KomBInoS – Modellgetriebene Entwicklung von multimodalen Dialogschnittstellen für Smart Services

Dissertation
zur Erlangung des Grades des
Doktors der Ingenieurwissenschaften
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

vorgelegt von
Dipl.-Inform. Daniel Porta

Saarbrücken, November 2017

Datum des Kolloquiums:	15. Dezember 2017
Dekan der Fakultät MI:	Prof. Dr. Frank-Olaf Schreyer

Prüfungsausschuss:

Vorsitzender:	Prof. Dr. Jörg Hoffmann
Erstgutachter:	Prof. Dr. Dr. h.c. mult. Wolfgang Wahlster
Zweitgutachter:	Prof. Dr. Wolfgang Maaß
Akademischer Beisitzer:	Dr. Dietmar Dengler

Danksagungen

Diese Arbeit entstand am Deutschen Forschungszentrum für Künstliche Intelligenz im Rahmen des vom Bundesministeriums für Wirtschaft und Energie geförderten THESEUS Forschungsprogramms, insbesondere dem TEXO Anwendungsfall, sowie in den vom Bundesministerium für Bildung und Forschung geförderten Projekten SINNODIUM im Rahmen des Software Clusters, SmartF-IT und dem Software Campus.

Ich freue mich sehr, dass ich mich an dieser Stelle bei den vielen Menschen für die Unterstützung, die ich von ihnen erfahren habe, von Herzen bedanken kann.

Ich möchte mich bei Prof. Wolfgang Wahlster bedanken. Dafür, dass ich in seinem Forschungsbereich an spannenden Projekten mitarbeiten konnte, dass er mich stets in meiner Entwicklung gefördert und mir die Möglichkeit und das Vertrauen entgegen gebracht hat, diese Arbeit zu schreiben. Auch wenn Sie es als Teil Ihrer Aufgabe ansehen, vielen Dank für die konstruktiven Diskussionen in angenehmer Atmosphäre in der Schlussphase der Arbeit, zu denen Sie stets am Wochenende Zeit fanden. Das Resultat wurde dadurch sehr viel besser.

Vielen Dank auch an Prof. Wolfgang Maaß für die spontane Bereitschaft, kurzfristig das Zweitgutachten für diese Arbeit anzufertigen.

Danke an Prof. Alexander Kröner, der mich vor etlichen Jahren zum Nachdenken gebracht und sich für mich eingesetzt hat. Vielen Dank an Dr. Tilman Becker für die intensiven Diskussionen am Anfang meiner Arbeit. Danke an Dr. Daniel Sonntag, von dem ich viel lernen konnte. Vielen Dank auch an Dr. Anselm Blocher für die Unterstützung sowie Sylvia Krüger für das mehrfache akribische Korrekturlesen dieser Arbeit. Einen besonderen Dank möchte ich Gerd Herzog aussprechen. Dafür, dass ich an seinem schier unendlichen Fundus an wissenschaftlicher Weltliteratur partizipieren durfte und für ganz viele andere materielle und immaterielle Kleinigkeiten. Schließlich möchte ich mich herzlichst bei Dr. Dietmar Dengler bedanken. Ich weiß nicht, wo ich anfangen soll und ich befürchte, es sprengt den Rahmen. Darum einfach nur danke!

Vielen Dank an die teils ehemaligen Kollegen, Saunagänger und Mannschaftskameraden, aber ganz sicher Leidensgenossen. Danke an Dr. Jens Haupert, Dr. Jochen Frey, Dr. Robert Neßelrath und Dr. Matthieu Deru für den gegenseitigen Zuspruch, den Ansporn und die Unterstützung. Herzlichen Dank an alle IUIler für den kollegialen und freundschaftlichen Zusammenhalt, den ich sehr schätze und der regelmäßig zur Er-

kenntnis führt, dass man die wichtigen Dinge sowieso nur in der Mensa, am Kopierer oder an der Kaffeemaschine lernt. Danke auch an alle Studenten, die im Rahmen ihrer Hiwi-Tätigkeiten und Abschlussarbeiten einen wichtigen Beitrag geleistet haben.

Das größte Dankeschön gebührt jedoch den Lieben zu Hause: meinen Eltern, Geschwistern und Schwiegereltern - sowie, allen voran, meiner Frau Margit. Danke für die selbstlose Unterstützung und die Freiräume, die ihr mir über die ganze Zeit hinweg und insbesondere über die letzten Wochen und Monate gewährt habt.

Zusammenfassung

Diese Arbeit ist angesiedelt im Kontext der drei Forschungsgebiete *Smart Service Welt*, *Modellgetriebene Softwareentwicklung* und *Intelligente Benutzerschnittstellen*. Das Ziel der Arbeit war die Entwicklung eines ganzheitlichen Ansatzes zur effizienten Erstellung von multimodalen Dialogschnittstellen für Smart Services. Um dieses Ziel zu erreichen, wurde mit KomBInoS ein umfassendes Rahmenwerk zur modellgetriebenen Erstellung solcher Benutzerschnittstellen entwickelt. Das Rahmenwerk besteht aus:

(1) einer **Metamodell-Architektur**, welche sowohl eine modellgetriebene Entwicklung als auch die Komposition von multimodalen Dialogschnittstellen für Smart Services erlaubt,

(2) einem **methodischen Vorgehen**, welches aus aufeinander abgestimmten Modelltransformationen, möglichen Kompositionsschritten und manuellen Entwicklungstätigkeiten besteht, sowie

(3) einer **integrierten Werkzeugkette** als Implementierung der Methode.

Es wurde außerdem eine cloud-fähige **Laufzeitumgebung zur mobilen Nutzung** der so erstellten Benutzerschnittstellen entwickelt. Als Proof-of-Concept werden acht **Beispielanwendungen und Demonstratoren** aus fünf Forschungsprojekten vorgestellt. Zusätzlich zur Smart Service Welt fand und findet KomBInoS auch Anwendung im Bereich der Industrie 4.0.

Abstract

This work is located the context of the three research areas *Smart Service Welt*, *Model-driven Software Development* and *Intelligent User Interfaces*. The aim of the work was the development of a holistic approach for the efficient creation of multimodal dialog interfaces for Smart Services. To achieve this goal, KomBInoS was developed as a comprehensive framework for the model-driven creation of such user interfaces. The framework consists of:

(1) a **meta-model architecture** that allows both model-driven development and the composition of multimodal dialogue interfaces for Smart Services,

(2) a **methodical approach** consisting of coordinated model transformations, possible compositional steps and manual development activities, as well as

(3) an **integrated tool chain** as an implementation of the method.

We also developed a cloud-enabled runtime environment for mobile use of the user interfaces created in this way. As a proof-of-concept, eight sample applications and demonstrators from five research projects will be presented. In addition to the Smart Service Welt, KomBInoS was also and is applied in the area of Industrie 4.0.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

KAPITEL 1

Einleitung

Dieses einleitende Kapitel motiviert die vorliegende Arbeit im Spannungsfeld der Forschungsthemen *Smart Service Welt*, *multimodale Dialogschnittstellen* und *modellgetriebene Softwareentwicklung*. Es werden die Ziele der Arbeit in Form einer Problemstellung und daraus abgeleiteten wissenschaftlichen und ingenieurtechnischen Fragestellungen formuliert. Außerdem wird der Aufbau der Arbeit kurz skizziert.

1.1 Motivation

Die zunehmende Digitalisierung ermöglicht neue internetbasierte Geschäftsmodelle und eine Globalisierung von IT-gestützten Dienstleistungen. In der Konsequenz entwickeln sich dynamische Wertschöpfungsnetze mit einer hohen Innovationsgeschwindigkeit. Das Zukunftsprojekt Smart Service Welt (Acatech 2014, 2015) fokussiert als Bestandteil sowohl der neuen Hightech-Strategie für Deutschland (Bundesregierung 2014, S. 20) als auch der Digitalen Strategie 2025 (BMWi 2016, S. 47) auf die Optimierung industrieller Prozesse über die gesamte vernetzte Wertschöpfungskette. Hierbei steht jedoch nicht das Produkt oder die Dienstleistung im Vordergrund, sondern der Nutzer von Smart Services. Dabei handelt es sich um individuell konfigurierbare Pakete aus intelligenten Produkten, physischen Dienstleistungen und digitalen Diensten. Aufgrund der Komplexität muss der Zugang zu einem Smart Service für Nutzer über eine intuitive intelligente Benutzerschnittstelle (UI, User Interface) erfolgen.

Multimodale Dialogschnittstellen ermöglichen eine natürlichere Mensch-Maschine-Interaktion (Oviatt und Cohen 2015). Sie gewähren Barrierefreiheit und sind besonders in mobilen Anwendungssituationen sinnvoll, weil dadurch die Effektivität einer Benutzerschnittstelle gesteigert wird. Einem breiten Nutzerkreis stehen solche Benutzerschnittstellen etwa in Form persönlicher digitaler Assistenten auf mobilen Plattformen zur Verfügung. Die Entwicklung solch intelligenter Benutzerschnittstellen ist in der Re-

gel aber sehr komplex und erfordert daher ein großes Maß an Expertise und Ressourcen. Domänenunabhängige und wiederverwendbare multimodale Dialogplattformen, wie etwa die am DFKI entwickelte Ontologiebasierte Dialogplattform (ODP) (Becker u. a. 2014) sowie die darauf konzeptuell aufbauende Situationsadaptive Multimodale Dialogplattform (SiAM-dp) (Neßelrath 2016), wirken dem entgegen und reduzieren den notwendigen Aufwand dank einer formalen Modellierung und darauf operierender generischer Komponenten deutlich. Diese Plattformen bestehen aus einer Laufzeitumgebung in Form eines multimodalen Dialogsystems und einer auf die Modelle abgestimmten Werkzeugunterstützung zur Entwurfszeit.

Aufgrund der hohen Innovationsgeschwindigkeit im Kontext der voranschreitenden Digitalisierung genügt dies jedoch noch nicht, um auf effiziente Art und Weise multimodale Dialogschnittstellen für Smart Services zu erstellen. Hier bietet die modellgetriebene Softwareentwicklung Lösungsansätze, um auf Basis einer Metamodell-Architektur mit Hilfe von Modelltransformationen methodisch und mit einem hohen Grad an Automatisierung ausführbare Software zu erzeugen. Modellgetriebene Verfahren zur Entwicklung von Benutzerschnittstellen sind nicht neu und gehen teilweise auch über klassische grafische Benutzerschnittstellen (GUI) hinaus. Dies schließt auch Kompositionsansätze ein, die sich aufgrund der Modellbasiertheit so erstellter Benutzerschnittstellen anbieten. Die Erstellung multimodaler Dialogschnittstellen für Smart Services wurde bislang jedoch nicht hinreichend untersucht.

Vor diesem Hintergrund wurden „*Maßnahmen zur Verbesserung der Alltagstauglichkeit semantischer Methoden und Werkzeuge für Dienstanbieter, -entwickler und -betreiber*“ gemeinsam mit der „*Verbesserung des Zugriffs auf Services durch intuitive, intelligente Benutzerschnittstellen*“ als konkreter Forschungsbedarf zur erfolgreichen Umsetzung der Smart Service Welt identifiziert (Acatech 2015, S. 166).

1.2 Ziele dieser Arbeit

Im Folgenden werden die Ziele der Arbeit erläutert und die Arbeit in den größeren Forschungskontext eingebettet. Anschließend werden daraus die wichtigsten wissenschaftlichen und ingenieurtechnischen Fragestellungen abgeleitet und formuliert.

1.2.1 Problemdarstellung

Das Ziel der Arbeit ist die Entwicklung eines ganzheitlichen Ansatzes zur effizienten Erstellung von multimodalen Dialogschnittstellen für Smart Services. Dieser Ansatz umfasst:

(1) die Entwicklung einer Metamodell-Architektur, welche sowohl eine modellgetriebene Entwicklung als auch die Komposition von Benutzerschnittstellen erlaubt,

(2) ein methodisches Vorgehen in Form von aufeinander abgestimmten Modelltransformationen und manuellen Entwicklungstätigkeiten unter Berücksichtigung sinn-

voller Wiederverwendung existierender Modellartefakte, sowie

(3) eine integrierte Werkzeugkette als Implementierung der Methode.

Ein weiteres Ziel liegt in der Entwicklung einer entsprechenden Laufzeitumgebung zur mobilen Nutzung der so erstellten multimodalen Dialogschnittstellen für Smart Services.

1.2.2 Einordnung in den Forschungskontext

Wie in der Motivation geschildert und in Abbildung 1.1 dargestellt, bewegt sich diese Arbeit im Kontext der drei Forschungsgebiete *Smart Service Welt*, *Modellgetriebene Softwareentwicklung* und *Intelligente Benutzerschnittstellen*.

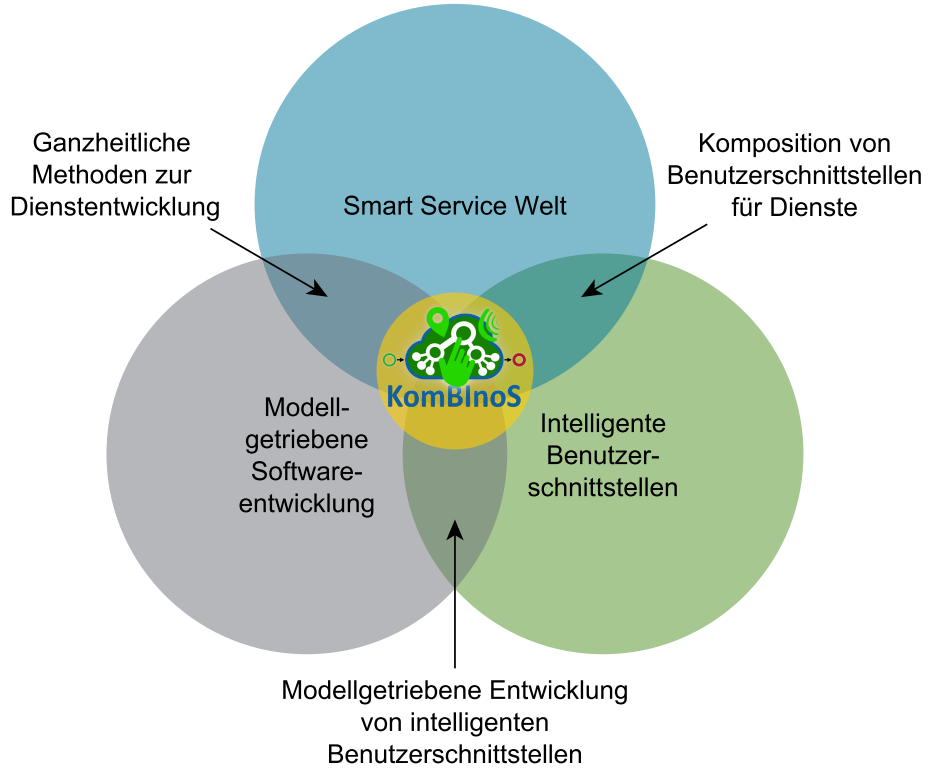


Abbildung 1.1: Einordnung der Arbeit in den Forschungskontext

Die paarweisen Schnittmengen der einzelnen Gebiete repräsentieren wiederum spezifische Themenfelder, die für diese Arbeit von besonderer Relevanz sind. Diese sind die *modellgetriebene Entwicklung von intelligenten Benutzerschnittstellen*, die *Komposition von Benutzerschnittstellen für Dienste*, sowie *ganzheitliche Methoden zur Dienstentwicklung*. Der Kern der Arbeit verortet sich in der Schnittmenge aller drei Themenfelder.

1.2.3 Wissenschaftliche und ingenieurtechnische Fragestellungen

Die vorliegende Arbeit untersucht die folgenden wissenschaftlichen Fragestellungen. Diese werden in der Diskussion am Ende dieser Arbeit in Kapitel 11 (S. 255) wieder aufgegriffen und beantwortet.

(1) **Modellgetriebene Entwicklung:** *Wie kann man multimodale Dialogschnittstellen für Smart Services schnell und mit geringem Programmieraufwand entwickeln?*
Die Innovationszyklen für Smart Services sind sehr kurz. Um für solche Dienste in hoher Geschwindigkeit adäquate intelligente Benutzerschnittstellen bereitstellen zu können, muss die UI-Entwicklung auf effiziente Softwareentwicklungstechniken zurückgreifen. Hier bieten sich modellgetriebene Verfahren an, die im Übrigen auch bei der eigentlichen Dienstentwicklung angewandt werden. Wie sehen eine notwendige Metamodell-Architektur und entsprechende Modelltransformationen aus, die funktional vollständige und über Modalitätsgrenzen hinweg konsistente Benutzerschnittstellen erschaffen?

(2) **Modellgetriebene Komposition:** *Wie kann man existierende multimodale Dialogschnittstellen von Smart Services wiederverwenden?*
In der Smart Service Welt können Dienste zu Mehrwertdiensten komponiert werden. Diese Kompositionsfähigkeit lässt sich prinzipiell auf die Ebene der Benutzerschnittstelle übertragen. Verfügen also Dienste einer Kompositionskette bereits über modellbasierte multimodale Dialogschnittstellen, bietet es sich an, diese so weit wie möglich im Rahmen einer nachgelagerten Benutzerschnittstellenkomposition wiederzuverwenden. Bis zu welchem Grad ist dies sinnvoll möglich? Wie können darüber hinaus auch Modellartefakte multimodaler Dialogschnittstellen wiederverwendet werden, deren zugehörige Dienste nicht Teil der Kompositionskette sind?

