von EMF benötigte und als `PIM_UIElement_To_PSM_UIElement` bezeichnete Relation R6.3a dient als Ersatz für die Relationen R6.3 bis R6.21. R6.3a überprüft in der *when*-Klausel durch die Hilfsmethode `isUIElement`, ob im übergebenen Containerelement des Strukturmodells eine Klasse des Stereotyps `Action-`, `Input-`, `Select-` oder `OutputElement` vorhanden ist. Ist dies der Fall, dann wird der Stereotyp dieser Klasse genau bestimmt (`PIMStereotypeName`) und anhand dessen *Tagged Values* der Stereotyp für das auf dem Zielelement neu erstellte Modellelement festgelegt (`getApplicablePSMStereotype`). Abschließend wendet die Relation den zuvor festgelegten Stereotyp auf das neu erzeugte Modellelement an und setzt auf allen Modellelementen zusätzlich benötigte *Tagged Values* (`setAdditionalTaggedValues`). Die Unterscheidung, welcher Stereotyp im Quellmodell vorliegt und im Zielmodell erzeugt werden soll, wird damit aus den Domänenmustern der ursprünglichen Relationen in die Hilfsmethoden von Relation R6.3a verlagert. Abbildung 101 gibt eine vereinfachte Darstellung dieser Relation.

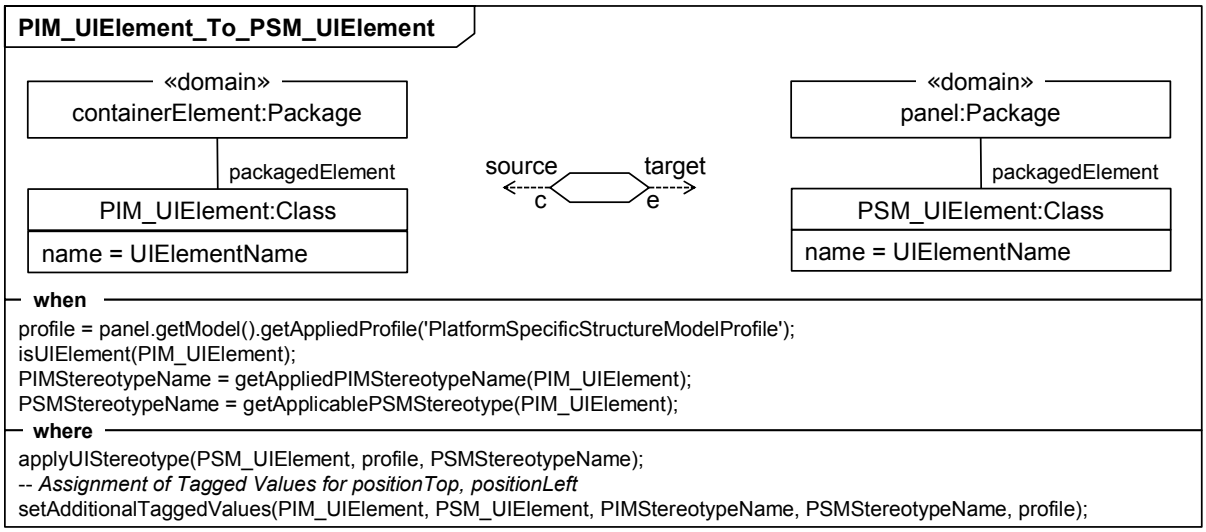

Abbildung 101: PIM_UIElement_To_PSM_UIElement-Relation in QVT-Relations

C. Abkürzungsverzeichnis

ASP.NET	*Active Server Pages* .NET
B2B	*Business to Business*
BAP	Benutzeraufgabenprofil
BDD	*Business-Driven Development* (dt. geschäftsgetriebene Entwicklung)
BPEL	*Business Process Execution Language*
BPMN	*Business Process Modeling Notation*
C&M	Cooperation & Management
CIM	*Computational-Independent Model* (dt. berechnungsunabhängiges Modell)
CRM	*Customer Relationship Management* (dt. Kundenbeziehungsmanagement)
CSS	*Cascading Style Sheets*
DSL	*Domain Specific Language* (dt. domänenspezifische Sprache)
EAI	*Enterprise Application Integration*
EMF	*Eclipse Modeling Framework*
ERP	*Enterprise-Resource Planning*
ESB	*Enterprise Service Bus*
GI	Gesellschaft für Informatik e.V.
GMF	*Graphical Modeling Framework*
GP	*Generative Programming* (dt. generative Programmierung)
GUI	*Graphical User Interface* (dt. grafische Benutzerschnittstelle)
HTTP	*Hyper Text Transfer Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
ISO	*International Organization for Standardization*
IT	Informationstechnologie (engl. *Information Technology*)
JEE	*Java Platform, Enterprise Edition*
KIM	Karlsruher Integriertes InformationsManagement
LDAP	*Lightweight Directory Access Protocol*
MDA	*Model-Driven Architecture* (dt. modellgetriebene Architektur)

MDSD	*Model-Driven Software Development*
MOF	*Meta Object Facility*
OASIS	*Organization for the Advancement of Structured Information Standards*
OAW	Open Architecture Ware
OCL	*Object Constraint Language*
OMG	*Object Management Group*
OSI	*Open Systems Interconnection*
PIM	*Platform-Independent Model* (dt. plattformunabhängiges Modell)
PM	*Platform Model* (dt. Plattformmodell)
PSI	*Platform-Specific Implementation* (dt. plattformspezifische Implementierung)
PSM	*Platform-Specific Model* (dt. plattformspezifisches Modell)
QVT	*Query, View, Transformation*
RSA	Rational Software Architect
RUP	*Rational Unified Process*
SOA	*Service-Oriented Architecture* (dt. dienstorientierte Architektur)
SOAP	früher: *Simple Object Access Protocol*
SPC	*Service Providing Component* (dt. diensterbringende Komponente)
SQL	*Structured Query Language*
UCD	*User-Centered Design* (dt. benutzerzentrierter Entwurf)
UDDI	*Universal Description, Discovery and Integration*
UML	*Unified Modeling Language*
UP	Unified Process
W3C	*World Wide Web Consortium*
WAS	WebSphere Application Server
WID	WebSphere Integration Developer
WPS	WebSphere Process Server
WFM	Workflow-Management
WfMC	*Workflow Management Coalition*
WSDL	*Web Services Description Language*
WSFL	*Web Service Flow Language*
WSOA	Webservice-orientierte Architektur
WSRP	*Web Services for Remote Portlets*
WS-BPEL	*Web Service Business Process Execution Language*
WWW	*World Wide Web*

XMI	*XML Metadata Interchange*
XML	*Extensible Markup Language*
XPDL	*XML Process Definition Language*
XSD	*XML Schema Definition*
XSL	*Extensible Stylesheet Language*
XSLT	*XSL Transformation*