(3) **Integrierte Entwicklungs- und Kompositionsmethode:** *Wie sieht ein strukturiertes Vorgehen aus, welches sowohl Entwicklungs- als auch Kompositionstätigkeiten bei der Erstellung einer multimodalen Dialogschnittstelle für einen Smart Service integriert?*
Die modellgetriebene Entwicklung ist in der Regel ein semiautomatischer Prozess, bestehend aus automatischen Modelltransformationen und manuellen Verfeinerungen. Durch die neue Möglichkeit der Wiederverwendung von Artefakten und der Komposition existierender Benutzerschnittstellen wird eine integrierte Entwicklungs- und Kompositionsmethode ungleich komplexer.

Außerdem werden folgende ingenieurtechnische Fragestellungen untersucht, deren praktische Entwurfsziele zur Beantwortung der wissenschaftlichen Fragen notwendig sind.

(4) **Nahtlose Werkzeugunterstützung:** *Wie können Entwickler bei der Erstellung von multimodalen Dialogschnittstellen für Smart Services bei ihrer Aufgabe unterstützt werden?*

Eine nahtlose Werkzeugunterstützung ermöglicht es einem Entwickler, eine Benutzerschnittstelle vollständig zu erstellen. Sie bietet einen leicht verständlichen Zugang zur Entwicklungsmethode und unterstützt darüber hinaus auch auf technischer Ebene die verteilte Entwicklung einer Benutzerschnittstelle.

(5) **Nutzungs- und Laufzeitumgebung:** *Wie kann man Smart Services im multimodalen Dialog auf mobilen Endgeräten nutzen?*
Eine Laufzeitumgebung zur mobilen Nutzung erfordert eine offene Architektur unter Ausnutzung relevanter netzwerkbasierter Kommunikationsstandards und -protokolle zur Etablierung einer losen Kopplung und Verteilung funktionaler Komponenten in einer Cloud-Infrastruktur. Benutzer können über leichtgewichtige mobile Klienten zu jeder Zeit mit für sie relevanten Diensten in Kontakt treten, ebenso können sie über neue Entwicklungen in Kenntnis gesetzt werden.

1.3 Aufbau der Arbeit

In diesem Kapitel wurde bisher die Arbeit motiviert. Außerdem wurden die Ziele der Arbeit in Form einer Problemstellung und daraus abgeleiteten wissenschaftlichen Fragestellungen formuliert. Der weitere Aufbau der Arbeit gliedert sich in vier größere Teile, die im Folgenden kurz skizziert werden. Abbildung 1.2 stellt den Aufbau und die Abhängigkeiten der einzelnen Kapitel grafisch dar, woraus sich die Leserichtung für die Arbeit ableitet.

I. **Grundlagen und verwandte Arbeiten.** In diesem Teil wird ein Überblick über die drei Forschungsgebiete Modellgetriebene Softwareentwicklung (Kapitel 2, S. 9), Smart Service Welt (Kapitel 3, S. 29) und Intelligente Benutzerschnittstellen (Kapitel 4, S. 49) erarbeitet. Es werden grundlegende Begrifflichkeiten und Konzepte erläutert, die zum weiteren Verständnis zwingend notwendig sind. Kapitel 5 (S. 71) diskutiert verwandte Arbeiten aus den paarweisen Schnittmengen der Forschungsgebiete und leitet daraus Anforderungen an die eigene Arbeit ab, anhand derer die verwandten Arbeiten untereinander und mit dieser Arbeit verglichen werden.

II. **KomBInoS Entwurfszeit.** Dieser Teil behandelt die drei wichtigen Bestandteile, die für Vanderdonckt (2005, S. 18) eine Methodik zur modellgetriebenen Erstellung von Benutzerschnittstellen ausmachen, jedoch erweitert um einen integrierten semiautomatischen Kompositionsansatz. Dazu zählen eine Metamodell-Architektur (Kapitel 6, S. 111), eine integrierte Entwicklungs- und Kompositionsmethode (Kapitel 7, S. 143) sowie eine nahtlose Werkzeugkette (Kapitel 8, S. 187). Diese Bestandteile formen die KomBInoS[1]-Plattform zur modellgetriebenen

[1]KomBInoS war ursprünglich der Name des Software-Campus-Projekts, in dem Teile des Kompositionsansatzes entwickelt wurden und stand für *Komposition von multimodalen dialogischen Benutzerschnittstellen im Internet of Services*. Dieser Name wurde als Bezeichner für die in dieser Arbeit vorgestellte Methodik und Laufzeitumgebung beibehalten.

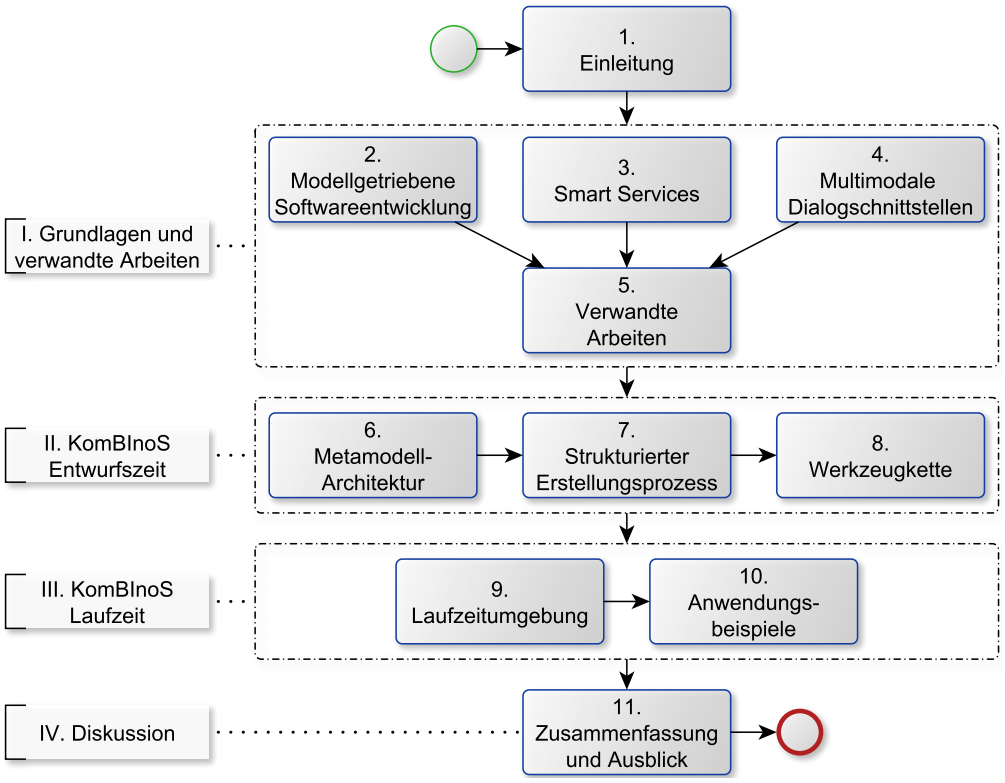


Abbildung 1.2: Aufbau der Arbeit

Entwicklung und Komposition von multimodalen Dialogschnittstellen für Smart Services.

III. **KomBInoS Laufzeit.** Dieser Teil umfasst zum einen in Kapitel 9 (S. 211) eine Laufzeitumgebung zur mobilen Nutzung von multimodalen Dialogschnittstellen für Smart Services. Zum anderen werden in Kapitel 10 (S. 231) Anwendungen vorgestellt, die im Rahmen dieser Arbeit mit Hilfe des modellgetriebenen Ansatzes und unter Verwendung der Laufzeitumgebung entwickelt wurden.

IV. **Diskussion.** Im letzten Teil wird diese Arbeit aus wissenschaftlich-technischer Sicht zusammengefasst. Vor dem Hintergrund der einleitend formulierten wissenschaftlichen und ingenieurtechnischen Fragestellungen werden die wissenschaftlichen und praktischen Beiträge der Arbeit erörtert. Es wird ein Überblick über die der Arbeit zugrundeliegenden wissenschaftlichen Veröffentlichungen und betreuten studentischen Abschlussarbeiten gegeben. Abschließend werden in einem Ausblick weiterführende zukünftige Erweiterungen der Arbeit diskutiert.

Teil I

Grundlagen und verwandte Arbeiten

KAPITEL 2

Modellgetriebene Softwareentwicklung

In diesem Kapitel wird zuerst der Begriff des Modells anhand der Allgemeinen Modelltheorie motiviert und dessen Bedeutung für die Informatik erläutert. Anschließend wird die modellgetriebene Softwareentwicklung genauer eingegrenzt, um ein gemeinsames Verständnis darüber zu erlangen. Dabei werden relevante Konzepte eingeführt, die am Beispiel der Modellgetriebenen Architektur konkretisiert werden.

2.1 Einleitung

Abstraktion ist der Schlüssel zur Erschaffung immer komplexerer Softwaresysteme. Um die Entwicklung komplexer Softwaresysteme beherrschbar zu machen, wurden auf Ebene des Managements Vorgehensmodelle spezifiziert, die den Prozess der Softwareentwicklung steuern. Ebenso wurden auf Implementierungsebene Entwurfsmuster (Gamma u. a. 1994) entwickelt, die generische Lösungsvorschläge für wiederkehrende Probleme beim Entwurf und der Implementierung von Software bieten. Dies hat zum Ziel, die Qualitätssicherung größerer Softwareprojekte zu gewährleisten, d.h. potentielle Fehlerquellen zu minimieren oder schnell und systematisch aufzuspüren, um Zeitpläne und Budgets einzuhalten. Vor diesem Hintergrund sind die Entwicklungen im Bereich der modellgetriebenen Softwareentwicklung als zusätzliche Abstraktionsschicht über dem eigentlichen Quellcode zu betrachten.

In Kapitel 2.2 (S. 10) wird der Begriff Modell anhand der Allgemeinen Modelltheorie eingeführt. Die Ausdrucksfähigkeit unterschiedlicher Modellierungsansätze wird in Kapitel 2.3 (S. 14) verglichen. Anschließend werden in Kapitel 2.4 (S. 16) die relevanten Konzepte der modellgetriebenen Softwareentwicklung erklärt, bevor diese in Kapitel 2.5 (S. 23) konkret am Beispiel der Modellgetriebenen Architektur veranschaulicht werden.

2.2 Die Allgemeine Modelltheorie

Heute wird der Begriff des Modells in vielen unterschiedlichen Lebensbereichen gebraucht. Die Wirtschaft entwickelt Geschäftsmodelle, die Politik spricht von Modellregionen und Automobilkonzerne aktualisieren in regelmäßigen Abständen ihre Modellpalette. In den Naturwissenschaften dienen Modelle, die hier auch als Theorien bezeichnet werden, als Abbilder der Wirklichkeit, anhand derer z. B. Vorhersagen über zukünftig eintretende Ereignisse gemacht bzw. vergangene erklärt werden können. Das deutsche Wort *Modell* geht auf das lateinische Wort *modulus* zurück und bedeutet soviel wie kleines Maß oder Maßstab[1]. Umgangssprachlich steht ein Modell für etwas anderes, das es ersetzt (Stachowiak 1989, S. 219). Im Folgenden wird der Modellbegriff eingeführt und dessen Bedeutung für die Informatik herausgestellt.

2.2.1 Modellbegriff und Klassifikation

Herbert Stachowiak (1973) liefert mit seiner Allgemeinen Modelltheorie eine vor allem philosophische Definition für den Modellbegriff. Diese wird aufgrund ihrer Relevanz für die vorliegende Arbeit mit Definition 1 eingeführt.

Definition 1: Modell **(Stachowiak 1973, S. 131ff.)**

Ein Modell wird anhand der folgenden drei charakterisierenden Merkmale definiert:

- **Abbildungsmerkmal**: „*Modelle sind stets Modelle von etwas, nämlich Abbildungen, Repräsentationen natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können.*“
- **Verkürzungsmerkmal**: „*Modelle erfassen im allgemeinen nicht alle Attribute des durch sie repräsentierten Originals, sondern nur solche, die den jeweiligen Modellerschaffern und/oder Modellbenutzern relevant erscheinen.*“
- **Pragmatisches Merkmal**: „*Modelle sind ihren Originalen nicht per se eindeutig zugeordnet. Sie erfüllen ihre Ersetzungsfunktion a) für bestimmte - erkennende und/oder handelnde, modellbenutzende - Subjekte, b) innerhalb bestimmter Zeitintervalle und c) unter Einschränkung auf bestimmte gedankliche oder tatsächliche Operationen.*“

Zur folgenden Diskussion sei zusätzlich auf Stachowiak (1973, S. 155ff.) verwiesen. Abbildung 2.1 veranschaulicht zudem den Zusammenhang mehrerer zweck- und zeitgebundener Modelle eines Originals für verschiedene Subjekte. Für ein Original können je nach Problemstellung, Zielsetzung und Zielgruppen zu verschiedenen Zeitpunkten mehrere Modelle existieren.

Das **Abbildungsmerkmal** lässt offen, ob ein Modell ein real existierendes, ein virtuelles oder ein imaginäres Original repräsentiert. Jedoch müssen nicht alle Eigenschaften

[1] siehe https://www.dwds.de/wb/Modell (letzter Zugriff: 04.11.2017)

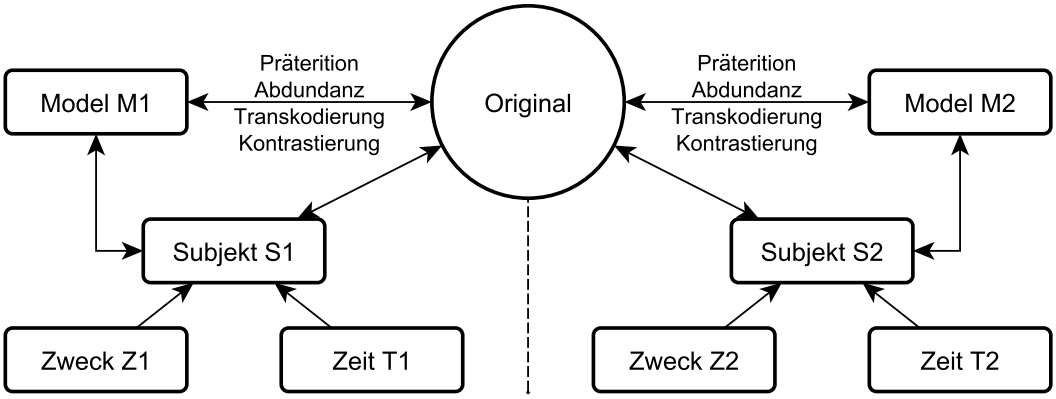


Abbildung 2.1: Zusammenhang zwischen Original und Modell nach Thomas (2001)

eines Modells im Original wiederzufinden sein (**Abdundanz**). Zudem kann ein Modell auch zeitlich vor dem repräsentierten Original existieren. In diesem Fall stellt das Modell ein Vorbild dar, welches durch eine präskriptive, realitätsschaffende Modellierung im Gegensatz zur deskriptiven, realitätsbeschreibenden Modellierung entstanden ist.

Das **Verkürzungsmerkmal** führt zur Reduzierung der Komplexität des Originals und damit zur Abstraktion vom Original, indem nicht relevante Merkmale des Originals vernachlässigt werden (**Präterition**). Dadurch treten wesentliche Merkmale deutlicher in den Vordergrund (**Kontrastierung**) und werden als solche wahrgenommen. Nach Thomas (2001) können Merkmale des Originals im Modell auch mit einer anderen Bedeutung belegt werden (**Transkodierung**). Die Auswahl der Merkmale und damit die Perspektive des Modells auf das Original obliegen dabei der subjektiven Auffassung des Modellschaffenden bzw. -nutzenden, die eng mit dem pragmatischen Merkmal verknüpft ist.

Das **pragmatische Merkmal** besagt schließlich, dass Modelle nicht zum Selbstzweck, sondern für eine konkrete Problemstellung und Zielsetzung sowie für jemanden geschaffen werden. Für unterschiedliche Problemstellungen, Zielsetzungen und Zielgruppen können also unterschiedliche Attribute eines Originals relevant sein, d.h. es können verschiedene Modelle für ein und dasselbe Original existieren. Ebenso kann sich die Relevanz eines Attributs über die Zeit verändern, sodass dieses in die Menge der relevanten Attribute aufgenommen oder daraus ausgeschlossen werden muss. Gründe dafür können - bei ansonsten unveränderter Problemstellung, Zielsetzung und Zielgruppe - Veränderungen am Original oder im individuellen Wissensstand des Modellschaffenden bzw. -nutzenden sein, die eine Aktualisierung des Modells nach sich ziehen.

Stachowiak (1973, 1989) nimmt eine Einteilung von Modellen vor. Im Sinne der Allgemeinen Modelltheorie unterscheidet er die drei Hauptarten der grafischen, technischen und semantischen Modelle, die jedoch nicht scharf gegeneinander abgegrenzt werden können. **Technische Modelle** beschreiben physikalische, biologische, psychologische

oder soziologische Systeme. Stachowiak spricht daher auch von physiko-, bio-, psycho-, und soziotechnischen Modellen, die auch in elektronischer Form vorliegen oder computergestützt ausgewertet werden können. Insbesondere sind etwa Modelle für die computerunterstützte Zusammenarbeit und allgemein Modelle für die Mensch-Computer-Interaktion den psycho- und soziotechnischen Modellen zuzuordnen.

2.2.2 Die Allgemeine Modelltheorie in der Informatik

Für den Duden (Claus und Schwill 1993, S. 294) ist die Informatik die „*Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen, besonders der automatischen Verarbeitung mit Hilfe von Digitalrechnern*“. Dabei spielen Modelle eine wesentliche Rolle, denn eine systematische und automatisierte Verarbeitung von Information mit Hilfe von Computern ist nur durch eine digitale Repräsentation des informationsverarbeitenden Systems möglich. Diese digitale Repräsentation wird zum Zweck der computergestützten Informationsverarbeitung von jemandem und für jemanden erstellt. Außerdem umfasst die digitale Repräsentation des Systems nur die für die Informationsverarbeitung relevanten Elemente des Systems und deren Beziehungen. Somit stellt eine solche digitale Repräsentation im Sinne der Allgemeinen Modelltheorie ein pragmatisches Modell des informationsverarbeitenden Systems dar.

Aufgrund der vielen interdisziplinären Ausprägungen der Informatik (z. B. Künstliche Intelligenz, Wirtschafts-, Bioinformatik, etc.) ist nach Thomas (2001) auch Stachowiaks Modellklassifikation für die Informatik relevant. So lassen sich leicht Beispiele für sämtliche Modellarten aus dem Bereich der Informatik finden, da dort oftmals reelle Systeme die Rolle des Originals einnehmen, um z. B. anhand abgeleiteter Modelle einen Erkenntnisgewinn durch computergestützte Simulation oder Berechnung zu erreichen.

Jockisch und Rosendahl (2009) argumentieren, dass aufgrund der vielen interdisziplinären Einflüsse in der Informatik natürlich domänenspezifische Modellklassifikationen, wie an den Beispielen Wirtschafts- und Ingenieurwissenschaft erläutert, angewendet werden können. Darüber hinaus entwickeln sie einen eigenen interdisziplinären Ansatz zur Modellklassifikation unter Berücksichtigung des in Abbildung 2.1 dargestellten Beziehungsdreiecks Original – Modell – Subjekt im Kontext der Allgemeinen Modelltheorie. Abbildung 2.2 zeigt die im Folgenden beschriebene zweidimensionale Klassifikationsmatrix.

Die Klassifikation bedient sich zweier Merkmale. Das erste Merkmal trifft eine Einordnung anhand des Originals, oder konkreter nach der Art des Systems. Unterschieden wird zwischen technischen und sozialen Systemen. Dazwischen bildet sich die Schnittmenge der soziotechnischen Systeme aus, in denen Menschen allgemein mit Technologie interagieren. Das zweite Merkmal trifft eine Unterscheidung anhand des Modells. Dazu ist es erforderlich, den Zweck des Modells und die Perspektive des Modellschaffenden zu hinterfragen. Zweck und Perspektive korrelieren dabei und implizieren eine gewisse Detailliertheit des Modells. Daraus resultieren unterschiedliche Betrachtungsebenen (von

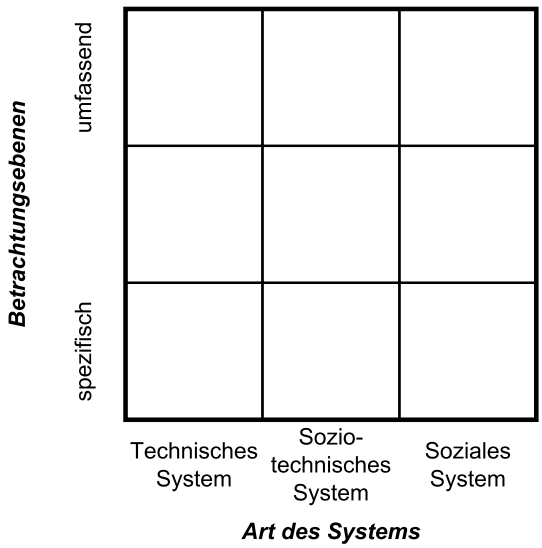


Abbildung 2.2: Matrix zur Klassifikation von Modellen nach Betrachtungsebene und Art des Systems nach Jockisch und Rosendahl (2009).

umfassend bis spezifisch), auf denen ein Modell einen Erkenntnisgewinn herbeiführen kann. Eine spezifische Betrachtungsebene erlaubt einen ganz spezifischen Erkenntnisgewinn und verlangt dafür oftmals eine sehr detaillierte Modellierung eines Teilaspekts. Im Gegensatz dazu ist der Erkenntnisgewinn auf einer umfassenden Betrachtungsebene allgemeiner. Dafür bedarf es - bildlich formuliert - einer breiteren Modellierung, anstatt einer tiefen. Es ist wichtig festzuhalten, dass dies keine quantitative, sondern eine qualitative Unterscheidung des Modells darstellt. Ausgehend von einem bestimmten Betrachtungspunkt, wird ein Modell als **abstrakt** bezeichnet, wenn es einen allgemeinen Aspekt eines Originals umfassend repräsentiert. In Relation dazu bildet ein **konkretes** Modell einen spezifischeren Teilaspekt des gleichen Originals detaillierter ab.

Für den Modellbegriff in der Informatik ist nach Kühne (2005) primär die Unterscheidung zwischen sogenannten Token-Modellen und Typ-Modellen essentiell, um Kommunikationsirrtümer zwischen den an der Modellierung beteiligten Personen zu vermeiden. **Token-Modelle** bilden singuläre Aspekte eines Originals direkt ab. Im Gegensatz dazu bilden **Typ-Modelle** universelle Aspekte eines Originals ab. Somit wird die Abstraktionsfähigkeit des Menschen ausgenutzt, um Objekte anhand gemeinsamer Eigenschaften zu klassifizieren und darauf aufbauend Schlussfolgerungen zu ziehen. Atkinson und Kühne (2002, 2003) differenzieren außerdem zwischen linguistischen und ontologischen Typ-Modellen. **Ontologische Typ-Modelle** stellen eine domänenspezifische Konzeptmodellierung dar. **Linguistische Typ-Modelle** bilden eine Sprachebene zur Repräsentation von ontologischen Typmodellen. Erst dadurch können domänenspezifische Konzepte mit Attributen versehen und Relationen zwischen domänenspezifischen Konzepten erfasst werden. Daher bezeichnet Kühne (2005) linguistische Typ-Modelle

als Repräsentationssprache und ontologische Typ-Modelle als Inhaltssprache. Aufgrund der fehlenden Abstraktion sind Token-Modelle für den Einsatz in der modellgetriebenen Softwareentwicklung (Kapitel 2.4, S. 16) ungeeignet. Stattdessen werden hauptsächlich Typ-Modelle verwendet.

Die Unterscheidung der Begriffe Token-Modell, ontologisches Typ-Modell und linguistisches Typ-Modell ermöglicht die Einführung des Begriffs **Metamodell**. Kühne (2005) merkt dazu an, dass die Vorsilbe *Meta* allgemein für das Ergebnis einer zweifach ausgeführten Operation verwendet wird. Führt man also im konkreten Fall ausgehend von einem Original eine zweifache Modellbildung bzw. Klassifikation durch, so stellt das Resultat ein Metamodell dar. Kühne (2005) bezeichnet ein solches Metamodell als *echtes* Metamodell, wenn die durchgeführten Modellbildungen nicht transitiv sind. Im Rahmen dieser Arbeit spielt diese Unterscheidung eine untergeordnete Rolle, weshalb Definition 2 lediglich die grundlegendste Eigenschaft eines Metamodells anführt.

Definition 2: Metamodell in KomBInoS

Ein Metamodell beschreibt eine Konzeptualisierung einer Domäne. Durch Bildung von Instanzen können bestimmte Sachverhalte innerhalb dieser Domäne in einem Modell formal abgebildet werden.

Im Folgenden wird die Vorsilbe *Meta* im Zusammenhang mit einem Modell gelegentlich ausgelassen. Die Bedeutung erschließt sich dennoch eindeutig aus dem Kontext, etwa durch Verwendung des bestimmten Artikels: das (Meta-)Modell.

2.3 Spektrum der Ausdrucksfähigkeit von Metamodellen

Vor dem Hintergrund des avisierten Zwecks der Modellierung ist es initial erforderlich, ein geeignetes linguistisches Typmodell auszuwählen. In Anlehnung an Daconta u. a. (2003, S. 157) vergleicht Abbildung 2.3 die Ausdrucksfähigkeit bekannter Metamodellierungssprachen und mathematisch-logischer Repräsentationsformalismen in Abhängigkeit von ihrer Komplexität.

Demnach ermöglichen relationale Datenbanksysteme eine sehr effiziente Datenverwaltung, aber selbst mit dem erweiterten Entity-Relationship-Modell stößt die Datenmodellierung an ihre Grenzen. Die auf das Paradigma der Objektorientierung ausgerichtete Unified Modeling Language (UML) bietet weitergehende Möglichkeiten. Deren Ausdrucksfähigkeit bleibt jedoch zugunsten einer geringeren Komplexität hinter logikbasierten formalen Ontologiesprachen wie der Web Ontology Language (OWL) (W3C OWL Working Group 2012) zurück. OWL ist zusätzlich zum Resource Description Framework (RDF) (Cyganiak u. a. 2014), RDF-Schema (Brickley und Guha 2014) und der SPARQL Protocol and RDF Query Language (SPARQL) (Harris und Seaborne 2013) ein wichtiger Teil des vom World Wide Web Consortium (W3C) standardisierten Technologiestacks zur Realisierung des Semantic Web (Fensel u. a. 2003). Es existieren

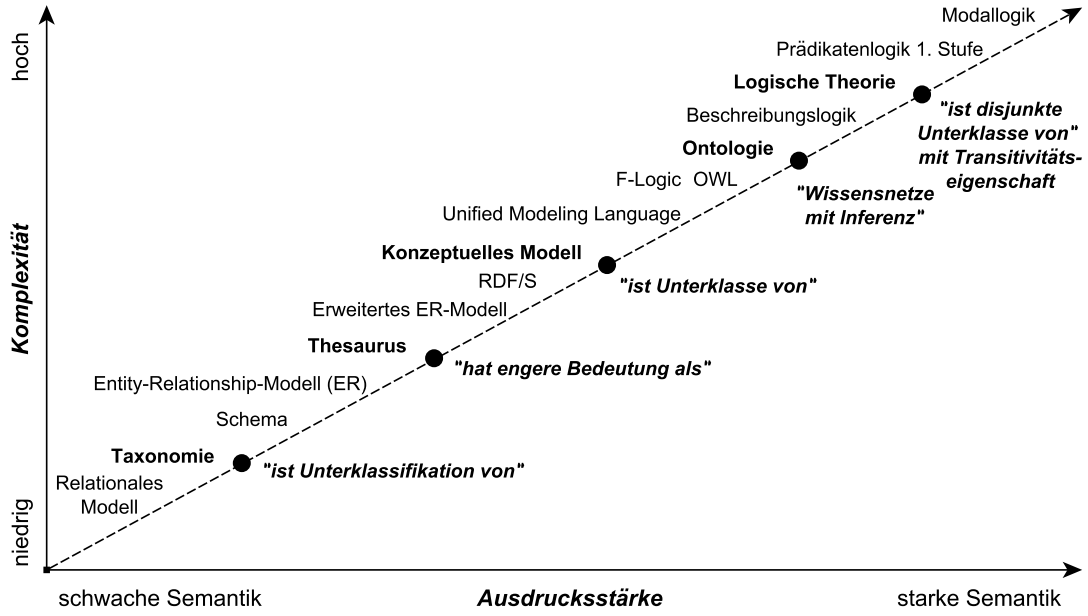


Abbildung 2.3: Spektrum der Ausdrucksfähigkeit von Metamodellen nach Daconta u. a. (2003, S. 157)

mit OWL-Lite, OWL-DL und OWL-Full drei in ihrer Axiomatisierung und damit in ihrer Ausdrucksfähigkeit abgestufte Varianten. Alle Ontologiesprachen verfügen über eine formale, modelltheoretische Semantik (Bußmann 2002, S.441), auf deren Grundlage effiziente Deduktionsalgorithmen logisches Schließen ermöglichen. So wird etwa zur Spezifikation der modelltheoretischen Semantik von OWL-DL (Hitzler u. a. 2008, S. 172ff.) auf epistemologischer Ebene die entscheidbare Beschreibungslogik $\mathcal{SHOIN}(\mathcal{D})$ (Horrocks und Patel-Schneider 2004) herangezogen. Eine mit OWL umzusetzende Domänenmodellierung kann auf wiederverwendbaren Upper-Level- und Basisontologien, wie SUMO (Niles und Pease 2001) oder DOLCE (Gangemi u. a. 2002), aufbauen, welche bereits abstrakte Konzepte und logische Zusammenhänge abbilden. Die Grenzen effizienter Berechenbarkeit schließen den praktischen Einsatz noch ausdrucksstärkerer mathematischer Modelltheorien, wie die Prädikaten- oder Modallogik, aufgrund ihrer Nichtentscheidbarkeit faktisch aus.

Paulheim und Probst (2013) sehen einen fundamentalen Unterschied zwischen einer im nächsten Unterkapitel eingeführten domänenspezifischen Sprache und einer ontologischen Modellierung in der Hinsicht, dass eine domänenspezifischen Sprache vor dem Hintergrund eines bestimmten Zwecks, etwa der einfachen Codegenerierung, erstellt wurde, wohingegen eine Ontologie versucht, die Realität allgemeingültig und nicht an einen konkreten Zweck gebunden möglichst genau abzubilden. Ontologiesprachen wie OWL können natürlich auch für die Modellbildung gemäß der Allgemeinen Modelltheorie eingesetzt werden, auch vor dem Hintergrund der in diese Arbeit angewendeten

modellgetriebenen Entwicklung. Hierzu wird in diesen Anwendungsfällen die größere Ausdrucksfähigkeit sowie die Fähigkeit zum logischen Schließen nur selten benötigt bzw. ausgenutzt. Umgekehrt kann aber verstärkt festgestellt werden, dass auch die Semantic Web-Gemeinde Vorteile der modellgetriebenen Softwareentwicklung für sich entdeckt (Gasevic u. a. 2009). Das Werkzeug *RDFReactor*[2] erzeugt etwa aus einer Ontologie Java-Klassen, die es einem Entwickler ermöglichen, Ontologiekonzepte objektorientiert zu handhaben. Dies hebt den technischen Abstraktionsgrad des Quellcodes im Umgang mit einer ontologischen Wissensrepräsentation, senkt die Einstiegshürde für mit semantischen Technologien unerfahrene Entwickler und stellt zugleich einen uniformen Zugriff auf die Wissensbasis sicher. Dieses Werkzeug wird auch im Projekt *twouse*[3] benutzt. Staab u. a. (2010) verbinden in diesem Rahmen die modellgetriebene Softwareentwicklung mit dem Semantic Web durch eine Verknüpfung der UML mit der OWL. Weiterhin beschreiben Rahmani u. a. (2010) eine Transformation zwischen OWL und dem Eclipse Modeling Framework zugrundeliegenden Meta-Metamodell Ecore. Dadurch kann mittelfristig auch Wissen und Erfahrungen vom Semantic Web in die modellgetriebene Softwareentwicklung zurückfließen.

2.4 Charakteristika der modellgetriebenen Softwareentwicklung

Dieses Unterkapitel erläutert zunächst die grundlegenden Konzepte der modellgetriebene Softwareentwicklung (MDSE, Model-driven Software Engineering) und grenzt den Begriff anschließend von verwandten Vorgehensweisen zur Softwareentwicklung ab.

2.4.1 Begriffsbestimmung und grundlegende Konzepte

Definition 3 führt den Begriff der MDSE auf Basis von Stahl u. a. (2007) ein. Die wesentlichen Bestandteile der Definition werden im Folgenden diskutiert und verfeinert, um ein vollständiges Bild der MDSE zu erhalten.

Definition 3: Modellgetriebene Softwareentwicklung **(Stahl u. a. 2007, S.11)**

Modellgetriebene Softwareentwicklung ist ein Oberbegriff für Techniken, die aus formalen Modellen (teil-) automatisiert lauffähige Software erzeugen.