D. Abbildungsverzeichnis

Abbildung 1: Betrachtetes Szenario ... 3
Abbildung 2: Beiträge der Arbeit im Überblick ... 6
Abbildung 3: Gliederung der Arbeit im Überblick ... 9
Abbildung 4: Begriffsbildung Metamodelle und domänenspezifische Sprachen ... 13
Abbildung 5: Funktionsweise von Modell-zu-Modell-Transformationen ... 16
Abbildung 6: Zusammenhang der Sprachteile der QVT-Spezifikation ... 18
Abbildung 7: Abstraktionsebenen der modellgetriebenen Architektur ... 20
Abbildung 8: Die Diagrammtypen der Unified Modeling Language ... 22
Abbildung 9: Modellierungshierarchie der Unified Modeling Language ... 24
Abbildung 10: Dreidimensionale Sicht eines Workflows ... 26
Abbildung 11: Abstrakte Syntax eines Geschäftsprozesses ... 28
Abbildung 12: Referenzarchitektur eines Workflow-Management-Systems ... 29
Abbildung 13: Konkretes Beispiel für die Visualisierung einer Aufgabenliste ... 31
Abbildung 14: Lebenszyklus einer Benutzeraufgabe ... 32
Abbildung 15: Logische Struktur des Service Solution Stack ... 44
Abbildung 16: Logische Struktur der dienstorientierten Architektur von C&M ... 45
Abbildung 17: Erweiterte Portalarchitektur von WSRP ... 48
Abbildung 18: Stereotypisiertes UML-Aktivitätsdiagramm ... 56
Abbildung 19: Benutzerschnittstellendiagramm nach UMLi ... 58
Abbildung 20: Beispiel eines entwurfsorientierten UML-Aktivitätsdiagramms ... 59
Abbildung 21: Erweitertes Lean Cuisine+-Diagramm zu Anwendungsfall „Suche nach Buch“ ... 60
Abbildung 22: Struktur- und Navigationsmodell des Wisdom-Ansatzes ... 62
Abbildung 23: Unterstützte Modelle der Benutzerinteraktions-Entwicklung ... 64
Abbildung 24: WebML Kompositions- und Navigationsmodellierung ... 66
Abbildung 25: UWE – Erweitertes Navigationsmodell ... 69
Abbildung 26: Phasen des DSL-basierten Entwicklungsansatzes ... 72
Abbildung 27: Erweiterte dienstorientierte Architektur ... 74
Abbildung 28: Schematischer Aufbau der Human Workflow Architecture ... 75
Abbildung 29: Modelle und verfolgtes Entwicklungsvorgehen im Überblick ... 80
Abbildung 30: Abstraktes Beispiel eines Geschäftsprozesses ... 81
Abbildung 31: Konzeptionelles Metamodell für Geschäftsprozesse ... 83
Abbildung 32: Stereotypen des Benutzeraktionsprofils ... 86
Abbildung 33: Synchrone und asynchrone Operationsaufrufe einer Benutzeraktion ... 88
Abbildung 34: Erweiterung der Spezifikation des Geschäftsprozessmodells ... 89
Abbildung 35: Entwurf des konzeptionellen Metamodells für Benutzeraufgaben ... 91
Abbildung 36: Metamodell zur Spezifikation von Benutzeraufgaben ... 92
Abbildung 37: Verhalten einer Benutzeraufgabeninstanz ... 94
Abbildung 38: Verhalten einer Systemaufgabe ... 96
Abbildung 39: Stereotypen des Benutzeraufgabenprofils ... 101
Abbildung 40: Vereinfachte Zuordnung von Indikatoren und Eskalationen ... 102
Abbildung 41: Spezifikation einer Benutzeraufgabe als UML-Anwendungsfalldiagramm ... 105
Abbildung 42: Geschäftsobjekte als Grundlage der Benutzerschnittstelle ... 107
Abbildung 43: Konzeptionelles Metamodell des benutzerzentrischen Domänenmodells ... 108
Abbildung 44: Entwurf des konzeptionellen Metamodells des Strukturmodells ... 111
Abbildung 45: Metamodell des Strukturmodells für Benutzerschnittstellen ... 112
Abbildung 46: Stereotypen des benutzerzentrischen Domänenprofils ... 115

Abbildung 47: Stereotypen des Strukturprofils 116
Abbildung 48: Abstraktes Beispiel zur Anwendung der Schnittstellen-Stereotypen 119
Abbildung 49: Verfeinerung des benutzerzentrischen Domänenmodells 119
Abbildung 50: Abbildung des benutzerzentrischen Domänenmodells auf das Strukturmodell 120
Abbildung 51: Konzeptionelles Metamodell zur Spezifikation von Diensten 122
Abbildung 52: Grundkonzept der Ableitung der Dienstnachrichten 123
Abbildung 53: Stereotypen des Dienstprofils 124
Abbildung 54: Beispielhafte Anwendung des Dienstprofils 126
Abbildung 55: Entwickelte Transformationen im Überblick 130
Abbildung 56: Grafische Notation einer Relation in QVT-Relations 132
Abbildung 57: Entwurf der Relation R1.2 der Transformation T1 133
Abbildung 58: Model_To_Model-Relation in QVT-Relations 135
Abbildung 59: UserAction_To_UserTask-Relation in QVT-Relations 135
Abbildung 60: Ergebnis der Anwendung der Transformation T1 136
Abbildung 61: Entwurf der Relationen R2.1 und R2.2 der Transformation T2 137
Abbildung 62: InputPin_To_UserOutputInterface-Relation in QVT-Relations 139
Abbildung 63: OutputPin_To_UserInputInterface-Relation in QVT-Relations 140
Abbildung 64: Ergebnis der Anwendung von Transformation T2 140
Abbildung 65: Entwurf der Relation R3.1 und R3.2 der Transformation T3 142
Abbildung 66: Model_To_Package-Relation in QVT-Relations 143
Abbildung 67: UserInputInterface_To_InputElement-Relation in QVT-Relations 144
Abbildung 68: UserOutputInterface_To_OutputElement-Relation in QVT-Relations 145
Abbildung 69: Ergebnis der Anwendung der Transformation T3 146
Abbildung 70: Kernkomponenten einer dienstorientierten Architektur 147
Abbildung 71: Erweiterte dienstorientierte Architektur 149
Abbildung 72: Spezifikationsmöglichkeiten einer Benutzeraufgabe 151
Abbildung 73: Kommunikationsbeziehungen in der erweiterten dienstorientierten Architektur 152
Abbildung 74: Verwendetes Vorgehen zur Erzeugung der Benutzeraufgabenspezifikation 155
Abbildung 75: Entwurf der Relationen der Transformation T4 156
Abbildung 76: UserTask_To_Task-Relation in QVT-Relations 156
Abbildung 77: Escalation_To_Escalation-Relation in QVT-Relations 157
Abbildung 78: Ergebnis der Anwendung von Transformation T4 158
Abbildung 79: Überblick über das Entwicklungsvorgehen 163
Abbildung 80: Benutzeraktionsprofil in Rational Software Architect 164
Abbildung 81: Geschäftsprozess „Prüfungsanmeldung“ 166
Abbildung 82: Geschäftsprozess „Prüfungsanmeldung“ mit stereotypisierten Aktionen 167
Abbildung 83: Auszug des Benutzeraufgabenmodells des Workflows “Prüfungsanmeldung” 169
Abbildung 84: WS-HumanTask-EMF-Modell des Workflows Prüfungsanmeldung 170
Abbildung 85: Alternative Abbildungsbezüge zu IBM-ITEL 172
Abbildung 86: Verfeinerung des benutzerzentrischen Domänenmodells 174
Abbildung 87: Plattformunabhängiges Strukturmodell Choose Examination Date 175
Abbildung 88: WebPart-Modell Choose Examination Date 177
Abbildung 89: Benutzerschnittstelle zu Choose Examination Date im Office SharePoint Server 179
Abbildung 90: Plattformspezifisches Geschäftsprozessmodell im ERP-System 181
Abbildung 91: Architektur des erweiterten ERP-Systems 182
Abbildung 92: Navigationsmodell einer Benutzerschnittstelle 193
Abbildung 93: Beispielhafte Erweiterung eines Strukturmodells 194
Abbildung 94: Relevanter Ausschnitt des Plattformmodells von ASP.NET 203
Abbildung 95: Umsetzung der WebPart-Elemente als UML-Profil in RSA 204

Abbildung 96: Abbildungsbezüge der Modellelemente der Strukturmodelle 205
Abbildung 97: Package_To_Package-Relation in QVT-Relations 206
Abbildung 98: ContainerElement_To_Panel-Relation in QVT-Relations 206
Abbildung 99: OutputElement_To_Label-Relation in QVT-Relations 207
Abbildung 100: Angepasste QVT-Relation InputPin_To_UserOutputInterface 211
Abbildung 101: PIM_UIElement_To_PSM_UIElement-Relation in QVT-Relations 212

E. Tabellenverzeichnis

Tabelle 1: Anforderungskatalog 55
Tabelle 2: Ergebnis der Bewertung der Forschungsansätze nach Kategorie A1 76
Tabelle 3: Ergebnis der Bewertung der Forschungsansätze nach Kategorie A2 77
Tabelle 4: Zuordnung der konzeptionellen Metamodellelemente 86
Tabelle 5: Bewertung des Benutzeraufgabenprofils 188

F. Literaturverzeichnis

[AA+03] R. Allan, C. Awre, M. Baker, A. Fish: Portals and Portlets 2003, Workshop Portals & Portlets, Edinburg, Großbritannien, 2003.