Der Begriff *Techniken zum automatisierten Erzeugen lauffähiger Software* meint sowohl die Benutzung von Codegeneratoren zur Erzeugung von Quellcode anhand vordefinierter Codegenerierungsschablonen als auch die Entwicklung und Nutzung von Interpretern, welche formale Modelle gemäß einer operationellen Semantik zur Laufzeit

[2] siehe http://semanticweb.org/wiki/RDFReactor (letzter Zugriff: 04.11.2017)
[3] siehe http://code.google.com/p/twouse/ (letzter Zugriff: 04.11.2017)

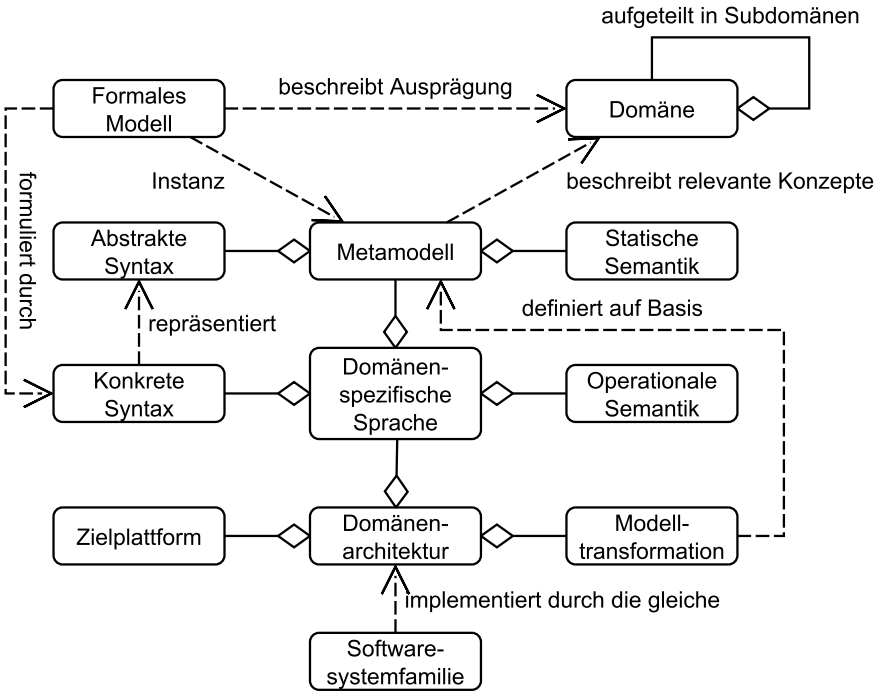


Abbildung 2.4: Wichtige Begriffe der modellgetriebenen Softwareentwicklung und deren Zusammenhang in UML-Notation nach Stahl u. a. (2007, S. 28 und 34)

ausführen. Ein formales Modell wird in Abbildung 2.4 charakterisiert. Diese Abbildung führt zugleich weitere Begriffe im Kontext der MDSE ein, die im Folgenden auf Basis von Stahl u. a. (2007, S. 28-35) erläutert werden.

Eine **Domäne** ist ein begrenztes Interessens- und Wissensgebiet, welches in einem formalen Modell abgebildet werden soll. Große Domänen können zur Reduktion der Modellkomplexität in mehrere disjunkte Unter- bzw. Teildomänen aufgespalten werden, sodass diese die Domäne im Sinne der Anforderungen an das spätere System im Verbund vollständig beschreiben. Vor diesem Hintergrund kann eine fachliche Aufteilung in statische und dynamische Teilaspekte des Systems erfolgen. Ein statischer Aspekt ist beispielsweise das domänenspezifische Datenmodell, dynamische Aspekte sind etwa zeitliche oder ereignisgesteuerte Prozesse innerhalb des modellierten Systems. Eine technische Aufteilung bündelt nichtfachliche, aber implementierungsrelevante Querschnittsfunktionalitäten wie Persistieren von Datenobjekten, Datenbankzugriffe oder Schreiben von Logdateien. Im Folgenden betrachten wir jedoch ausschließlich fachliche Aspekte.

Ein **Metamodell** beschreibt gemäß Definition 2 relevante Konzepte einer Domäne und deren Verknüpfungen untereinander. Diese werden mit Hilfe der abstrakten Syntax und der statischen Semantik formalisiert. Wurde eine Domäne in mehrere Teildomänen zerlegt, so existiert für jede dieser Teildomänen ein entsprechendes Metamodell.

Die **abstrakte Syntax** definiert in Form von Klassen, Attributen und Referenzen die Konstruktionsregeln zur Erstellung von formalen Modellen. Im Gegensatz dazu bestimmt die **konkrete Syntax** die Notation der abstrakten Syntax. Häufig ist die Wahl der Notation entscheidend für das Verständnis eines Modells und beeinflusst gleichermaßen die Entwicklung und Bereitstellung von Modellierungswerkzeugen. Da der Modellerschaffer ein Modell anhand der gewählten Notation erstellt, sollte diese für den modellierten Teilaspekt geeignet sein. Zur Modellierung dynamischer Teilaspekte eignen sich graphische Notationen, für statische Teilaspekte genügen oft schon textuelle Notationen. Für eine abstrakte Syntax können mehrere Notationen definiert sein.

Die **statische Semantik** eines Metamodells definiert die Bedeutung der einzelnen, isoliert betrachteten Elemente der abstrakten Syntax[4]. Diese kann intuitiv verständlich sein (etwa durch geeignete Namensgebung) bedarf aber auch einer formalen Spezifikation. Dazu werden auf Ebene der abstrakten Syntax Bedingungen festgelegt, die für alle Instanzen eines Metamodells gelten müssen. So werden Modellierungsfehler vermieden bzw. schon während der Modellerstellung durch eine statische Validierung erkannt. Damit entscheidet die statische Semantik über die Wohlgeformtheit eines Modells.

Ein **formales Modell** beschreibt schließlich eine Ausprägung der Domäne mit Hilfe eines Metamodells. Dazu instanziiert ein formales Modell Konzepte des Metamodells unter Berücksichtigung der statischen Semantik und verknüpft diese sinngemäß. Wie bereits angemerkt, können formale Modelle in Abhängigkeit vom zugrundeliegenden Metamodell statische oder dynamische Sachverhalte einer Domäne beschreiben. Für letztere bedarf es einer zusätzlichen operationellen Semantik.

Die **operationelle Semantik** spezifiziert valide Zustandsänderungen eines prozessbeschreibenden Modells und wird auf Basis des Metamodells definiert. Zustandsänderungen werden zur Laufzeit durch die Anwendung von Operationen auf ein Modell angestoßen und dürfen die statische Semantik nicht verletzen. Dazu muss für jede Operation definiert sein, welche Vorbedingungen ein Modell erfüllen muss, um anwendbar zu sein. Ebenso muss definiert sein, welche Auswirkungen die Anwendung der Operation auf das Modell - und damit gegebenenfalls auf das entsprechende Original - hat.

Sind zu einem Metamodell neben der obligatorischen abstrakten Syntax und der statischen Semantik zusätzlich auch eine konkrete Syntax sowie eine operationelle Semantik spezifiziert, so erhält man eine **domänenspezifische Sprache**, die als eine Programmiersprache für die betrachtete Domäne fungiert. Oft ist es praktikabel die operationelle Semantik einer neuen domänenspezifischen Sprache durch eine Abbildung auf eine bereits existierende (domänenspezifische) Programmiersprache zu spezifizieren.

[4]Die Definition der statischen Semantik umfasst in (Stahl u. a. 2007) lediglich die Spezifikation von formalen Rahmenbedingungen an Metamodellelemente. Die eigentliche Bedeutungsdefinition der einzelnen Metamodellelemente wird der „*dynamischen Semantik*“ zugesprochen. Diese Aufteilung ist wenig intuitiv, da man zur Spezifikation von formalen Rahmenbedingungen zuerst die eigentlich intendierte Bedeutung eines Metamodellelements verstehen muss. Zudem ist die Bezeichnung „*dynamische Semantik*“ etwas irreführend. Die Semantik ist natürlich nicht dynamisch, d.h. sie ändert sich nicht. Gemeint ist vielmehr die Semantik von (prozessbeschreibenden) Metamodellen zur Laufzeit, was im Folgenden als operationelle Semantik bezeichnet wird.

Allgemein führt eine **Modelltransformation** eine Umwandlung von ein oder mehreren Quellmodellen durch. Hinsichtlich des Resultats einer Modelltransformation lassen sich zwei grundlegende Arten von Modelltransformationen unterscheiden. Ist das Resultat einer Modelltransformation ebenso ein formales Modell wie das der Transformation unterzogene Quellmodell, so spricht man von einer **Modell-zu-Modell-Transformation (M2M, model-to-model)**. Dabei wird die Transformation als Abbildung eines formalen Quellmodells ins Zielmodell auf Basis der zugrundeliegenden Metamodelle definiert. Anhand der Fachlichkeit und der Abstraktionsebene des formalen Quell- und Zielmodells lassen sich M2M-Transformationen weiterhin in vertikale und horizontale M2M-Transformationen unterteilen (Metzger 2005). Vertikale M2M-Transformationen durchbrechen eine Abstraktions- bzw. Betrachtungsebene. Meist ist das Quellmodell dabei fachlich auf einer umfassenderen Betrachtungsebene (Abbildung 2.2) angesiedelt als das Zielmodell. Aus technischer Sicht ist das Zielmodell bezogen auf die spätere Zielplattform konkreter bzw. *näher* am Quellcode als das Quellmodell. Bei einer horizontalen M2M-Transformation modelliert das Quellmodell einen anderen Aspekt der betrachten Domäne als das Zielmodell, trotzdem lässt sich Information vom Quellmodell ins Zielmodell übertragen. Vor allem wenn mehrere Quell- oder Zielmodelle an einer Transformation beteiligt sind, treten Mischformen auf. So kann eine M2M-Transformation ein Modell eines Aspektes auf eine spezifischere Betrachtungsebene überführen und gleichzeitig wird Information aus einem fachlich orthogonalen Modell in das Zielmodell integriert. Durch eine M2M-Transformation wird im Allgemeinen kein vollständiges, sondern vielmehr lediglich ein partielles Zielmodell aufgebaut, welches im Anschluss an die vorangegangene Transformation ausspezifiziert werden muss. Jedoch ist das partielle Zielmodell aufgrund der Abbildungsvorschrift konsistent zu den Quellmodellen, worin neben der Zeitersparnis ein Hauptvorteil besteht. Ist das Resultat einer Modelltransformation kein formales Modell, sondern Quellcode, so handelt es sich dabei um eine **Modell-zu-Code-Transformation (M2C, model-to-code)** oder synonym um eine Codegenerierung. Die Abbildung des oder der Quellmodelle in den Quellcode erfolgt anhand von Codegenerierungsschablonen. Die Codegenerierung stellt das letzte Glied in einer Transformationskette dar. Analog zu einem abgeleiteten partiellen Zielmodell einer M2M-Transformation, muss der durch eine M2C-Transformation erzeugte Quellcode im Allgemeinen ebenfalls manuell vervollständigt werden. Zusätzlich erfordert die Spezifikation von M2C-Transformationen fundiertes Wissen über die Zielplattform, auf der das generierte System letztlich ausgeführt werden soll.

Eine **Zielplattform** beschreibt die Ausführungsumgebung der zu entwickelnden Software. Diese ist geprägt durch das zugrundeliegende Laufzeitsystem, der gewählten Programmiersprache des zu generierenden Quellcodes, der zur Verfügung stehenden Programmierschnittstellen und Softwarebibliotheken sowie daraus resultierender Besonderheiten hinsichtlich unterstützter Programmiermodelle und -paradigmen, z. B. Model-View-Controller. Während die domänenspezifische Sprache einer Domäne ausschließlich den Problemraum des zu entwickelnden Systems beschreibt, stellt die Zielplattform den Lösungsraum dar. Eine Zielplattform sollte daher zur Stützung der Domäne möglichst

leistungsfähig zusammengestellt werden.

Eine DSL samt Modelltransformationen und einer Zielplattform bilden zusammen eine **Domänenarchitektur**. Diese ermöglicht es schließlich, wie eingangs in Definition 3 festgelegt, ausgehend von einem formalen Modell (teil-) automatisiert lauffähige Software zu erzeugen. Die Domänenarchitektur bestimmt dabei, welche domänenspezifischen Konzepte betrachtet und wie diese auf die Zielplattform abgebildet werden. Somit ist eine Domänenarchitektur nicht nur domänenspezifisch, sondern auch plattformspezifisch.

Eine **Softwaresystemfamilie** bezeichnet alle Softwaresysteme, die sich mit einer Domänenarchitektur ausdrücken lassen. Somit kann eine Domänenarchitektur zur Entwicklung dieser Systeme wiederverwendet werden.

2.4.2 Abgrenzung der modellgetriebenen Softwareentwicklung

Nachdem im vorangegangenen Abschnitt die modellgetriebene Softwareentwicklung charakterisiert und wichtige Konzepte in deren Kontext erläutert wurden, wird sie in diesem Abschnitt gegen andere, teils verwandte Vorgehensweisen zur Softwareentwicklung unter Beteiligung von Modellen abgegrenzt. Dabei können involvierte Modelle stark im Grad des Formalismus und der Abstraktion variieren. Während informale Modelle reine Gedankenmodelle oder handgeschrieben sein können, sind formale Modelle mit Hilfe einer definierten Syntax und Semantik erstellt und können automatisiert in Quellcode übersetzt werden. Der Zusammenhang zwischen Modell und Code lässt sich in einem Modellierungsspektrum darstellen, welches in Abbildung 2.5 gezeigt wird. Das Modellierungsspektrum schlüsselt auf, wie und in welchem Umfang eine Systemmodellierung zur Softwareentwicklung eingesetzt wird.

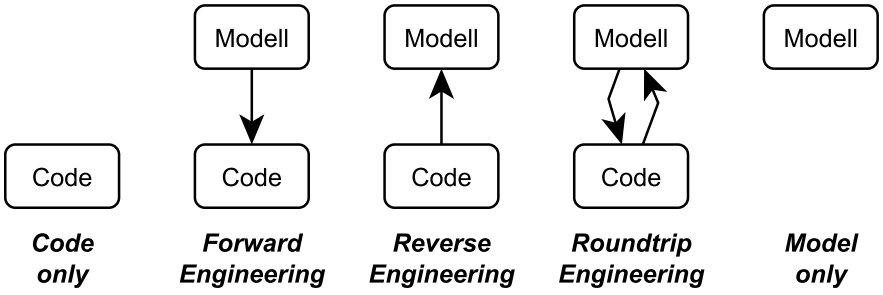


Abbildung 2.5: Das Modellierungsspektrum nach Brown u. a. (2005, S. 4)

Stahl u. a. (2007, S. 44f.) teilen dieses Spektrum in die Facetten Forward, Reverse und Roundtrip Engineering ein. Brown u. a. (2005) unterscheiden zusätzlich noch die Randfacetten Code only und Model only.

Beim **Code only**-Ansatz existiert eine abstrakte Modellierung des zu entwickelnden Systems in Form von Architekturskizzen, Schnittstellenbeschreibungen und Designent-

würfen. Diese sind informal, z. B. in Form von handgeschriebenen Notizen oder elektronischen Präsentationen, angefertigt. Oftmals existieren diese Modelle auch nur in den Köpfen der verantwortlichen Entwickler, sodass sie für Außenstehende nicht ohne Weiteres nachvollziehbar sind. Die abstrakten Modelle werden manuell mit den zur Verfügung stehenden Mitteln der gewählten Programmiersprache umgesetzt. Aufgrund des dabei entstehenden Medienbruchs, aber auch durch den fortschreitenden Entwicklungsprozess kann die reale Implementierung vom ursprünglichen Modell abweichen. Trotzdem ist dieses Vorgehen aufgrund der geringen Einstiegshürde gerade bei kleineren bis mittleren Softwareprojekten heute immer noch beliebtes Mittel zum Zweck.

Unter **Forward Engineering** versteht man das Entwickeln eines Systems anhand formaler Modelle mit niedrigem Abstraktionsgrad. Diese können daher recht einfach und automatisiert in Quellcode überführt werden. Dabei werden z. B. anhand von Codegenerierungsschablonen eines Codegenerators Codefragmente und -hülsen erzeugt, die im Anschluss ausimplementiert werden müssen (z. B. Methodenkörper). Anpassungen am generierten Code dürfen jedoch nicht vorgenommen werden. Solche Programmänderungen müssen zwingend am Modell bzw. den Generierungsschablonen durchgeführt werden, da diese gegebenenfalls durch erneutes Generieren überschrieben werden. Die Notwendigkeit kleiner Anpassungen am Code nehmen so zwar unverhältnismäßig viel Zeit in Anspruch. Ein solch diszipliniertes Vorgehen hat jedoch den Vorteil, dass die Software einer einheitlichen Architektur folgt, welche durch die in den Codegenerierungsschablonen definierten Abbildungen der Modelle in den Code festgelegt wird. Dies erleichtert die Nachvollziehbarkeit, die Dokumentation sowie das Testen des Codes und führt somit zu einem Qualitätsgewinn. Demgegenüber stehen auf der anderen Seite relativ hohe Anfangsinvestitionen für die Entwicklung der fachlichen Modelle und der dazugehörigen Generierungsschablonen, die sich erst durch Wiederverwendung, etwa bei der Entwicklung gleichartiger Systeme, lohnen. Damit kann für die Entwicklung dieser Systeme auf bereits vorhandenes Expertenwissen zurückgegriffen werden. Gleichzeitig wird die Entwicklung beschleunigt.