[AA+05] A. Akram, R. Allan, R, Crouchley: WSRP Reincarnation of Service Oriented Architecture, All Hands Meeting (AHM), Nottingham, Großbritannien, 2005.

[AA+07a] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, M. Zeller: WS-BPEL Extension for People (BPEL4People), 2007.

[AA+07b] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, M. Zeller: Web Services Human Task (WS-HumanTask), 2007.

[Aa98] W.M.P. van der Aalst: The Application of Petri Nets to Workflow Management, The Journal of Circuits, Systems and Computers, 1998.

[AB+08] F. Allerding, J. Buck, P. Freudenstein, B. Klosek, T. Höllrigl, W. Juling, B. Keuter, S. Link, F. Majer, A. Maurer, M. Nussbaumer, D. Ried, F. Schell: Integriertes Service-Portal zur Studienassistenz, Jahrestagung der Gesellschaft für Informatik e.V., München, 2008

[AC+04] G. Alonso, F. Casati, H. Kuno, V. Machiraju: Web Services – Concepts, Architectures and Applications, Springer, 2004.

[AC+05] A. Akram, D. Chohan, X.D. Wang, X. Yang, R. Allan: A Service Oriented Architecture for Portals Using Portlets, All Hands Meeting (AHM), Nottingham, Großbritannien, 2005.

[AC+07] N.I. Altintas, S. Cetin, A.H. Dogru: Industrializing Software Development The Factory Automation Way, Trends in Enterprise Application Architecture, Springer, 2007.

[AH+00] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros: Advanced Workflow Patterns, 7th Conference on Cooperative Information Systems (CoopIS), Eilat, Israel, 2000.

[AH+03] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros: Workflow Patterns, Distributed and Parallel Databases, Springer Niederlande, 2003.

[AH02] W.M.P. van der Aalst, K. van Hee: Workflow Management – Models, Methods, and Systems, MIT Press, 2002.

[AH06] A. Arsanjani, K. Holley: The Service Integration Maturity Model – Achieving Flexibility in the Transformation to SOA, IEEE International Conference on Services Computing, 2006.

[AH-WPI] W.M.P. van der Aalst, A. ter Hofstede: The Workflow Patterns Initiative.
URL: http://www.workflowpatterns.com

[AI04a] J.M. Almendros-Jiménez, L. Iribarne: Describing use cases with Activity charts, Metainformatics Symposium (MIS'04), Springer Verlag, 2004.

[AI04b] J.M. Almendros-Jiménez. L. Iribarne: Method based on UML Use Cases for GUI Design, 9th Conference on Software Engineering and Databases, Málaga, Spanien, 2004.

[AI05] J.M. Almendros-Jiménez, L. Iribarne: Designing GUI components from UML Use Cases, 12th IEEE Int. Conf. and Workshop on the Engineering of Computer Based Systems (ECBS'05), IEEE Computer Society Press, 2005.

[AK+05] M. Acharya, A. Kulkarni, R. Kuppili, R. Mani, N. More, S. Narayanan, P. Patel, K.W. Schuelke, S.N. Subramanian: SOA in the Real World – Experiences, International Conference on Service-Oriented Computing (ICSOC), 2005.

[AL+02] S. Abeck, P.C. Lockemann, J. Schiller, J. Seitz: Verteilte Informationssysteme, dpunkt.verlag, 2002.

[AL+05] S. Abeck, S. Link, C. Mayerl, O. Mehl, T. Vogel: A system-supported method to design IT services, IEEE Conference on Integrated Network and System Management (IM), Nizza, Frankreich, 2005.

[AL+07] A. Arsanjani, L. Zhang, M. Ellis, A. Allam, K. Channabasavaiah: A Service-Oriented Reference Architecture, IT Pro, IEEE Computer Society, 2007.

[Al08] T. Allweyer: BPMN Business Process Modeling Notation – Einführung in den Standard für die Geschäftsprozessmodellierung, Books on Demand, 2008.

[Ar04] A. Arsanjani: Service-Oriented modeling and architecture – How to identify, specify, and realize services for your SOA, IBM developerWorks, 2004.

[AS89] M.D. Apperley, R. Spence: Lean Cuisine – A low-fat notation for menus, Interacting with Computers, Band 1, Ausgabe 1, 1989.

[Ba00] H. Balzert: Lehrbuch der Software-Technik – Software-Entwicklung, Spektrum Akademischer Verlag, 2000.

[Ba08] G. Bauer: Architekturen für Web-Anwendungen – Eine praxisbezogene Konstruktions-Systematik, Vieweg + Teubner, 2008.

[Ba96] B. Baumgarten: Petri-Netze: Grundlagen und Anwendungen, Spektrum Akademischer Verlag, Mannheim, 1996

[BB+05] N. Bieberstein, S. Bose, M. Fiammante, K. Jones, R. Shah: Service-Oriented Architecture (SOA) Compass – Business Value, Planning, and Enterprise Roadmap, IBM developerWorks, IBM Press, 2005.

[BB+08] A. Baier, S. Becker, M. Jung, K. Krogmann, C. Röttgers, N. Streekmann, K. Thoms, S. Zschaler: Modellgetriebene Software-Entwicklung, in: Handbuch der Software-Architektur, 2. Auflage, dpunkt-Verlag, 2008.

[BB+96] D. Bäumer, W. R. Bischofberger, H. Lichter, H. Züllighoven: User Interface Prototyping – Concepts, Tools, and Experience, Proceedings of the 18th International Conference on Software Engineering (ICSE), 1996.

[BC+03] M. Brambilla, S. Ceri, S. Comai, P. Fraternali: Specification and Design of Workflow-driven Hypertexts, Journal of Web Engineering, Band 1, Ausgabe 2, Rinton Press, 2003.

[BC+06] M. Brambilla, S. Ceri, P. Fraternali, I. Manolescu: Process modeling in Web applications, ACM Transactions on Software Engineering and Methodology (TOSEM), Band 15, Ausgabe 4, 2006.

[BE+02] H.J. Bullinger, C.T. Eberhardt, T. Gurzki, H. Hinderer: Marktübersicht Portal Software für Business-, Enterprise-Portale und E-Collaboration, Fraunhofer IRB, 2002.

[Be04] F. Bellas: Standards for Second-Generation Portals, IEEE Internet Computing, Band 8, Ausgabe 2, 2004.

[Be08] J. Bernal: Application Architecture for WebSphere – A Practical Approach to Building WebSphere Applications, IBM developerWorks, IBM Press, 2008.

[BF+92] D. de Baar, J.D. Foley, K.E. Mullet: Coupling Application Design and User Interface Design, ACM Conference on Human Factors in Computing Systems, Monterey, USA, 1992.

[BG+07] S. Becker, T. Goldschmidt, H. Groenda, J. Happe, H. Jacobs, C. Janz, K. Jünemann, B. Klatt, C. Köker, H. Koziolek, K. Krogmann, M. Kuperberg, C. Rathfelder, R. Reussner, B. Veliev: Transformationen in der modellgetriebenen Software-Entwicklung, Forschungsbericht, Universität Karlsruhe (TH), 2007.

[BH04] J. Bishop, N. Horspool: Developing principles of GUI programming using views, Technical Symposium on Computer Science Education Archive, Norfolk, USA, 2004.

[BI+06] A. W. Brown, S. Iyengar, S. K. Johnston: A Rational approach to model-driven development, IBM Systems Journal, Ausgabe 45, Nummer 3, 2006.

[Bi06] J. Bishop: Multi-platform User Interface Construction – a Challenge for Software Engineering-in-the-Small, International Conference on Software Engineering (ICSE), 2006.

[BJ+06] M. Broy, M. Jarke, M. Nagl, D. Rombach: Strategische Bedeutung des Software Engineering in Deutschland, Dagstuhl-Manifest, in: Informatik-Spektrum, Ausgabe 3, Springer, 2006.

[BJ04] K. Blankenhorn, M. Jeckle: A UML Profile for GUI Layout, Object-Oriented and Internet-Based Technologies, Springer, 2004.

[BJ05] A. Brown, S.K. Johnston, G. Larsen, J. Palistrant: SOA Development Using the IBM Rational Software Development Platform – A Practical Guide, IBM, 2005.

[Bl91] U. Black: OSI – A Model for Computer Communication Standards, Prentice Hall, 1991.

[BM+04] B. Bauer, J.P. Müller, S. Roser: A Model-Driven Approach to Designing Cross-Enterprise Business Processes, On The Move to Meaningful Internet Systems (OTM), Agia Napa, Zypern, 2004.

[Bo06] M. Bohlen: QVT und Mulit-Metamodelltransformationen in MDA, OBJEKTSpektrum – Die Zeitschrift für Software-Engineering und Management, Ausgabe 2, 2006.

[Bo07] U. B. Boddenberg: Office SharePoint Server 2007 und Windows SharePoint Services 3.0 – Das Lösungsbuch für Administratoren und Entwickler, Galileo Press, 2007.

[Br06] M. Brambilla: Generation of WebML Web Application Models from Business Process Specifications, International Conference on Web Engineering (ICWE), Palo Alto, USA, 2006.

[BS05] I. Braun, A. Schill: A Service-Oriented Architecture for Teleworking Applications, IASTED International Conference on Internet and Multimedia Systems, and Applications (IMSA), Honolulu, Hawaii, USA, 2005.

[BV96] F. Bodart, J. Vanderdonckt: Widget Standardisation Through Abstract Interaction Objects, Advances in Applied Ergonomics, Istanbul, Türkei, 1996.

[Ca05] B. Castle: Introduction to Web Services for Remote Portlets, IBM developerWorks, IBM Press, 2005.

[CB+04] J. Cardoso, R.P. Bostrom, A. Sheth: Workflow Management Systems and ERP Systems – Differences, Commonalities, and Applications, Information Technology and Management, Ausgabe 5, Springer, 2004.

[CB+06] P. Chowdhary, K. Bhaskaran, N. S. Caswell, H. Chang, T. Chao, S.-K. Chen, M. Dikun, H. Lei, J.-J. Jeng, S. Kapoor, C. A. Lang, G. Mihaila, I. Stanoi, L. Zeng: Model Driven Development for Business Performance Management, IBM Systems Journal, Band 45, Ausgabe 3, 2006.

[CC03] J. Coronado, B. Casey: A Multicultural Approach to Task Analysis – Capturing User Requirements for a Global Software Application, in: D. Diper, N. Stanton (Hrsg.): The Handbook of Task Analysis for Human Computer Interaction, Lawrence Erlbaum Associates, 2003.

[CE00] K. Czarnecki, U. W. Eisenecker: Generative Programming, Methods, Tools and Applications, Addison-Wesley, 2000.

[CF+00] S. Ceri, P. Fraternali, A. Bongio: Web Modeling Language (WebML) – a modeling language for designing Web sites, Computer Networks, Ausgabe 33, 2000.

[CF+02] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, M. Materna: Designing Data-Intensive Web Applications, Morgan Kaufmann, 2002.

[CH+06] S. Conrad, W. Hasselbring, A. Koschel, R. Tritsch: Enterprise Application Integration – Grundlagen - Konzepte - Entwurfsmuster - Praxisbeispiele, Spektrum Akademischer Verlag, 2006.

[Ch04] D. A. Chappell: Enterprise Service Bus, O'Reilly, 2004.

[CH06] K. Czarnecki, S. Helsen: Feature-based survey of model transformation approaches, IBM Systems Journal, Band 45, Nummer 3, 2006.

[Cl88] J.C. Cleaveland: Building application generators, IEEE Software, Band 5, Ausgabe 4, 1988.

[CN05a] P.F. Campos, N.J. Nunes: CanonSketch – A User-Centered Tool for Canonical Abstract Prototyping, International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS), 2005.

[CN05b] P.F. Campos, N.J. Nunes: A UML-Based Tool for Designing User Interfaces, UML Modeling Languages an Applications, Springer, 2005.

[CN07] P. Campos, N. Nunes: Towards useful and usable interaction design tools – CanonSketch, Interacting with Computers, Ausgabe 19, 2007.

[Co03] L. Constantine: Canonical Abstract Prototypes for abstract visual and interaction design, 10th International Conference on Design, Specification and Verification of Interactive Systems (DSV-IS), Madeira, Portugal, 2003.

[Co06] L. Constantine: Users, Roles, and Personas, in: J. Pruitt, T. Adlin (Hrsg.): The Persona Lifecycle, Keeping People in Mind Throughout Product Design, Morgan Kaufmann, 2006.

[CS04] D. Cohn, M. Stolze: The Rise of the Model-Driven Enterprise, International Conference on E-Commerce Technology for Dynamic E-Business, Peking, China, 2004.

[Da93] T. Davenport: Process Innovation: Reengineering Work through Information Technology, Harvard Business Press, 1993.

[Di82] E. W. Dijkstra: On the role of scientific thought, Selected Writings on Computing – A Personal Perspective, Spinger, 1982.

[DJ+05] W. Dostal, M. Jeckle, I. Melzer, B. Zengler: Service-orientierte Architekturen mit Web Services – Konzepte, Standards, Praxis, Spektrum Akademischer Verlag, 2005.

[DK+00] A. van Deursen, P. Klint, J. Visser: Domain-specific languages – an annotated bibliography, ACM SIGPLAN Notices, Ausgabe 35, Nummer 6, 2000.

[DR04] O. Diaz, J. J. Rodriguez: Portlets as Web Components – an Introduction, Journal of Universal Computer Science, Band 10, Nummer 4, 2004.

[Du06] Dudenredaktion: Die deutsche Rechtschreibung – Das umfassende Standardwerk auf der Grundlage der neuen amtlichen Regeln (August 2006), Band 1, 24. Auflage, Bibliographisches Institut, 2006.

[EF-Eclipse] Produktwebseite der Eclipse Foundation: Eclipse
URL: http://www.eclipse.org/

[EF-EMF] The Ecplise Foundation: Eclipse Modeling Framework (EMF)
URL: http://www.eclipse.org/modeling/emf/

[EF-GMF] The Ecplise Foundation: Eclipse Graphical Modeling Framework (GMF)
URL: http://www.eclipse.org/modeling/gmf/

[EF-XSD 2ECORE] Eclipse Foundation: XML Schema to Ecore Mapping
URL: http://www.eclipse.org/modeling/emf/docs/overviews/XMLSchemaToEcoreMapping.pdf

[Ei07] K. Eilebrecht: Patterns kompakt : Entwurfsmuster für effektive Softwareentwicklung, Heidelberg, Spektrum Akademischer Verlag, 2007

[EK+07] C. Emig, K. Krutz, S. Link, C. Momm, S. Abeck: Model-Driven Development of SOA Services, Forschungsbericht, Universität Karlsruhe (TH), 2007.

[EL+06] C. Emig, K. Langer, K. Krutz, S. Link, C. Momm, S. Abeck: The SOA's Layers, Forschungsbericht, Universität Karlsruhe (TH), 2006.

[Em08] C. Emig: Zugriffskontrolle in dienstorientierten Architekturen – Absicherung von Webservices und Webservice-Kompositionen, Verlag Günther Emig, 2008.

[Er05] T. Erl: Service-Oriented Architecture – Concepts, Technology, and Design, Prentice Hall, 2005.

[EW+06] C. Emig, J. Weisser, S. Abeck: Development of SOA-Based Software Systems – an Evolutionary Programming Approach, IEEE Conference on Internet and Web Applications and Services (ICIW), Guadeloupe, Frankreich, 2006.

[FB+07] P. Freudenstein, J. Buck, M. Nussbaumer, M. Gaedke: Model-driven Construction of Workflow-based Web Applications with Domain-specific Languages, 3rd International Workshop on Model-Driven Web Engineering (MDWE), Como Italy, 2007.

[FH+09] M. Funk, P. Hoyer, S. Link: Model-Driven Instrumentation of Graphical User Interfaces, IEEE Conference on Advances in Computer-Human Interaction (ACHI), Cancun, Mexico, Februar 2009.