Reverse Engineering beschreibt den umgekehrten Weg, sprich das Extrahieren und Visualisieren von Modellen anhand der Codebasis. Wegen der Nähe zur Codebasis – Code und Modell weisen typischerweise den gleichen Abstraktionsgrad auf – spricht man auch von einem Code-Diagramm. Eine solche Visualisierung macht die Implementierung nachvollziehbarer.

Das **Roundtrip Engineering** ist eine Abfolge von Forward und Reverse Engineering. Im einfachsten Fall bietet eine moderne integrierte Entwicklungsumgebung (IDE, Integrated Development Environment) die Möglichkeit, die existierende Codebasis mit Hilfe einer graphischen Notation zu veranschaulichen, z. B. in Form von Klassen- oder Aufrufdiagrammen. Durch die enge Kopplung zwischen Quellcode und Diagramm ist es leicht möglich, anstatt des textuellen Quellcodes alternativ das Diagramm zu editieren. Änderungen am Diagramm werden sofort mit der Codebasis synchronisiert und umgekehrt. Je größer der Abstraktionsunterschied zwischen Modell und Codebasis, desto mehr Aufwand und Werkzeugunterstützung wird zur Synchronisierung dieser benötigt.

Dann geschieht ein Abgleich nicht mehr unmittelbar, sondern in regelmäßigen zeitlichen Abständen. Zur Vermeidung von Inkonsistenzen und Konflikten wird Entwicklern hierbei ein sehr hohes Maß an Disziplin abverlangt. Generell sollte generierter Code nicht manuell verändert und von handgeschriebenem Code getrennt gehalten werden.

Im **Model only**-Ansatz dienen Modelle lediglich als Gedankenstütze, Kommunikationsmedium und Diskussionsgrundlage zum Erschließen der Anwendungsdomäne und zum Herausbilden eines gemeinsamen Verständnisses in und zwischen den an der Entwicklung beteiligten Gruppen. Die tatsächliche Implementierung ist bei diesem Ansatz meist von der Modellierung losgelöst.

2.4.3 Vor- und Nachteile der modellgetriebenen Softwareentwicklung für KomBInoS

Auf Grundlage der bisherigen Ausführungen wird im Folgenden ein prägnanter Überblick über die Vor- und Nachteile der modellgetriebenen Softwareentwicklung für KomBInoS gegeben.

Nachteile

- Die im Voraus zu leistenden Aufwände bei der Konzeption und Implementierung einer Domänenarchitektur machen sich erst nach einer bestimmten Anzahl an damit umgesetzten System bezahlt.
- Um eine gewisse Varianz innerhalb einer Softwaresystemfamilie zu ermöglichen, sollte die zugrundeliegende Domänenarchitektur bis zu einem gewissen Grad flexibel gestaltet sein.
- Im Allgemeinen ist das nachträgliche Einarbeiten neuer Funktionalität oder bis dato unbeachteter Konzepte aufwändig.

Vorteile

- Die (Wieder-) Verwendung einer Domänenarchitektur wirkt sich positiv auf die Entwicklungsgeschwindigkeit und die Qualität eines danach zu erstellenden Systems aus.
- Insbesondere kann hierdurch die Vollständigkeit und die Konsistenz einer multimodalen Dialogschnittstelle in Bezug auf die zur Verfügung stehenden Ein- und Ausgabemodalitäten sichergestellt werden.
- Im Speziellen ist die Integration neuer Ein- und Ausgabegeräte durch Erweiterung einer bestehenden oder Definition einer neuen Zielplattform bei entsprechend modularer Domänenarchitektur einfach zu bewerkstelligen. So kann eine massive Multimodalität erreicht werden.
- Ein modellgetriebenes Vorgehen unter Anwendung einer Domänenarchitektur gewährt ein einheitliches Nutzererlebnis über Benutzerschnittstellen hinweg.
- Die Anwendung von MDSE erleichtert die kontinuierliche Wartung von pflegeintensiven multimodalen Dialogschnittstellen.

2.5 Modellgetriebene Architektur

Die Modellgetriebene Architektur (MDA) ist ein Rahmenwerk der Object Management Group (Miller und Mukerji 2003) für die MDSE. Die Idee hinter der Modellgetriebene Architektur (MDA, Model-driven Architecture) ist die Trennung der fachlichen Spezifikation eines Softwaresystems von den technischen Rahmenbedingungen und Eigenschaften einer Zielplattform, auf der das Softwaresystem ausgeführt wird. Die Ziele der MDA sind die **Portabilität**, **Interoperabilität** und **Wiederverwendbarkeit** von Systemspezifikationen durch die architektonische Trennung der fachlichen und technischen Aspekte (Miller und Mukerji 2003, S. 2-2) . Vor diesem Hintergrund erlaubt die MDA die Spezifikation eines Systems als plattformunabhängiges Modell, die Spezifikation von Zielplattformen mittels eines Plattformdefinitionsmodells, sowie die Überführung der plattformunabhängigen Spezifikation in eine Spezifikation für eine bestimmte Zielplattform, einem plattformspezifischen Modell. Dieses Vorgehen ist in Abbildung 2.6 dargestellt und kann als Muster wiederholt angewendet werden, da die Unterscheidung zwischen Plattformabhängigkeit und -unabhängigkeit jeweils relativ zu einer Zielplattform getroffen wird.

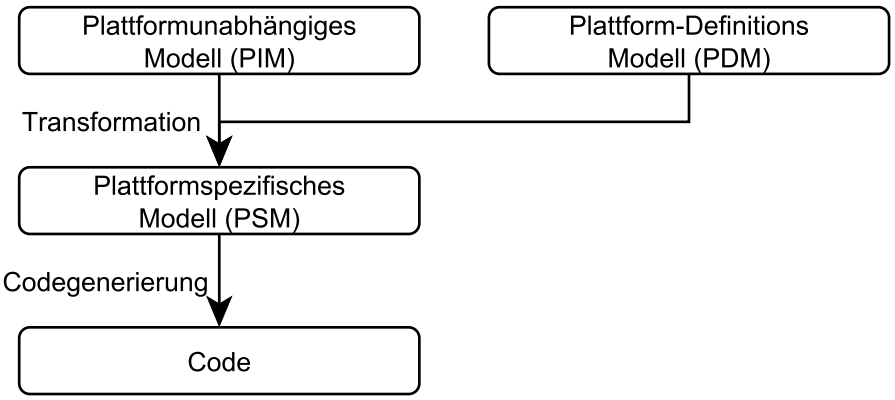


Abbildung 2.6: Architektonische Trennung von fachlichen und technischen Systemaspekten in der MDA nach Miller und Mukerji (2003, S. 2-7)

Ein komplexes Softwaresystem setzt letztendlich auf einer Kaskade unterschiedlich abstrakter Zielplattformen auf. Zur Umsetzung und weiten Verbreitung der MDA stellt die Object Management Group unentgeldlich relevante Standards zur Verfügung, welche nachfolgend erläutert werden.

Die **Meta Object Facility (MOF)** definiert eine Architektur zur Erstellung und Verwaltung von Metamodellen auf mindestens zwei Abstraktionsebenen (OMG 2013b, S. 9). Vor allem in Verbindung mit der im nächsten Abschnitt beschriebenen UML werden aber vier Abstraktionsebenen angenommen, die in Abbildung 2.7 dargestellt sind.

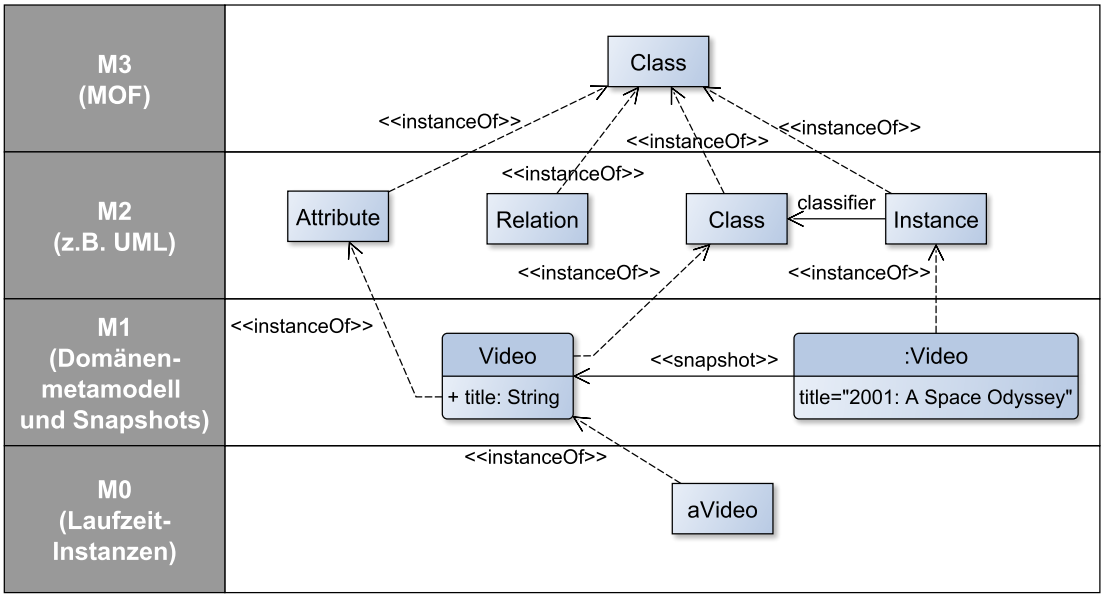


Abbildung 2.7: Die vier Abstraktionsebenen einer Metamodell-Hierarchie nach OMG (2011d, S. 20)

Auf Ebene M3 werden abstrakte Kernkonzepte und Sprachkonstrukte der MOF spezifiziert, die zur Definition von (linguistischen) Metamodellen auf Ebene M2 benötigt werden, z. B. der in Abbildung 2.8 gezeigten Spracheinheiten der Unified Modeling Language. Repräsentationssprachen der Ebene M3 werden daher auch als Meta-Metamodell bezeichnet. Ebene M1 enthält einerseits ontologische Metamodelle der betrachteten Domäne sowie daraus zur Entwurfszeit initial erstellte oder zur Laufzeit persistierte Momentaufnahmen als Instanzen. Laufzeitinstanzen eines ontologischen Metamodells innerhalb des flüchtigen Speichers eines Softwaresystems werden auf Ebene M0 verortet. Darüber hinaus bietet die MOF die Möglichkeit zur Modellreflexion und eindeutigen Identifizierung von Modellelementen und -instanzen, sowie einen Erweiterungsmechanismus in Form von Schlüssel-Wert-Paar-basierten Modellannotationen. Die MOF ist modular aufgebaut. Die **Essential MOF** stellt den kleinsten gemeinsamen Nenner dar, um einfache Metamodelle mit Hilfe der MOF zu spezifizieren. So ist die Essential MOF beispielsweise die Grundlage für die Implementierung des im Eclipse Modeling Framework (EMF) enthaltenen Meta-Metamodells **Ecore** (Steinberg u. a. 2009), welches auch in dieser Arbeit zur Erstellung von Metamodellen verwendet wurde (Kapitel 6, S. 111). Im Gegensatz dazu stellt die **Complete MOF** den kompletten Funktionsumfang der MOF zur Verfügung und wird etwa zur Definition der Unified Modeling Language verwendet. Auf Basis der Extensible Markup Language (XML) können Modelle aller Ebenen mit Hilfe des Standards **XML Metadata Interchange** (OMG 2011c) serialisiert und somit zwischen verschiedenen Werkzeugen ausgetauscht werden.

Die **Unified Modeling Language (UML)** „*ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme.*“ (Oestereich 2009, S.241) Die Spezifikation der UML selbst ist dabei zweigeteilt. In der UML Infrastruktur (OMG 2011d) wird das grundlegende Metamodell auf Basis der MOF definiert. Die UML Superstruktur (OMG 2011e) beinhaltet diverse Spracheinheiten zur Spezifikation eines Systems. Die Spracheinheiten sind abhängig von ihrer Ausdrucksfähigkeit einer oder mehreren Schichten zugeordnet. Auf diese Weise wird etwa die Erstellung von Werkzeugen für die UML erleichtert, da diese nur einen gewissen Teil einer Spracheinheit abdecken müssen. Die Spracheinheiten der UML stellen deren abstrakte Syntax dar, die durch Diagramme als grafische Notation bzw. konkrete Syntax visualisiert werden. Abbildung 2.8 gliedert diese Diagramme hierarchisch nach Einsatzzweck und unterscheidet dabei zwischen Struktur- und Verhaltensdiagrammen.

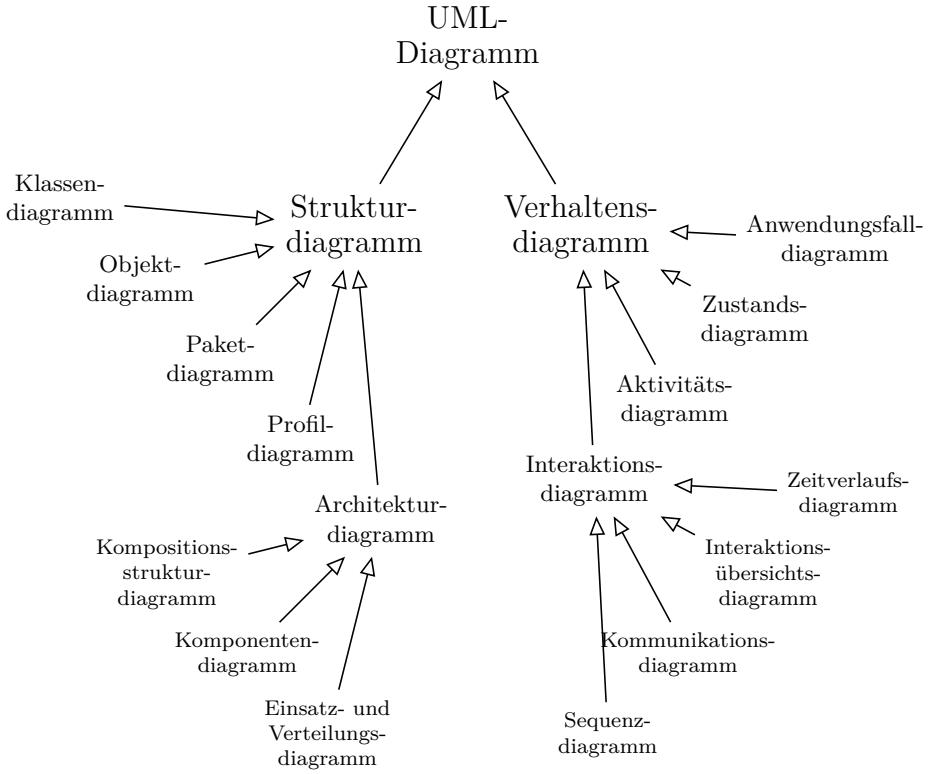


Abbildung 2.8: Hierarchische Gliederung der UML-Spracheinheiten und Diagramme nach Einsatzzweck

Außerdem ermöglicht die UML Superstruktur durch einen Adaptionsmechanismus in Form von Profilen die Erweiterung von UML-Metamodellen an bestimmte Domänen, Plattformen oder (Entwicklungs-) Methoden (OMG 2011e, S. 659f.).

Basierend auf der MOF und der UML erlaubt die **Object Constrained Language**

(OCL) (OMG 2012) die Spezifikation feingranularer Nebenbedingungen und Invarianten für Elemente eines Metamodells sowie deren Beziehungen untereinander, die von allen Instanzen eingehalten werden müssen. Durch die OCL kann die statische Semantik eines Metamodells (Kapitel 2.4.1, S. 18) deutlich herausgearbeitet und automatisiert überprüfbar gemacht werden, da etwa der gültige Wertebereich einzelner Elementeigenschaften formal auf ein plausibles Maß eingeschränkt werden kann.

Die **Query View Transformation (QVT)** (OMG 2011b) dient zum Spezifizieren von M2M-Transformationen von Instanzen MOF-basierter Metamodelle. Hierbei dient OCL als Mechanismus zur Selektion von Elementen aus einem Modell sowie zur Konsistenzsicherung der Transformation. Weiterhin unterscheidet man zwischen der relationalen und der operationalen QVT, die sich nicht nur in ihrer konkreten Syntax unterscheiden. Während die relationale QVT eine bidirektionale Transformation von Modellen erlaubt, kann eine Transformation auf Basis der operationalen QVT nur in eine Richtung durchgeführt werden. Schließlich ermöglicht die **MOF Model zu Text Transformation** (OMG 2008) eine M2C-Transformation von MOF-basierten Modellen in textuellen Quellcode.

2.6 Fazit

Dieses Kapitel hat die theoretischen Grundlagen der Modellbildung anhand der Allgemeinen Modelltheorie erklärt und deren Bedeutung für die Informatik aufgezeigt. Auf dieser Basis wurde die modellgetriebene Softwareentwicklung motiviert, deren Ziel es ist, aus formalen Modellen automatisiert Software zu erzeugen. Die Modellgetriebene Architektur definiert Standards zur Umsetzung der modellgetriebenen Softwareentwicklung. Das Eclipse Modeling Framework kann im Rahmen der Eclipse-Plattform als Referenzimplementierung der Modellgetriebenen Archhitektur in der Programmiersprache Java angesehen werden.

Auch im Kontext dieser Arbeit ist die Modellklassifikation nach der Allgemeinen Modelltheorie relevant. Die Modelle für die computerunterstützte Zusammenarbeit und allgemein Modelle für die Mensch-Computer-Interaktion können den technischen, speziell den psycho- und soziotechnischen Modellen zugeordnet werden. Die in Kapitel 7 (S. 143) vorgestellte modellgetriebene Methode bewegt sich in der Klassifikation von Jockisch und Rosendahl (2009) also im Rahmen der soziotechnischen Systeme ausgehend von der umfassenden Betrachtungsebene konsequent zur spezifischen. Dazu werden die in Kapitel 6 (S. 111) vorgestellten Modelle in Kapitel 7 (S. 143) durch entsprechende Modelltransformationen von einer Abstraktionsebene zur nächst tieferen transformiert. Technisch werden hierzu Java-basierte Werkzeuge der Eclipse-Plattform – insbesondere das Eclipse Modeling Framework – eingesetzt. Die resultierende KomBInoS-Werkbank ist in Kapitel 8 (S. 187) beschrieben. Damit erzeugte multimodale Dialogschnittstellen für Smart Services gehören aufgrund derselben Modell- bzw. Domänenarchitektur der selben Softwaresystemfamilie an – mit allen in Kapitel 2.4.3 (S. 22) geschilderten Vor- und Nachteilen.

Die Networked European Software and Services Initiative beobachtet, dass die industrielle Wertschöpfung vermehrt auf Ebene der Softwaresysteme stattfindet. Dementsprechend entsteht ein besonderer Bedarf an effizienten und effektiven modellgetriebenen Ansätze zur Softwareentwicklung (NESSI 2014, S. 5), insbesondere im Rahmen der Smart Service Welt. Das nächste Kapitel führt in dieses Themenfeld ein.

KAPITEL 3

Dienste in der Smart Service Welt

In diesem Kapitel wird einleitend die Vision der Smart Service Welt erläutert. Nach einer Begriffsbestimmung wird anschließend ein besonderes Augenmerk auf digitale Dienste als ein Bestandteil von Smart Services gelegt. Hierzu werden die Referenzarchitektur der Smart Service Welt sowie der Lebenszyklus von Diensten diskutiert. Es werden außerdem die zur Realisierung von Diensten notwendigen Schlüsseltechnologien detailliert erläutert.

3.1 Einleitung

Die zunehmende Digitalisierung ermöglicht neue internetbasierte Geschäftsmodelle und eine Globalisierung von IT-gestützten Dienstleistungen. In der Konsequenz entwickeln sich dynamische Wertschöpfungsnetze mit einer hohen Innovationsgeschwindigkeit. Unter dem Schlagwort *Zukünftiges Internet* kann man Forschungsprogramme und Initiativen verstehen, die sich mit neuen Anwendungsfeldern des Internets sowie den Potentialen und Auswirkungen der verstärkten Internetnutzung und Digitalisierung aus technologischer, wirtschaftlicher und gesellschaftlicher Perspektive befassen.

Das Zukunftsprojekt Smart Service Welt (Acatech 2014, 2015) fokussiert auf die Optimierung industrieller Prozesse über die gesamte vernetzte Wertschöpfungskette. Dazu werden zahlreiche Forschungsgebiete, etwa das Internet der Dinge, cyber-physische Systeme, effiziente Verfahren zum Umgang mit großen Datenmengen (Big Data) sowie das Internet der Dienste, mit dem Ziel der Entwicklung und Bereitstellung von Smart Services gebündelt. Definition 4 führt die wesentlichen Aspekte von Smart Services auf. Im Vordergrund stehen jedoch nicht das Produkt oder die Dienstleistung, sondern der Nutzer von Smart Services. Aufgrund der Komplexität muss der Zugang zu Smart Services für Nutzer über robuste und intuitive intelligente Benutzerschnittstellen erfolgen (Acatech 2015, S. 78).

Definition 4: Smart Services **(Acatech 2017)**

„Smart Services sind über das Internet individuell konfigurierte Pakete aus Produkten, Dienstleistungen und Diensten. Die privaten und gewerblichen Nutzer stehen dabei im Mittelpunkt. Mithilfe digitaler Daten aus allen Lebensbereichen werden Smart Services auf ihre Vorlieben bedarfsgerecht und situationsspezifisch 'as a Service' zugeschnitten. Eine zentrale Rolle spielen digitale Plattformen: Hier werden Produkte und Dienstleistungen virtuell abgebildet, kombiniert, mit zusätzlichen digitalen Diensten veredelt und als Smart Service angeboten.“

Im weiteren Verlauf wird in Kapitel 3.2 (S. 30) zunächst die Referenzarchitektur der Smart Service Welt in Kombination mit dem datengetriebenen Wertstrom erläutert. Von besonderem Interesse für diese Arbeit sind die digitalen Dienste eines Smart Service. In Kapitel 3.3 (S. 33) wird der Lebenszyklus eines solchen Dienstes beschrieben, der sich ohne Weiteres auf einen umfassenden Smart Service ausdehnen lässt. Schließlich werden in Kapitel 3.4 (S. 35) Schlüsseltechnologien aufgezeigt, die notwendig sind, um digitale Dienste eines Smart Service umzusetzen.

3.2 Referenzarchitektur der Smart Service Welt

Abbildung 3.1 (links) illustriert die Referenzarchitektur der Smart Service Welt. Diese besteht aus vier Schichten, um die herum sich digitale Ökosysteme und innovationsorientierte Rahmenbedingungen bilden.

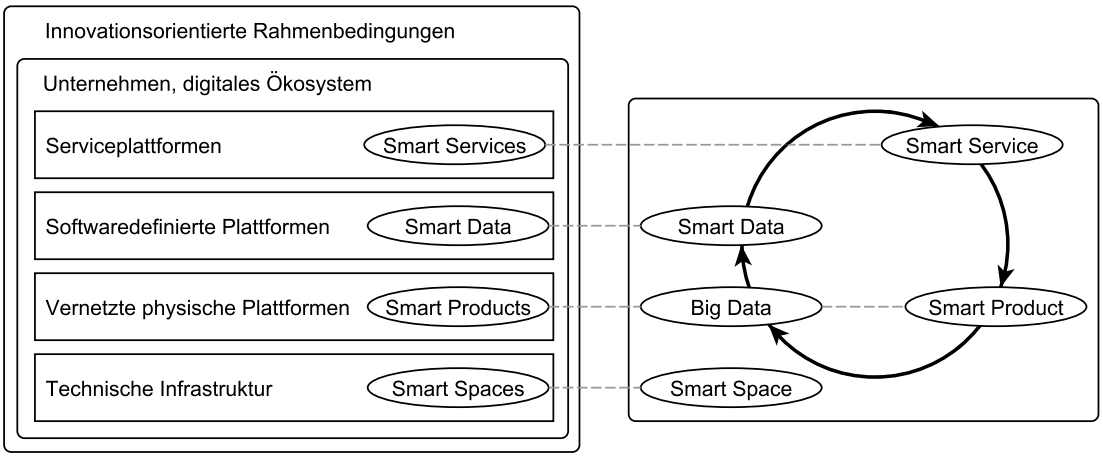


Abbildung 3.1: Die Referenzarchitektur der Smart Service Welt (links) im datengetriebenen Wertstrom (rechts) nach Acatech (2015, S. 17)

Die **Serviceplattform** stellt eine Kollaborationsumgebung zur Verfügung, auf deren Basis sowohl der Endnutzer die Möglichkeit hat, neuartige Smart Services zu finden,

einzukaufen und zu benutzen oder in Anspruch zu nehmen, als auch der Dienstanbieter dazu befähigt wird, aus bestehenden Diensten neue Smart Services zu schaffen. Serviceplattformen dienen somit als Kristallisationspunkt für neue digitale Wertschöpfungsketten. Durch Prozessdefinitionen, Schnittstellenbeschreibungen, Standards und Werkzeuge geben Serviceplattformen einen formalen Rahmen für die (betriebswirtschaftlich getriebene) Entwicklung von Smart Services auf Basis der zugrundeliegenden softwaredefinierten Plattformen.

Die **softwaredefinierte Plattform** stellt die virtuelle Laufzeitumgebung für Smart Services zur Verfügung. Sie beinhaltet die grundlegenden Basistechnologien und Bestandteile, die für den Betrieb von Smart Services und die Verknüpfung mit der realen Welt, den intelligenten Produkten und deren Anwendern notwendig sind. Hierzu können leicht konfigurierbare, generische technologische Befähiger und höherwertige zusammengesetzte Leistungsbündel als Bausteine wiederverwendbar eingesetzt werden. Insbesondere spielen eine semantische Dienstintegration sowie effiziente Mechanismen zum Auffinden und Orchestrieren von Smart Services eine wichtige Rolle. Aus Sicht von Benutzerschnittstellen für Smart Services sind aber vor allem technologische Bausteine für eine ubiquitäre Benutzermodellierung, Serviceorchestrierung, Servicebeschreibung- und Auffindung, für Context Brokering sowie für Alerting Management und Kollaboration essentiell.

Zur nahtlosen Integration digitaler Dienste mit intelligenten Produkten und physischen Dienstleistungen in einer cyber-physischen Umgebung sind zudem **vernetzte physische Plattformen** auf Grundlage einer adäquaten **technischen Infrastruktur** erforderlich.

Ein Blick auf den in Abbildung 3.1 (rechts) gezeigten datengetrieben Wertstrom verdeutlicht den zyklischen Daten- und Informationsfluss über die geschichteten Plattformen der Referenzarchitektur. Intelligente Produkte als Bestandteil eines Smart Service sind in eine intelligente Umgebung eingebettet und produzieren eine potentiell sehr große Datenmenge. Diese Daten werden durch digitale Dienste innerhalb einer softwaredefinierten Plattform zu Smart Data wertschöpfend aggregiert und wiederum darauf aufbauenden, gegebenenfalls anderen Smart Services zur Verfügung gestellt.

Das Real-time Smart Farming Services-Rahmenwerk (Maaß u. a. 2017) stellt eine Instanziierung der Referenzarchitektur zur Umsetzung datengetriebener Smart Services für die landwirtschaftliche Domäne bereit. Am Beispiel einer Kartoffelernte wird gezeigt, wie ein intelligenter Sensor in Form und Größe einer durchschnittlichen Kartoffel im Zusammenspiel mit Cloud-basierten Analyse- und Vorhersagediensten den Ernteprozess in Echtzeit schonender und effizienter gestalten sowie den monetären Ertrag maximieren kann.

Im IKS-Projekt (Janzen u. a. 2011) wurde auf Basis semantischer Technologien ein intelligentes Badezimmer als personalisiertes ubiquitäres Informationssystem umgesetzt. Mitglieder eines Haushaltes konnten sich bei der Nutzung des instrumentierten Badezimmers gemäß ihrer Interessen über relevante Ereignisse im multimodalen Dialog informieren und im gleichen Zuge etwa Eintrittskarten für ein bevorstehendes Konzert

oder die abendliche Filmvorführung bestellen. Trotz der angebotenen Funktionalität für den Benutzer stellt das intelligente Badezimmer ein geschlossenes System dar, welches die Einbringung weiterer Geräte und Dienste im laufenden Betrieb nicht vorsieht. Eine prototypische Instanziierung der Referenzarchitektur der Smart Service Welt für den Bereich Smart Home wurde von Frey (2015) erstmals im Rahmen einer agentenbasierten Integrationsplattform für Smart Services umgesetzt. Sie ist offen, hochflexibel, erweiterbar und auf Standards basierend. Abbildung 3.2 illustriert die zugrundeliegende Plattformarchitektur.

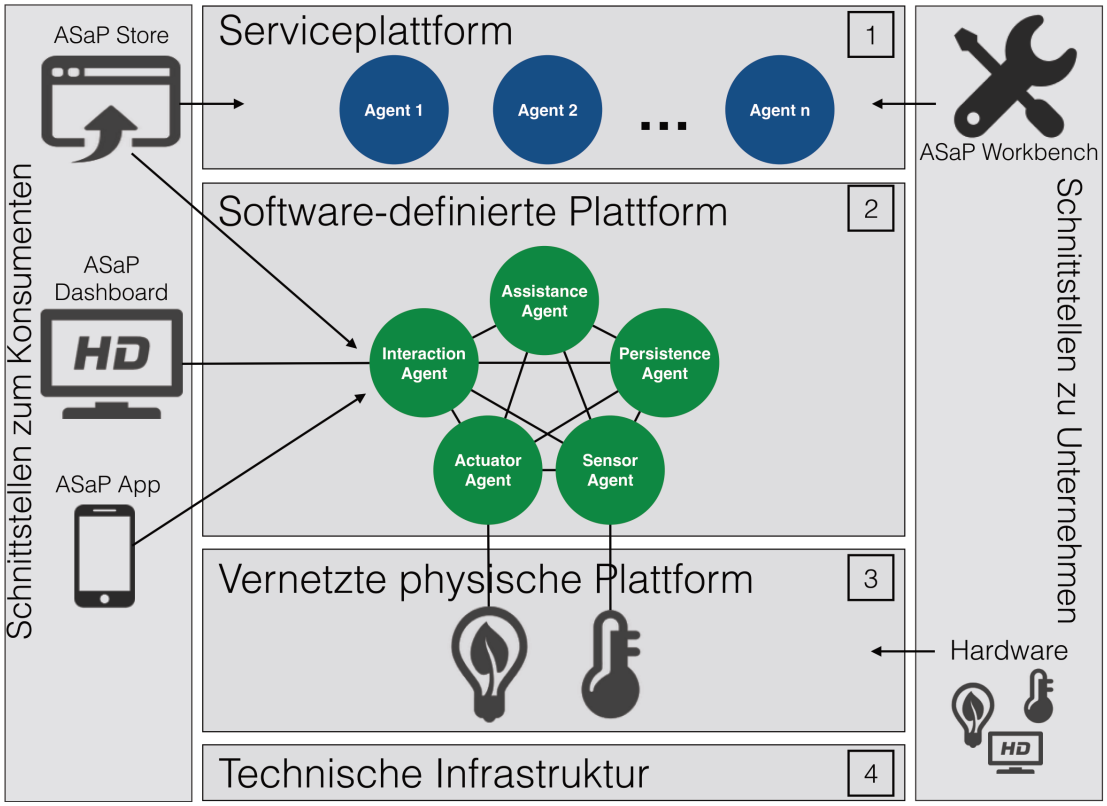


Abbildung 3.2: Die Architektur von Freys (2015, S. 135) agentenbasierter Integrationsplattform für Smart Services

Innerhalb einer vernetzten physischen Plattform werden intelligente Geräte als Kombination von Sensorik und Aktorik über als Agenten realisierte Dienste ausgelesen und gesteuert. Ein Nutzer hat über dedizierte grafische Benutzerschnittstellen Zugriff auf die Agenten der softwaredefinierten Plattform. Mit Hilfe des ASaP-Marktplatzes der Serviceplattform kann der Nutzer die Zusammensetzung *seiner* softwaredefinierten Plattform sowie *seiner* vernetzten physischen Plattform jederzeit anpassen. So wird beispielsweise mit dem Einkauf eines neuen vernetzten Rauchmelders auch der dazugehörige Agent zur Installation bereitstellt.

Auf der Grundlage der diskutierten Referenzarchitektur und des Wertstroms können nun unter Einbeziehung aller beteiligten Akteure und Intermediäre – vom Plattformbetreiber und Dienstanbieter bis zum Dienstnutzer – eine Vielzahl weiterer innovativer Geschäftsmodelle und Wertschöpfungsnetze entstehen.

3.3 Der Lebenszyklus von Diensten

Der Anwendungsfall TEXO hat als Teil des THESEUS Forschungsprogramms Konzepte, Methoden und Werkzeuge entwickelt, um neue digitale Geschäftsmodelle im Internet der Dienste umsetzen zu können (Kuhlmann u. a. 2014, S. 281). Solche Geschäftsmodelle basieren nicht mehr auf starren Wertschöpfungsketten, sondern vielmehr auf flexiblen und dynamischen Wertschöpfungsnetzen, in denen unterschiedliche Partner ad-hoc zusammenarbeiten, um schnell eine neue Dienstleistung anbieten zu können. Die Wertschöpfung findet dabei entlang des in Abbildung 3.3 dargestellten Lebenszyklus von Diensten statt, welcher sich auch auf umfassende Smart Services übertragen lässt.