[FN+08] P. Freudenstein, M. Nussbaumer: Constructing Advanced Web-Based Dialog Components with Stakeholders - A DSL Approach, 8th International Conference On Web Engineering, New York USA, 2008.

[Fo04] M. Fowler: UML konzentriert – Eine kompakte Einführung in die Standard-Objektmodellierungssprache, Addison-Wesley, 2004.

[FR07] R. France, B. Rumpe: Model-driven Development of Complex Software – A Research Roadmap, 29th International Conference on Future of Software Engineering, Minneapolis, USA, 2007.

[GB+01] T. Griffiths, P.J. Barclay, N.W. Paton, J. McKirdy, J.B. Kennedy, P.D. Gray, R. Cooper, C.A. Goble, P. Pinheiro da Silva: Teallach – A model-based user interface development environment for object databases, Interacting with Computers, Ausgabe 14, 2001.

[Ge09] P. Gerr: Modellgetriebene Entwicklung dienstorientierter Benutzerschnittstellen, Diplomarbeit, Universität Karlsruhe (TH), C&M (Prof. Abeck), 2009.

[GI-IB] Gesellschaft für Informatik: Informatik-Begriffsnetz
URL: http://public.tfh-berlin.de/~giak/

[GK+03] K. Geihs, R. Kalckloesch, A. Grode: Single Sign-On in Service-Oriented Computing, International Conference on Service-Oriented Computing (ICSOC), 2003.

[GK05] M. Großmann, H. Koschek: Unternehmensportale – Grundlagen, Architekturen, Technologien, Springer, 2005.

[Gl00] P.A. Gloor: Making the e-Business Transformation, Springer, 2000.

[GN+04] M. Gaedke, M. Nussbaumer, J. Meinecke: WSLS - An Agile System Facilitating the Production of Service-Oriented Web Applications, in Engineering Advanced Web Applications, Rinton Press, 2004.

[Go95] G. Goos: Vorlesungen über Informatik Band 1 – Grundlagen und funktionales Programmieren, Springer, 1995.

[GS03] J. Greenfield, K. Short: Software Factories - Assembling Applications with Patterns, Models, Frameworks and Tools, 18th Conference on Object Oriented Programming Systems Languages and Applications, Anaheim, USA, 2003.

[Gu06] T. Gurzki: Ein Architekturmodell für Portale im Technischen Vertrieb, Dissertation an der Universität Stuttgart, Jost-Vettler Verlag, 2006.

[HA+97] M. van Harmelen, J. Artim, K. Butler, A. Henderson, D. Roberts, M. B. Rosson, J.-C. Tarby, S.Wilson: Object Models in User Interface Design. ACM SIGCHI Bulletin, Band 29, Nummer 4, 1997.

[Ha05] M. Harvey: Essential Business Process Modeling, O'Reily, 2005.

[HC93] M. Hammer, J. Champy: Reengineering the Corporation – A Manifesto for Business Revolution, HarperBusiness, 1993.

[He05] M. Hebach: Mit der QVT wird MDA erst schön, Objektspektrum Model-Driven Development, 2005.

[HH+06] A. Hess, B. Humm, M. Voß: Regeln für serviceorientierte Architekturen hoher Qualität, Informatik Spektrum, Band 29, Nummer 6, Springer, 2006.

[HO07] A. Heinzl, A. Oberweis: Industrialisierung der Softwareentwicklung – Eine Bestandsaufnahme, in: Wirtschaftsinformatik, Band 49, Ausgabe 3, Springer, 2007.

[Ho08a] P. Horváth: Controlling, 11. Auflage, Vahlen, 2008.

[Ho08b] P.A. Hoyer: Modellgetriebene Integration von Benutzeraufgaben in automatisierte Geschäftsprozesse, Diplomarbeit, Universität Karlsruhe (TH), C&M (Prof. Abeck), 2008.

[HT06] B. Hailpern, P. Tarr: Model-driven development – The good, the bad, and the ugly, IBM Systems Journal, Ausgabe 45, Nummer 3, 2006.

[IBM-RSA] IBM-Produktwebseite: Rational Software Architect (RSA).
URL: http://www-306.ibm.com/software/awdtools/swarchitect

[IBM-WPoS] IBM-Produktwebseite: WebSphere Portal Server (WPoS).
URL: http://www-01.ibm.com/software/websphere/portal/

[IBM-WPS] IBM-Produktwebseite: WebSphere Process Server (WPS).
URL: http://www-01.ibm.com/software/integration/wps/

[IBM-WSFL] IBM: Web Services Flow Language (WSFL 1.0). IBM Software Group, Mai 2001. http://xml.coverpages.org/WSFL-Guide-200110.pdf

[IKV-Medini] Produktwebseite der ikv++ technologies ag: Medini QVT
URL: http://projects.ikv.de/qvt/

[IS+95] T. Isakowitz, E. Stohr, P. Balasubramanian: A Methodology for the Design of Structured Hypermedia Applications, Communications of the ACM, 1995

[Ja08] C. Janz: Modellgetriebene Entwicklung von Geschäftsprozessen in Enterprise Resource Planning Systemen, Diplomarbeit, Universität Karlsruhe (TH), C&M (Prof. Abeck), 2008.

[Ja92] I. Jacobson: Object-oriented software engineering – a use case driven approach, Addison-Wesley Professional, 1992.

[JB+99] I. Jacobson, G. Booch, J. Rumbaugh: The Unified Software Development Process, Addison Wesley, 1999.

[JB06] S.K. Johnson, A.W. Brown: A Model-Driven Development Approach to Creating Service-Oriented Solutions, International Conference on Service-Oriented Computing (ICSOC), Chicago, USA, 2006.

[JCP-JSR168] Java Community Process: Java Specification Request 168 – Java Portlet Specification Version 1.0
URL: http://www.jcp.org/en/jsr/detail?id=168

[JCP-JSR286] Java Community Process: Java Specification Request 286 – Java Portlet Specification Version 2.0
URL: http://www.jcp.org/en/jsr/detail?id=286

[Ji09] W. Ji: Metamodell-basierte Transformation von Benutzeraufgabenmodellen, Studienarbeit, Universität Karlsruhe (TH), C&M (Prof. Abeck), 2009.

[Jo08] N. Josuttis: SOA in der Praxis, System-Design für verteilte Geschäftsprozesse, dpunkt.verlag, 2008.

[KA+04] M. Keen, A. Acharya, S. Bishop, A. Hopkins, S. Milinski, C. Nott, R. Robinson, J. Adams, P. Verschueren: Patterns – Implementing an SOA Using an Enterprise Service Bus, IBM Reedbooks, 2004.

[KB+05] D. Krafzig, K. Banke, D. Slama: Enterprise SOA – Serivce-Oriented Architecture Best Practices, Prentice Hall, 2005.

[KB+07] M. Keen, O. Bahy, W. Croson, A. Garratt, B. Karchner, I. Lehmann, F. Neumann, L. Roach: Human-Centric Business Process Management with WebSphere Process Server V6, IBM Redbook, 2007.

[Ke06] C. Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.

[KF93] W.C. Kim, J.D. Foley: Providing High-level Control and Expert Assitance in the User Interface Presentation Design, ACM Conference on Human Factors in Computing Systems, Amsterdam, Niederlande, 1993.

[KG+07] J. Koehler, T. Gschwind, J. Küster, C. Pautasso, K. Ryndina, J. Vanhatalo, H. Völzer: Combining Quality Assurance and Model Transformations in Business-Driven Development, 3rd International Workshop and Symposium on Applications of Graph Transformation with Industrial Relevance (AGTIVE), Kassel, 2007

[KH+08] J. Koehler, R. Hauser, J. Küster, K. Ryndina, J. Vanhatalo, M. Wahler: The Role of Visual Modeling and Model Transformations in Business-driven Development, Electronic Notes in Theoretical Computer Science, Band 211, 2008.

[KK+01] N. Koch, A. Kraus, R. Hennicker: The Authoring Process of the UML-based Web Engineering Approach, 3rd international Workshop on Web-oriented Software Technology (IWWOST), Oviedo, Asurias, 2003.