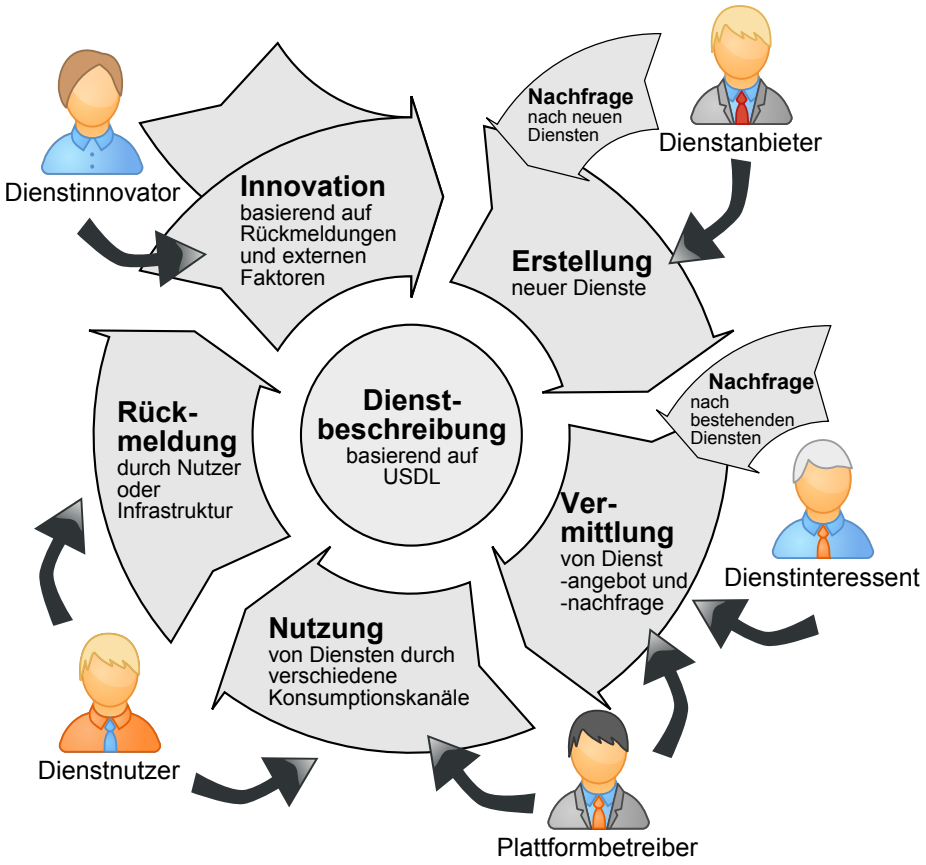


Abbildung 3.3: Der Dienstelebenszyklus nach Kuhlmann u. a. (2014, S. 283)

Der Lebenszyklus besteht aus fünf aufeinander aufbauenden Phasen und verknüpft mit ihnen unterschiedliche Rollen, welche von handelnden Personen oder Organisationen innerhalb einer Phase übernommen werden können. Ergebnisse aus TEXO ermöglichen es, digitale Diente und darauf aufbauende elektronische Dienstleistungen im Rahmen des Lebenszyklus auffindbar, nutzbar, komponierbar und schließlich handelbar zu machen. Damit zeichnete TEXO erstmals eine Blaupause für zukünftige Diensteplattformen und trägt so nachhaltig zur weiteren Digitalisierung bei.

(1) **Innovation.** Ein solides Innovationsmanagement ist eine wichtige Grundvoraussetzung, um wettbewerbsfähig zu werden bzw. zu bleiben. In der Innovationsphase werden darum auf Basis einer Ideen-Ontologie (Riedl u. a. 2009) Ideen strukturiert gesammelt, bewertet und weiterentwickelt. Ideen haben ihren Ursprung z. B. in Nutzerfeedback zu vorhandenen Diensten, in Auswertungen von Suchanfragen aus der Vermittlungsphase oder in Marktbeobachtungen durch Trend- und Meinungsanalysen (Li u. a. 2011).

(2) **Erstellung.** Ausgereifte Ideen müssen schnell umgesetzt und an den Markt gebracht werden. In der Erstellungsphase werden abstrakte Ideen zu konkreten Dienstimplementierungen weiterentwickelt. Das Integrated Service Engineering (ISE) Rahmenwerk (Kapitel 5.4.1, S. 101) definiert dazu eine Entwicklungsmethodik, die alle relevanten Facetten eines Dienstes berücksichtigt und gleichzeitig besonders auf die organisatorischen Strukturen in größeren Softwareunternehmen eingeht. Zusätzlich können Teilaufgaben des neuen Dienstes durch Einbinden bereits existierender Dienste realisiert werden. Dadurch wird letztlich ein gewisser Mehrwert generiert. Nach der Fertigstellung eines Dienstes wird dieser auf einer oder mehreren Dienstmarktplätzen angeboten. Hierbei spielt eine strukturierte Dienstbeschreibung die zentrale Rolle, damit der Dienst in der nächsten Phase auffindbar und vergleichbar wird.

(3) **Vermittlung.** Potentielle Dienstnutzer bzw. -interessenten haben ganz bestimmte funktionale und nichtfunktionale Anforderungen an Dienste. In der Vermittlungsphase erfolgt ein Abgleich der Anforderungen mit den angebotenen Diensten anhand der Dienstbeschreibung. Ein zentraler Meta-Suchdienst auf Basis des quelloffenen SMILA Rahmenwerks[1] für die Verarbeitung von (un-) strukturierten großen Datenmengen kapselt dazu unterschiedlichste (semantische, formularbasierte und textuelle) Suchverfahren und aggregiert zurückgelieferte Ergebnismengen in gewissem Umfang entsprechend. Mindestens genauso wichtig sind aufgrund der hochdimensionalen Dienstbeschreibung adäquate Benutzerschnittstellen zum Formulieren von Anforderungen in Form von Suchanfragen und Präsentieren von Ergebnismengen. Neben bekannten, einfachen Listendarstellungen können die Resultate für einen besseren Überblick auch grafisch arrangiert und über multimodale dialogische Interaktion auch auf mobilen Endgeräten sortiert und gefiltert werden (Porta u. a. 2009b). Zusätzlich zum Suchen und Finden von Diensten zählt

[1] siehe http://www.eclipse.org/smila/ (letzter Zugriff: 04.11.2017)

auch der nachgelagerte Einkauf von Diensten zur Vermittlungsphase. Geeignete Werkzeuge unterstützen den Kunden bei der Ausgestaltung von Dienstgütevereinbarungen, einer juristischen Kompatibilitätsprüfung, der Preisfindung sowie beim Vertragsschluss (Spillner u. a. 2009).

(4) **Nutzung.** Sobald ein Dienst eingekauft wurde, steht dieser zur Nutzung zur Verfügung. Zur technischen Bereitstellung, Ausführung und Überwachung von Diensten sind softwaredefinierte Plattformen wie die FLIGHTspace-Plattform (Winkler u. a. 2009) oder Freys (2015) agentenbasierte Integrationsplattform für Smart Services notwendig. Darüber hinaus umfasst die Nutzungsphase aber auch die Interaktion eines Dienstes mit einem oder mehreren Benutzern. In TEXO wurden hierzu bereits drei verschiedene Nutzungskanäle, also mögliche Wege der Dienstnutzung durch einen Benutzer, untersucht. Prinzipiell können Dienste als eigenständige, stationäre reichhaltige Internetanwendung oder integriert in eine bestehende, emergente Geschäftsanwendung (Heller und Allgaier 2010) oder, wie im Fall der vorliegenden Arbeit, mobil konsumiert werden.

(5) **Rückmeldung.** Nach der Verwendung eines Dienstes kann der Nutzer seine Meinung z. B. in Form von strukturierten Bewertungen oder natürlichsprachlichen Forenbeiträgen äußern. Dieser nutzergenerierte Inhalt liefert wertvolle Anregungen und Ideen für die Weiterentwicklung eines bestehenden Dienstes bzw. die Entwicklung eines neuen Mehrwertdienstes. Damit schließt sich der Kreis.

(6) **Dienstbeschreibung.** Allen bisher beschriebenen Phasen des Dienstlebenszyklus liegt eine gemeinsame Dienstbeschreibungssprache zugrunde. Als umfassende Dienstbeschreibungssprache kann die Unified Service Description Language (USDL) dienen, welche in Kapitel 5.4.2 (S. 103) als verwandte Arbeit beschrieben wird.

3.4 Schlüsseltechnologien

Wichtige Schlüsseltechnologien zur Realisierung internetbasierter Dienstleistungen sind nach Dufft u. a. (2010) auf technischer Ebene **Webdienste**, die punktuell von teilweise monolithischen und historisch gewachsenen Altsystemen abstrahieren und diese über eine definierte Schnittstelle in einem Netzwerk verfügbar machen. Innerhalb einer **dienstorientierten Architektur** werden diese technischen Dienste gebündelt und einheitlich verwendbar, sodass sie als wiederverwendbare Bausteine im Rahmen von **Geschäftsprozessen** genutzt und miteinander verschaltet werden können. **Cloud Computing** liefert letztlich die technologische Grundlage zum Auslagern und flexiblen Bereitstellen von benötigten IT-Ressourcen. Im Folgenden werden diese grundlegenden Konzepte und Technologien näher erläutert.

3.4.1 Webdienste

Ältere Definitionen für den Begriff Websdienst, z. B. die vom W3C (Booth u. a. 2004), stellen vor allem die Relevanz von XML-basierten Technologien und darauf aufbauenden Standards heraus. Die Entwicklungen der letzten Jahre erfordern es jedoch, eine solche Definition unabhängiger von konkreten Technologien zu verfassen, um diese mit einzuschließen. Eine solche Formulierung bietet Definition 5, welche die für diese Arbeit wesentlichen Merkmale erfasst.

Definition 5: Webdienst in KomBInoS

Ein Webdienst ist eine Softwarekomponente, die eine Funktionalität über ein Netzwerk, etwa über das Internet oder das lokale Firmennetzwerk, unter Verwendung von standardisierten formalen Beschreibungssprachen sowie standardisierten Protokollen und Formaten zum Datenaustausch bereitstellt.

Heute unterscheidet man auf technischer Ebene hauptsächlich zwischen SOAP-basierten und RESTful Webdiensten (Bean 2009). In einer qualitativen Gegenüberstellung vergleichen Pautasso u. a. (2008) SOAP-basierte und RESTful Webdiensten u. a. anhand architektonischer Prinzipien miteinander. Sie kommen zu dem Schluss, dass sich RESTful Webdienste sehr gut in einfachen, ad-hoc Integrationsszenarien eignen. SOAP-basierte Webdienste sind hingegen gerade im Geschäftsumfeld aufgrund ihrer größeren Flexibilität und Möglichkeiten hinsichtlich Dienstgüte besser geeignet. Im Rahmen dieser Arbeit wurden sowohl SOAP-basierte als auch RESTful Webdienste für unterschiedliche Zwecke eingesetzt (Sonntag u. a. 2010a). Darüber hinaus eröffnen semantische Webdienste die Möglichkeit, einer automatisierten, planbasierten Orchestrierung ebendieser zu komplexeren Mehrwertdiensten.

3.4.1.1 SOAP-basierte Webdienste

Das Simple Object Access Protocol (SOAP) ist ein vom W3C (Mitra und Lafon 2007) standardisiertes Nachrichtenformat auf Basis von XML. Dadurch wird ein automatisches (Un-) Marshalling, also die Überführung einer SOAP-Nachricht in ein Objektmodell (und zurück), möglich. Eine SOAP-Nachricht besteht aus einem XML-Wurzelelement namens envelope, welches einen header und einen body umfasst. Der body enthält die Nutzdaten der Nachricht. Im header kann etwa Information für das Routing der Nachricht oder für Dienstgüte-Aspekte abgelegt werden. Relevante Spezifikationen diesbezüglich wie WS-Addressing, WS-Realiable Messaging oder WS-Policy sind Bestandteil der Sammlung dienstbezogener Spezifikationen WS-*. WS-* ermöglicht eine sehr flexible Nutzung SOAP-basierter Webdienste, steigert zugleich aber auch deren Komplexität. Die technische Schnittstelle eines SOAP-Webdienstes ist durch die Web Service Description Language (WSDL) (Chinnici u. a. 2007) formal beschrieben. Eine solche Schnittstelle spezifiziert eine Menge von abstrakten Operationen. Eine Operation kann synchron oder asynchron aufgerufen werden und besteht aus einer Anfrage-

und je nach Aufrufmuster aus einer Antwortnachricht. Die Nachrichten enthalten XML-basierte Datenstrukturen, die ebenfalls innerhalb der Dienstbeschreibung in Form von XML-Schema spezifiziert sind. Die abstrakten Operationen werden an ein oder mehrere konkrete Transportprotokolle, wie etwa das Hypertext Transfer Protocol (HTTP) und Datenformate, z. B. SOAP Version 1.2, gebunden. Eine Bindung ist wiederum mit einem Endpunkt, bestehend aus einem Uniform Resource Identifier (URI) und einer Portangabe, verknüpft, der das Transportprotokoll und das Datenformat unterstützt.

3.4.1.2 RESTful Webdienste

Während bei SOAP-basierten Webdiensten eher Operationen im Vordergrund stehen, stellt Representational State Transfer (REST) von Fielding (2000) einen Ressourcen-zentrierten Architekturansatz zur Erstellung von Webdiensten dar. Dabei kommen Konzepte und Technologien zum Einsatz, die auch dem World Wide Web zugrundeliegen. Mit Hilfe der HTTP-Operationen GET, PUT, POST und DELETE können Ressourcen abgefragt, angelegt, modifiziert und gelöscht werden. Die im Fokus einer Anfrage stehende Ressource wird eindeutig anhand eines URIs adressiert. Die konkrete syntaktische Repräsentation (Kapitel 2.4.1, S. 18) einer Ressource kann über die Angabe des gewünschten Internet Media Types, aus historischen Gründen oft auch noch als Multipurpose Internet Mail Extensions-Type bezeichnet, gesteuert werden. So ist es z. B. möglich, die gleiche Ressource einmal als menschenlesbare Webseite in der Hypertext Markup Language (HTML), als maschinenverarbeitbares XML-Dokument oder in effizienter JavaScript Object Notation (JSON) darzustellen. RESTful Webdienste arbeiten in der Regel zustandslos, d.h. die zur Beantwortung einer Anfrage notwendige Information muss vollständig in der Anfrage an den Dienst enthalten sein.

3.4.1.3 Semantische Webdienste

Semantische Webdienste verfügen über eine semantische Beschreibung ihrer Funktionalität und Daten. Hierbei kommen ausdrucksstarke Repräsentationsformalismen wie OWL, OWL für Services oder die Web Service Modeling Ontology zum Einsatz (Kapitel 2.3, S. 14). Dadurch ergibt sich, wie z. B. im Projekt SmartWeb (Sonntag u. a. 2007) gezeigt, die Möglichkeit, Dienste kontextabhängig automatisiert zu orchestrieren. Dies beinhaltet u. a. sowohl die Planung als auch die koordinierte Nutzung der Orchestrierung (Klusch 2008). Hierbei stellt sich generell das Problem des Ontologiematchings (Pavel und Euzenat 2013), wofür etwa Do (2012) mit COMA++ eine Lösung in Form von adäquaten Algorithmen und einer Benutzerschnittstelle anbietet. Die technische Abbildung der semantischen Dienstbeschreibung auf eine technische Dienstschnittstelle greift üblicherweise auf SOAP-basierte Webdienste zurück.

Parallel dazu machen semantische Wissensbasen Information via Netzwerk zugänglich. Über eine SPARQL-Schnittstelle können semantische Daten in Form von RDF-

Tripeln zur weiteren Verarbeitung abgefragt werden. Das Jena-Rahmenwerk[2] bietet eine solche Funktionalität und kann in Verbindung mit unterschiedlichen Inferenzmaschinen betrieben werden. Eine Anwendung von Jena findet beispielsweise im System Tip'n Tell (Maass und Filler 2007) statt. Hierin können zu einem Smart Product auf Basis semantischer Produktbeschreibungen und Inferenzregeln ähnliche und ergänzende Produkte sowie unterstützende Multimediainhalte gefunden werden.

3.4.1.4 Mashup-Technologien

Laut dem Gabler Wirtschaftslexikon (2017) bezeichnen Mashups eine „*Kombination vorhandener Dienste im Internet, durch die ein weiterer Mehrwert geschaffen wird. Unterschiedliche Datenbestände zweier Dienste werden zusammengebracht, um so Zusatzinformationen zu generieren. Ermöglicht wird dies durch mehr oder weniger offen liegende Schnittstellen.*“ Die Erstellung von Mashups erfolgt mittels (semantischer) Webtechnologien, teilweise unterstützt durch grafische Editoren (Endres-Niggemeyer 2013). Vancea u. a. (2008) unterscheiden prinzipiell zwischen Daten-Mashups zur Integration und Aggregation unterschiedlicher Datenströme sowie UI-Mashups zur gemeinsamen Darstellung von Daten aus unterschiedlichen Quellen. Daniel und Matera (2014) fassen die wesentlichen Konzepte, Modelle und Architekturen von Mashups anschaulich zusammen.

3.4.2 Diensteorientierte Architektur

Eine dienstorientierte Architektur (SOA, Service-oriented Architecture) ist eine konzeptuelle Softwarearchitekur mit dem Ziel, Dienste über ein Netzwerk anbieten, suchen und nutzen zu können. Dahinter verbirgt sich die Notwendigkeit, Funktionalität monolithischer und heterogener Softwareanwendungen in Diensten zu kapseln, um diese einheitlich wiederverwendbar zu machen. Dadurch können Softwaresysteme besser modularisiert und vor allem Geschäftsprozesse flexibler realisert und angepasst werden. Eine SOA wird technisch mit Hilfe von Webdiensten realisiert.