[KK+03] N. Koch, A. Kraus, C. Cachero, S. Melia: Modeling Web Business Processes with OO-H and UWE, 3rd International Worskhop on Web-oriented Software Technology (IWWOST´03), Oviedo, Spanien, 2003.

[KK+05] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, I. Trickovic: WS-BPEL Extension for People (BPEL4People), Joint White Paper by IBM and SAP, 2005.

[KK+07a] A. Knapp, N. Koch, M. Wirsing, G. Zhang: UWE – Approach to Model-Driven Development of Web Applications, i-com – Zeitschrift für interaktive und kooperative Medien, Ausgabe 3, Oldenbourg Wissenschaftsverlag, 2007.

[KK+07b] A. Kraus, A. Knapp N. Koch: Model-Driven Generation of Web Applications in UWE, 3rd International Workshop on Model-Driven Web Engineering, 2007.

[KK02] N. Koch, A. Kraus: The Expressive Power of UML-based Web Engineering, 2nd International Worskhop on Web-oriented Software Technology (IWWOST´02), Malaga Spanien, 2002.

[KK08] C. Kroiß, N. Koch: UWE Metamodel and Profile – User Guide and Reference, LMU Technical Report 0801, 2008.

[Ko01] N. Koch: Software Engineering for Adaptive Hypermedia Systems Reference Model, Modeling Techniques and Development Process, Dissertation an der Ludwig-Maximilians-Universität München, 2001.

[Ko06] N. Koch: Transformation Techniques in the Model-Driven Development Process of UWE, International Conference on Web Engineering (ICWE) Workshops, Menlo Park California USA, 2006.

[Kr03] P. Kruchten: The Rational Unified Process An Introduction Third Edition, Addison-Wesley, 2003.

[KR05] P. Kroll, W. Royce: Key principles for business-driven development, IBM developerWorks, 2005.

[Kr07] K. Krutz: Integriertes Modell zur Entwicklung und Suche von Web-Diensten, Dissertation an der Universität Karlsruhe (TH), dissertation.de, 2007.

[KV06] A. Kalnins, V. Vitolins: Use of UML and Model Transformations for Workflow Process Definitions, 7th International Baltic Conference on Databases and Information, 2006.

[KW+03] A. Kleppe, J. Warmer, W. Bast: MDA Explained – The Model Driven Architecture: Practice and Promise, Addison-Wesley, 2003.

[LC+03] K. Luyten, T. Clerckx, K. Coninx, J. Vanderdonckt: Derivation of a Dialog Model from a Task Model by Activity Chain Extraction, Interactive Systems - Design, Specification, and Verification, Springer, 2003.

[Le01] F. Leymann: Web Services for Business Process Design, Microsoft Corporation, 2001.

[Le03] F. Leymann: Web Services – Distributed Applications without Limits, Database Systems for Business, Technology and Web (BTW), Leipzig, 2003.

[LH+08] S. Link, P. Hoyer, T. Schuster, S. Abeck: Model-Driven Development of Human Tasks for Business Processes, IEEE Conference on Software Engineering Advances (ICSEA), Sliema Malta, 2008.

[LH+09] S. Link, P. Hoyer, T. Kopp, S. Abeck: A Model-Driven Development Approach Focusing Human Interaction, IEEE Conference on Advances in Computer-Human Interaction (ACHI), Cancun Mexico, 2009.

[Li07] D. Liebhart: SOA goes real – Service-orientierte Architekturen erfolgreich planen und einführen, Hanser, 2007.

[LJ+08] S. Link, C. Janz, M. Schober, S. Abeck: Modellgetriebene Geschäftsprozess-Entwicklung in ERP-Systemen, ERP-Management, GITO-Verlag, Oktober 2008.

[Lo06] A. Lorenz: Anpassung von UML-Aktivitäten an den Prozess der Webapplikationsentwicklung, 36. Jahrestagung der Gesellschaft für Informatik, Bonner Köllen Verlag, 2006.

[LR+02] F. Leymann, D. Roller, M.T. Schmidt: Web services and business process management, IBM Systems Journal, Band 41, Nummer 2, 2002.

[LS+07a] S. Link, T. Schuster, P. Hoyer, S. Abeck: Modellgetriebene Entwicklung grafischer Benutzerschnittstellen, i-com – Zeitschrift für interaktive und kooperative Medien, Oldenbourg Wissenschaftsverlag, 2007.

[LS+07b] S. Link, T. Schuster, P. Hoyer, S. Abeck: Modellgetriebene Entwicklung von grafischen Benutzerschnittstellen, Informatik trifft Logistik, Jahrestagung der Gesellschaft für Informatik e.V., 2007.

[LS+08] S. Link, T. Schuster, P. Hoyer, S. Abeck: Focusing Graphical User Interfaces in Model-Driven Software Development, IEEE Conference on Advances in Computer Human Interaction (ACHI), Saint Luce, Martinique, 2008.

[LV+05] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, V. López-Jaquero: USIXML - A Language Supporting Multi-path Development of User Interfaces, 9th IFIP Working Conference on Engineering for Human-Computer Interaction, Hamburg, 2004.

[MA+06] J. Martínez-Ruiz, J. M. Arteaga, J. Vanderdonckt, J. M. González-Calleros: A First Draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications, 4th Latin American Web Congress, IEEE Computer Society Press, 2006.

[Ma01] C. Mayerl: Eine integrierte Dienstmanagement-Architektur für die qualitätsgesicherte Bereitstellung von Netz- und Systemdiensten, Dissertation an der Universität Karlsruhe (TH), Shaker-Verlag, 2001.

[Ma05] A. Maurer: Karlsruher Integriertes InformationsManagement, UniKATH, Februar 2005

[MD+01] S. Murugesan, Y. Deshpande, S. Hansen, A. Ginige: Web Engineering – A New Discipline for Development of Web-Based Systems, in: S. Murugesan, Y. Deshpande: Web Engineering – Managing Diversity and Complexity of Web Application Development, Springer, 2001.

[Me07] O. Mehl: Modellgetriebene Entwicklung managementfähiger Anwendungssysteme, Dissertation, dissertation.de, 2007.

[MF+07] N. Moreno, P. Fraternali, A. Vallecillo: WebML modelling in UML, IET Software, Band 1, Ausgabe 3, 2007.

[Mi05] T. Mitra: Business-driven development, IBM developerWorks, 2005.
URL: http://www-128.ibm.com/developerworks/webservices/library/ws-bdd/

[Microsoft-WSRP] Microsoft Produktwebseite: WSRP Toolkit for SharePoint 2007
URL: http://code.msdn.microsoft.com/WSRPToolkit

[MK+00] B. Myers, S.E. Hudson, R. Pausch: Past, Present, and Future of User Interface Software Tools, ACM Transactions on Computer-Human Interaction 7, 2000.

[MM03] J. Miller, J. Mukerji: MDA Guide Version 1.0. 1, Object Management Group, 2003.

[Mo09] C. Momm: Modellgetriebene Entwicklung überwachter Webservice-Kompositionen, Dissertation, Universität Karlsruhe (TH), Shaker Verlag, 2009.

[NF+06a] M. Nussbaumer, P. Freudenstein, M. Gaedke: Stakeholder Collaboration - From Conversation To Contribution, 6. International Conference on Web Engineering (ICWE), Menlo Park California, USA, 2006.

[NF+06b] M. Nussbaumer, P. Freudenstein, M. Gaedke: Towards DSL-based Web Engineering, 15th International World Wide Web Conference (WWW), Edinburgh, UK, 2006.

[NF00] N.J. Nunes, J. Falcao e Cunha: Towards a UML Profile for Interaction Design: the Wisdom Approach, in: A. Evans, S. Kent, B. Selic (Hrsg.): UML 2000, Springer, 2000.

[NF01] N.J. Nunes, J. Falcao e Cunha: Wisdom – A UML Based Architecture for Interactive Systems, Lecture Notes in Computer Science, Springer, 2000

[NT+99] N.J. Nunes, M. Toranzo, J. Falcao e Cunha, J. Castro, S. Kovacevic, D. Roberts, J. Tarby, M. Collins-Cope, M. van Harmelen: Interactive System Design with Object Models (WISDOM'99), in: A. Moreira, S. Demeyer (Hrsg.): ECOOP'99 Workshop Reader, Springer, 1999.

[OASIS-SOA-RMCD] Organization for the Advancement of Structured Information Standards: Reference Model for Service Oriented Architecture, Version 1.0, Committee Draft.
URL: http://www.oasis-open.org/committees/download.php/16587/wd-soa-rm-cd1ED.pdf

[OASIS-UDDI2] Organization for the Advancement of Structured Information Standards: Universal Description, Discovery and Integration (UDDI) Version 2.0 API Specification, OASIS Standard.
http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf

[OASIS-WS-BPEL2] Organization for the Advancement of Structured Information Standards: Web Services Business Process Execution Language Version 2.0, OASIS Standard.
URL: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

[OASIS-WSRP2] Organization for the Advancement of Structured Information Standards: Web Services Remote Portlets Specification Version 2.0, OASIS Standard
URL: http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec-os-01.pdf

[OAW] Produktwebseite Open Architecture Ware
URL: http://www.openarchitectureware.org/

[Oe06] B. Oestereich: Analyse und Design mit UML 2.1 – Objektorientierte Softwareentwicklung, Oldenbourg, 2006.

[OMG-BPDM1.0] Object Management Group: Business Process Definition Metamodel, Version 1.0
URL: http://www.omg.org/spec/BPDM/1.0/

[OMG-BPMN] Object Management Group: Business Process Modeling Notation, Version 1.2
URL: http://www.omg.org/spec/BPMN/1.2/

[OMG-MDA] Object Management Group: Model Driven Architecture.
URL: http://www.omg.org/mda

[OMG-MOF2] Object Management Group: Meta Object Facility (MOF) Core Specification, Version 2.0.
URL: http://www.omg.org/spec/MOF/2.0

[OMG-OCL2] Object Management Group: Object Constraint Language, Version 2.0.
URL: http://www.omg.org/spec/OCL/2.0

[OMG-QVT] Object Management Group: Meta Object Facility (MOF) Query/View/Transformation Specification, Version 1.0.
URL: http://www.omg.org/spec/QVT/1.0/

[OMG-UML2-Infra] Object Management Group: OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.1.2.
URL: http://www.omg.org/spec/UML/2.1.2/

[OMG-UML2-Super] Object Management Group: OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.1.2.
URL: http://www.omg.org/spec/UML/2.1.2/

[OW+03] B. Oestereich, C. Weiss, C. Schroeder, T. Weilkiens, A. Lenhard: Objektorientierte Geschäftsprozessmodellierung mit der UML, dpunkt.verlag, 2003.

[Pa04] F. Paternò: The ConcurTaskTrees Boundary Object, ISTI C.N.R., Pisa, Italy 2004

[Pe03] C. Peltz: Web Services Orchestration and Choreography, IEEE Computer, Band 36, Ausgabe 10, 2003.

[PM06] R. Petrasch, O. Meimberg: Model Driven Architecture – Eine praxisorientierte Einführung in die MDA, dpunkt.verlag, 2006.

[PP00] P. Pinheiro da Silva, N. W. Paton: User Interface Modelling with UML, Proceedings of 10th European-Japanese Conference on Information Modelling and Knowledge Bases, Saariselk, Finnland, 2000.

[PP01] P. Pinheiro da Silva, N. W. Paton: A UML-Based Design Environment for Interactive Applications, Proceedings of UIDIS'01, IEEE Computer Society, Zürich Schweiz, Mai 2001.

[PP02] P. Pinheiro da Silva, N. W. Paton: Improving UML Support for User Interface Design – A Metric Assessment of UMLi, International Conference on Software Engineering (ICSE), 2002.

[PP03] P. Pinheiro da Silva, N. W. Paton: User Interface Modeling in UMLi, Los Alamitos, USA, IEEE Computer Society Press, Ausgabe Juli/August, 2003.

[PS+08] F. Paternò, C. Santoro, J. Mäntyjärvi, G. Mori, S. Sansone: Authoring pervasive multimodal user interfaces, International Journal on Web Engineering and Technology, Band 4, Nummer 2, 2008.

[PT+07] G. Pietrek, J. Trompeter, B. Niehues, T. Kamann, B. Holzer, M. Kloss, K. Thoms, J. C. Flores Beltran, S. Mork: Modellgetriebene Softwareentwicklung – MDA und MDSD in der Praxis, entwickler.press, 2007.

[Pu04] T. Puschmann: Prozessportale – Architektur zur Vernetzung von Kunden und Lieferanten, Springer, 2004

[Pu07] T. Puschmann: Prozessportale als Grundlage serviceorientierter Architekturen, Industrie Management Ausgabe 23, GITO-Verlag, 2007.

[PW07] D. Pfeiffer, A. Winkelmann: Ansätze zur Wiederverwendung von Software im Rahmen der Softwareindustrialisierung am Beispiel von Softwarekomponenten, serviceorientierten Architekturen und modellgetriebenen Architekturen, in: Wirtschaftsinformatik, Band 49, Ausgabe 3, Springer, 2007.

[QP06] T. Quartani, J. Palistrant: Visual Modeling with IBM Rational software architect and UML, IBM developerWorks, IBM Press, 2006.

[RA+05] N. Russel, W.M.P. van der Aalst, A.H.M. ter Hofstede, D. Edmond: Workflow Resource Patterns – Identification, Representation and Tool Support, 17th Conference on Advanced Information Systems Engineering (CAiSE), Porto, Portugal, 2005.

[RA+06a] N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, P. Wohed: On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling, 3rd Asia-Pacific Conference on Conceptual Modeling (APCCM), Hobart, Australia, 2006.

[RA+06b] N. Russell, W.M.P van der Aalst, A.H.M. ter Hofstede: Workflow exception patterns, 18th Conference on Advanced Information Systems Engineering (CAiSE'06), Luxembourg, Luxembourg, Springer, 2006.

[RA07] N. Russell, W.M.P. van der Aalst: Evaluation of the BPEL4People and WS-HumanTask Extensions to WS-BPEL 2.0 using the Workflow Resource Patterns, BPM Center Report BPM-07-10, BPMcenter.org, 2007.

[Re05] M. Reiter: Infrastruktur für die Serviceorientierung steht, Computer Zeitung Nr. 19, 2005.

[RF+04] D. Reichart, P. Forbrig, A. Dittmar: Task Models as Basis for Requirements Engineering and Software Execution, 3rd International Workshop on Tasks Models and Diagrams (Tamodia), Prag, Tschechien, 2004.

[RF07] M. Richter, M. D. Flückiger: Usability Engineering kompakt – Benutzbare Software gezielt entwickeln, Spektrum Akademischer Verlag, 2007

[RFC-2251] The Internet Engineering Task Force: RFC 2251, Lightweight Directory Access Protocol (v3), 1997.

[RG08] C. Rathfelder, H. Groenda: iSOAMM – An Independent SOA Maturity Model, International Conference on Distributed Applications and Interoperable Systems, Oslo, Norway, 2008.

[RH+04] N. Russell, A.H.M. ter Hofstede, D. Edmond, W.M.P. van der Aalst: Workflow Resource Patterns, BETA Working Paper Series, WP 127, Eindhoven University of Technology, 2004.

[RH+05] N. Russel, A.H.M. ter Hofstede, D. Edmond, W.M.P. van der Aalst: Workflow Data Patterns – Identification, Representation and Tool Support, Conceptual Modeling – ER 2005, Springer, 2005.

[RH+06] N. Russell, Arthur H.M. ter Hofstede, W.M.P. van der Aalst, N. Mulyar: Workflow Control-Flow Patterns – A Revised View, BPM Center Report BPM-06-22, BPMcenter.org, 2006.

[RH06] R. Reussner, W. Hasselbring: Handbuch der Softwarearchitektur, dpunkt.verlag, 2006.

[Ri07] J. Rickern: Top-Down Modeling Methodology for Model-Driven SOA Construction, On The Move to Meaningful Internet Systems (OTM), Vilamoura, Portugal, 2007.

[RM06] J. Rodríguez, J. Mariscal: Exploring Human Workflow Architectures, The Architecture Journal, Microsoft Corporation, Ausgabe 7, 2006.

[Ro03] G. Dreo-Rodosek: A Generic Model for IT Services and Service Management, 8th Symposium on Integrated Network Management, Colorado Springs, USA, 2003.

[RS+00] G. Rossi, D. Schwabe, F. Lyardet: Web Applications Models are More than Conceptual Models, Advances in Conceptual Modelling, Springer Verlag Berlin / Heidelberg, 2000.

[RS04] C. Richter von Hagen, W. Stucky: Business-Process- und Workflow-Management – Prozessverbesserung durch Prozess-Management, Vieweg + Teubner, 2004.

[Rü01] J. Rütschlin: Ein Portal – Was ist das eigentlich?, Wirtschaft und Wissenschaft in der Network Economy – Visionen und Wirklichkeit, GI/OCG-Jahrestagung, Wien, 2001.

[Ry09] J. Rydzy: Automatisierte Erzeugung von Benutzeraufgabenspezifikationen, Studienarbeit, Universität Karlsruhe (TH), C&M (Prof. Abeck), 2009.

[SB+09] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks: EMF: Eclipse Modeling Framework 2. Auflage, Addison-Wesley, 2009.

[SB05] T. Seifert, G. Beneken: Evolution and Maintenance of MDA Applications, in: S. Beydeda, M. Book, V. Gruhn (Hrsg.): Model-Driven Software Development, Springer, 2005.

[SC+06] J. C. Silva, J. C. Campos, J. Saraiva: Models for the Reverse Engineering of Java/Swing Applications, 3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies (ateM), Genua, Italien, 2006.

[SF+93] P. Sukaviriya, J.D. Foley, T. Griffith: A Second Generation User Interface Design Environment - The Model and The Runtime Architecture, INTERACT and CHI conference on Human factors in computing systems, Amsterdam, Niederlande, 1993.

[SG+05] A. Seffah, J. Gulliksen, M.C. Desmarais: Human-Centered Software Engineering – Integrating Usability in the Development Process, Human-Computer Interaction Series, Nummer 8, Springer, 2005.

[SK+05] N. Sukaviriya, S. Kumaran, P. Nandi, T. Heath: Integrate Model-driven UI with Business Transformations – Shifting Focus of Model-driven UI, Workshop on Model Driven Design of Advanced User Interfaces (MDDAUI), Montego Bay, Jamaica, 2005.

[Sm04] M.A. Smith: Portals – toward an application framework for interoperability, Communications of the ACM, Band 47, Ausgabe 10, 2004.

[So06] I. Sommerville: Software Engineering, 8. Ausgabe, Addison Wesley, 2006.

[SP03] C. Scoginigs, C. Phillips: Linking Task and Dialogue Modeling – Toward an Integrated Software Engineering Method, in: D. Diper, N. Stanton (Hrsg.): The Handbook of Task Analysis for Human Computer Interaction, Lawrence Erlbaum Associates, 2003.

[SS+07a] N. Sukaviriya, V. Sinha, T. Ramachandra, S. Mani, M. Stolze: User-Centered Design and Business Process Modeling – Cross Road in Rapid Prototyping Tools, Human-Computer Interaction (INTERACT), Rio de Janeiro, Brasillien, 2007.

[SS+07b] N. Sukaviriya, V. Sinha, T. Ramachandra, S. Mani: Model-Driven Approach for Managing Human Interface Design Life Cycle, International Conference on Model Driven Engineering Languages and Systems (MoDELS), Nashville, USA, 2007.

[St07] T. Stapelkamp: Screen und Interfacedesign – Gestalung und Usability für Hard- und Software, Springer, 2007.

[ST07] G. Starke, S. Tilkov: SOA-Expertenwissen – Methoden, Konzepte und Praxis serviceorientierter Architekturen, dpunkt.verlag, 2007.

[St73] H. Stachowiak: Allgemeine Modelltheorie, Springer Verlag, 1973.

[SV05] T. Stahl, M. Völter: Modelgetriebene Softwareentwicklung – Techniken, Engineering, Management, dpunkt.verlag, 2005.

[Ta05] D. Taubner: Software-Industrialisierung, in: Informatik-Spektrum, Band 28, Ausgabe 4, Springer, 2005.

[TB+07] S. Trujillo, D. Batory, O. Diaz: Feature Oriented Model Driven Development – A Case Study for Portlets, 29th International Conference on Software Engineering (ICSE), 2007

[Tig-AgroUML] Tigris Open Source Software Engineering Tools: Produktwebseite AgroUML
URL: http://argouml.tigris.org/

[TP+07] J. Thomas, F. Paci, E. Bertino, P. Eugster: User Tasks and Access Control over Web Services, IEEE International Conference on Web Services (ICWS), Salt Lake City, USA, 2007.

[Uh06] A. Uhl: Model-Driven Architecture, in: R. Reussner, W. Hasselbring (Hrsg.): Handbuch der Softwarearchitektur, dpunkt.verlag, 2006.

[UKA-KIM] Universität Karlsruhe, W. Juling, H. Hartenstein, A.Maurer: Karlsruher Integriertes InformationsManagement – Webseite des Projekts, 2009.
URL: http://www.kim.uni-karlsruhe.de

[Va05] J. Vanderdonckt: A MDA-Compliant Environment for Developing User Interfaces of Information Systems, 17th Conference on Advanced Information Systems Engineering (CAiSE), Porto, Portugal, 2005.

[Va96] J. Vanderdonckt (Hrsg.): Computer-Aided Design of User Interfaces, Proceedings of CADUI '96, Presses Universitaires de Namur, 1996.

[W3C-CSS2.1] Word Wide Web Consortium (W3C): Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification
URL: http://www.w3.org/Style/CSS/

[W3C-SOAP1.2] World Wide Web Consortium (W3C) Recommendation: SOAP Version 1.2.
URL: http://www.w3.org/TR/soap

[W3C-WSDL1.1] World Wide Web Consortium (W3C) Recommendation: Web Services Description Language (WSDL), Version 1.1.
URL: http://www.w3.org/TR/wsdl

[W3C-XSD] World Wide Web Consortium (W3C) Recommendation: XML Schema Part 1 - Structures Second Edition
URL: http://www.w3.org/TR/xmlschema-1/

[WA+06] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, N. Russell: On the Suitability of BPMN for Business Process Modelling, Business Process Management, Springer, 2006.

[WC+05] S. Weerawarana, F. Curbera, F.Leymann, T. Storey, D.F. Ferguson: Web Services Platform Architecture – SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More, Prentice Hall, 2005.

[WF+05] A. Wolff, P. Forbrig, A. Dittmar, D. Reichart: Linking GUI elements to tasks – supporting an evolutionary design process, 4th International Workshop on Task Models and Diagrams (TAMODIA), Gdansk, Polen, 2005.

[WfMC-T&G3] Workflow Management Coalition: Terminology & Glossary, Version 3.0. http://www.wfmc.org/Download-document/WFMC-TC-1011-Ver-3-Terminology-and-Glossary-English.html

[WfMC-WRM1.1] Workflow Management Coalition: The Workflow Reference Model, Version 1.1. URL: http://www.wfmc.org/Download-document/TC00-1003-The-Workflow-Reference-Model.html

[WfMC-XPDL2.1] Workflow Management Coalition: Process Definition Interface - XML Process Definition Language, Version 2.1. URL: http://www.wfmc.org/xpdl-developers-center.html

[WK03] J.B. Warner, A. Kleppe: The object constraint language: getting your models ready for MDA, Addison-Wesley, 2003.

[WM06] D. Woods, T. Mattern: Enterprise SOA – Designing IT for Business Innovation, O'Reilly, 2006.

[WS06] R. Winter, J. Schelp: Dienstorientierte Architekturgestaltung auf unterschiedlichen Abstraktionsebenen, in: R. Reussner, W. Hasselbring (Hrsg.): Handbuch der Softwarearchitektur, dpunkt.verlag, 2006.

[Zö06] S. Zörner: Portlets – Portalkomponenten in Java, entwickler.press , 2006.

[ZT+05] O. Zimmermann, M. Tomlinson, S. Peuser: Perspectives on Web Services, Springer, 2005.