Die erstmalige Verwendung des Begriffs SOA wird laut Abrams und Schulte (2008, S. 5) dem amerikanischen Marktforschungsunternehmen Gartner im Jahre 1996 zugeschrieben. Eine Definition für eine SOA ist stark blickwinkelabhängig. Je nach Kontext werden betriebswirtschaftliche und/oder technische Aspekte in den Vordergrund gestellt. Das Standardisierungsgremium OASIS hat zur Schaffung eines gemeinsamen Verständnisses ein Referenzmodell einer SOA (OASIS 2006, S. 8) erarbeitet. Darin wird eine SOA bezeichnet als „*Paradigma für die Organisation und Nutzung von verteilten Funktionen, die unter der Kontrolle von unterschiedlichen Besitzern verantwortet werden können.*“ Ebenso werden grundlegende Merkmale einer SOA identifiziert. Melzer und Eberhard (2008, S. 13) verstehen unter einer SOA eine „*Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wie-*

[2] siehe https://jena.apache.org/ (letzter Zugriff: 04.11.2017)

derverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht.“

Dienste in einer SOA sind dezentral in einem Netzwerk verteilt. Um sie jedoch auffindbar zu machen, ist laut Melzer und Eberhard (2008, S. 10ff.) ein **zentraler Verzeichnisdienst** notwendig, der Dienste auf Basis einer **standardisierten Dienstbeschreibung** erfasst und vergleichbar macht. Dazu bietet die Dienstbeschreibung die Möglichkeit, sowohl funktionale als auch nichtfunktionale Eigenschaften eines Dienstes zu erfassen. Funktionale Eigenschaften beschreiben jenseits der technischen Schnittstellenbeschreibung, semantisch was ein Dienst leistet. Nichtfunktionale Eigenschaften garantieren die Bedingungen, unter denen ein Dienst arbeitet und werden dazu eingesetzt, eine Dienstgütevereinbarung zwischen dem Dienstanbieter und einem Dienstnutzer zu schließen. Zur Kontrolle der Dienstgütevereinbarung ist zur Laufzeit eine Überwachung und Messung der Dienstausführung notwendig. Üblicherweise werden mehrere Dienste miteinander zu einem größeren Prozess verknüpft. Innerhalb eines Prozesses können die Dienste lose gekoppelt sein. Das bedeutet, dass erst während der Prozessausführung durch Zugriff auf den Verzeichnisdienst der konkret auszuführende Dienst unter Berücksichtigung von etwaigen Dienstgütevereinbarungen gefunden und dynamisch in den Prozessablauf eingebunden wird. Die Kommunikation zwischen einzelnen Diensten eines Prozesses verläuft folglich ereignisgetrieben in Form von Nachrichten und damit asynchron. Dazu existiert eine zentrale **nachrichtenorientierte Middleware**, die die Kommunikation zwischen den Diensten automatisiert. Dient die nachrichtenorientierte Middleware darüber hinaus auch zur Integration von Diensten in ein Unternehmensnetzwerk und sorgt sie gleichzeitig auch für die notwendige Datenmediation zwischen den Diensten, so wird sie oft als **Enterprise Service Bus** bezeichnet. Integriert ein Enterprise Service Bus im Cloud-Umfeld auch externe Dienste, nennt man ihn auch **Internet Service Bus** (Dufft u. a. 2010). Somit wird eine Plattformunabhängigkeit erreicht, die die Wiederverwendung von Diensten sowohl unternehmensintern als auch unternehmensübergreifend in verschiedenen Geschäftsprozessen ermöglicht.

3.4.3 Geschäfts- und Kollaborationsprozesse

Webdienste stellen Geschäftslogik wiederverwendbar über ein Netzwerk zur Verfügung. Innerhalb einer SOA können diese gefunden und uniform über eine nachrichtenorientierte Middleware oder einen Enterprise Service Bus angesprochen werden. Dies erfolgt im Rahmen eines Geschäftsprozesses, der die benötigten Webdienste orchestriert, also in eine Aufrufreihenfolge bringt. Zahlreiche Definitionen des Begriffs *Geschäftsprozess* wurden bereits von Rump (1999) zusammengetragen. Diese wurden von Staud (2006, S. 9) auf Gemeinsamkeiten untersucht und wiederum zu einer eigenen Definition aggregiert. Das Ziel eines jeden Geschäftsprozesses leitet sich aus den Unternehmenszielen ab. Diese betriebswirtschaftliche Betrachtung steht im Rahmen dieser Arbeit außen vor. Stattdessen sind vielmehr die Aspekte der Zusammenarbeit von Agenten im Rahmen eines Prozesses von Interesse. Darum wird in Definition 6 eine auf die Arbeit zuge-

schnittene Definition des Begriffs *Kollaborationsprozess* gegeben.

Definition 6: Kollaborationsprozess in KomBInoS

Ein Kollaborationsprozesses besteht aus manuellen, teil- oder vollautomatisierten, klar abgegrenzten Aufgaben, welche wiederum hierarchisch gegliedert sein können. Während vollautomatisierte Aufgaben ohne physische Interaktion von einem IT-System durchgeführt werden, werden im Rahmen eines Kollaborationsprozesses Agenten aufgrund ihrer Qualifikation und Rolle mit der Durchführung manueller bzw. teilautomatisierter Aufgaben betraut.

Zur erfolgreichen Durchführung eines Kollaborationsprozesses bzw. seiner Teilaufgaben bedarf es adäquater personeller und materieller Ressourcen sowie Information. In diesem Zusammenhang können Agenten, wie in Kapitel 6.4 (S. 118) gezeigt, etwa Personen oder Roboter sein, welche gemäß den Vorgaben des Prozesses über die geforderten Fähigkeiten verfügen und eine entsprechende Rolle einnehmen können. Ein Kollaborationsprozess kann auf betrieblicher Ebene ein Fertigungs-, Instandhaltungs- oder ein Verwaltungsprozess sein. Damit IT-Systeme Kollaborationsprozesse unterstützen können, müssen diese formal in einem adäquaten Modell repräsentiert werden. Dies kann mit Hilfe entsprechender Modellierungssprachen für Geschäftsprozesse erreicht werden, welche in Kapitel 3.4.3.2 (S. 41) diskutiert werden. Zuvor werden im nächsten Abschnitt die Ebenen der Kollaboration unterschieden.

3.4.3.1 Ebenen der Kollaboration

Im Kontext von Leontievs (1974) Aktivitätstheorie unterscheiden Kaptelinin und Nardi (2006, S. 220) nach dem Grad der Zusammenarbeit insgesamt zwischen drei Ebenen der Kollaboration, die in Abbildung 3.4 dargestellt sind und nachfolgend ausgeführt werden.

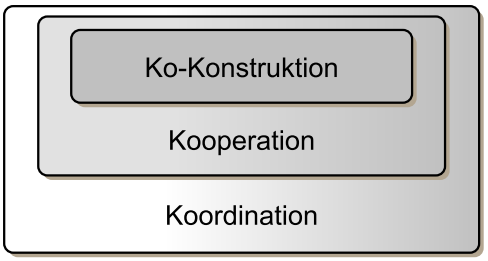


Abbildung 3.4: Die drei Ebenen der Zusammenarbeit

- **Koordination** ist die einfachste Form der Kollaboration. Im Rahmen einer kollektiven Aktivität führen Agenten individuelle Aufgaben weitestgehend unabhän-

gig voneinander durch. Die Ergebnisse der einzelnen Aufgaben tragen zum Gesamtergebnis des übergeordneten Kollaborationsprozesses bei, jedoch werden die Ergebnisse nicht innerhalb der Aufgaben koordiniert. Dies geschieht durch eine externe Instanz.

- **Kooperation** ist eine fortschrittlichere Form der Koordination. Hier müssen Agenten ihre individuellen Ziele im Rahmen eines Kollaborationsprozesses an das übergeordnete kollektive Ziel annähern. Zudem müssen Agenten über die Aufgaben anderer am Prozess beteiligten Agenten informiert sein und ihre eigenen Aktivitäten gegebenenfalls daraufhin anpassen, um die Erreichung des gemeinschaftlichen Ziels nicht zu gefährden.
- **Ko-Konstruktion** übersteigt die Kooperation nochmals in der Hinsicht, dass zusammenarbeitende Agenten sogar die eigentliche Zielsetzung oder das erwartete Resultat während der Durchführung anpassen oder gar komplett verwerfen. Dies geschieht z. B. aufgrund einer objektiven und kontinuierlichen Überprüfung der Sinnhaftigkeit oder der Erfüllbarkeit der Zielsetzung.

3.4.3.2 Modellierungssprachen für Geschäftsprozesse

Geschäftsprozesse beschreiben teils komplexe Arbeitsabläufe, die eine Koordination und Kooperation aller Beteiligten sowie der zur Verfügung stehenden Ressourcen erfordern. So hat sich die *Workflow Patterns Initiative* (van der Aalst und ter Hofstede 2012) zum Ziel gesetzt, auf konzeptioneller Ebene allgemeingültige Koordinationsmuster zu identifizieren, die als Grundlage für Geschäftsprozessmodellierungssprachen dienen. Sie betrachten dazu Koordinationsmuster u. a. für den Kontrollfluss (Van Der Aalst u. a. 2003), Ressourcen (Russell u. a. 2005a), Daten (Russell u. a. 2005b) und Fehlerbehandlung (Russell u. a. 2006). Je mehr Muster von einer Modellierungssprache unterstützt werden, desto größer ist ihre Ausdrucksmächtigkeit.

Je nach Einsatzzweck – von der Dokumentation bis hin zur Ausführbarkeit von Arbeitsabläufen – existieren unterschiedliche Sprachen zur Prozessmodellierung. Zusätzlich zu ihrer Ausdrucksmächtigkeit kann man sie auch anhand ihres Modellierungsansatzes unterscheiden: blockstrukturiert oder graphbasiert (Kopp u. a. 2009). Bei der **blockstrukturierten Modellierung** werden aus prozeduralen Programmiersprachen bekannte Kontrollstrukturen wie Sequenzen, Schleifen oder bedingte Anweisungen als Aktivitäten in Blöcken hierarchisch strukturiert. Im Gegensatz dazu liegen der **graphbasierten Modellierung** Knoten und Kanten zugrunde. Kanten können mit Bedingungen versehen werden, um sie nur in bestimmten Situationen passierbar zu machen. Spezielle Konnektoren zur Synchronisierung verschiedener Ausführungsstränge führen einen Prozess wieder zusammen.

Gerade weil ein Geschäftsprozess mehrere Abteilungen eines Unternehmens oder sogar mehrere Unternehmen international in Beziehung setzt, sind Standards für die Modellierung von Geschäftsprozessen essentiell, um den notwendigen Informations- und Datenaustausch zu gewährleisten.

Web Services Business Process Execution Language (WS-BPEL)

Die WS-BPEL ist eine von der OASIS (2007) standardisierte, XML-basierte Modellierungssprache zur Orchestrierung von Webdiensten. Dadurch ergeben sich ausführbare Geschäftsprozesse, die ausschließlich aus vollautomatisierten Aufgaben zusammengesetzt sind. WS-BPEL-Prozesse sind vornehmlich blockstrukturiert. Durch spezielle Sprachelemente kann auch mit Einschränkungen graphbasiert modelliert werden. Die unterschiedlichen Modellierungsansätze können auch kombiniert werden. Des Weiteren werden Konzepte zur Fehlerbehandlung und Kompensation definiert. Eine grafische Notation ist nicht Bestandteil des Standards. Ebenso werden keine manuellen oder teilautomatisierten Aufgaben betrachtet. Für diese definiert die WS-BPEL Extension for People (WS-BPEL4People) (Ings u. a. 2010) auf Basis von WS-HumanTask (Ings u. a. 2012) eine Erweiterung des Standards.

Ereignisgesteuerte Prozessketten (EPKs)

Im Vergleich zu anderen Ländern werden in Deutschland Prozesse häufig in Form von ereignisgesteuerten Prozessketten (EPK) modelliert (Fettke 2009, S. 24). Einer EPK liegt sowohl eine semiformale Modellierungsmethode als auch eine grafische Notation zugrunde. Diese wurden im Rahmen von Scheers (2002) Architektur integrierter Informationssysteme an der Universität des Saarlandes entwickelt. Einen guten Überblick darüber und den EPK-Modellierungsansatz samt seiner Erweiterung um Elemente zur Organisations- und Datenstrukturierung geben auch Staud (2006) und Seidlmeier (2010). EPKs neigen aufgrund des Modellierungsansatzes dazu, sehr groß zu werden. Aufgrund der geringen Anzahl an Modellierungselementen ist das Erstellen von fachlichen EPKs jedoch recht einfach.

Im direkten Vergleich zwischen dem EPK-Ansatz und der im nächsten Abschnitt vorgestellten Business Process Model and Notation (BPMN) sieht Kurczynski (2008) die BPMN leicht im Vorteil. Dies ließe sich u. a. auf den 10 Jahre späteren Entwicklungsstart der BPMN und den damit einhergehenden Ausgleich einiger Schwachpunkte des EPK-Ansatzes sowie auf stringentere Modellierungsregeln mit Fokus auf ausführbare Prozesse zurückführen. Aus diesem Grunde und weil die EPK-Notation bislang nicht durch ein Standardisierungsgremium genormt ist, wird die Notation an dieser Stelle lediglich der Vollständigkeit halber kurz aufgeführt.

Business Process Model and Notation (BPMN)

Die BPMN ist durch die OMG (2013a) aktuell in Version 2.0.2 standardisiert. Die BPMN definiert eine grafische Notation und ein XML-basiertes Datenformat zur Spezifikation von Geschäftsprozessen. Im Gegensatz zu WS-BPEL ist die BPMN eine rein graphbasierte Modellierungssprache und auf die Modellierung von rollenspezifischen manuellen oder teilautomatisierten Aufgaben ausgerichtet. Vollautomatische Aufgaben können ebenso leicht in einen Geschäftsprozess eingebunden werden. Abbildung 3.5 zeigt die dazu zur Verfügung stehenden Basiselemente der BPMN.

Verantwortlichkeiten in einem Geschäftsprozess werden mit Hilfe von Schwimmbe-

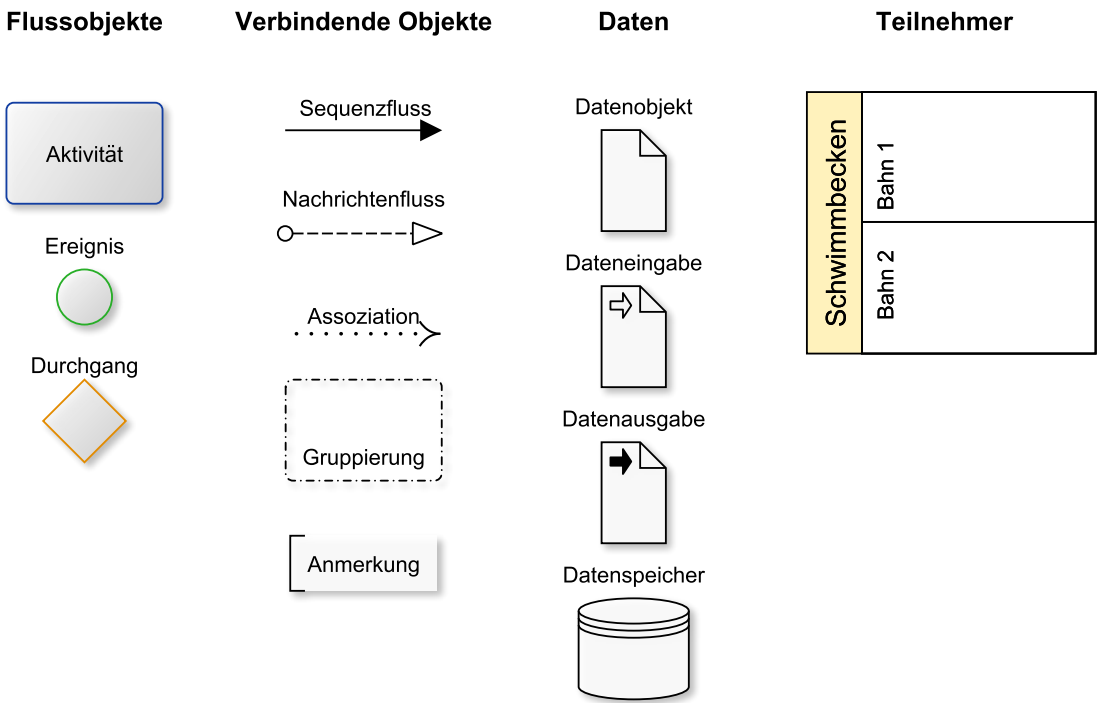


Abbildung 3.5: Die Basiselemente der BPMN nach Freund und Rücker (2016, S. 29)

cken und Bahnen modelliert. Ein Schwimmbecken repräsentiert dabei gemein hin eine Organisation. Bahnen innerhalb eines Beckens stehen für einzelne Organisationseinheiten, Abteilungen oder Rollen. Aktivitäten innerhalb eines Beckens oder einer Bahn drücken aus, was ein Agent gemäß seiner einzunehmenden Rolle zu tun hat. Mehrere Aktivitäten können über einen Sequenzfluss hintereinander geschaltet werden. Mittels Durchgängen werden Verzweigungen und Zusammenführungen von Aktivitäten dargestellt. Mit Hilfe von Ereignissen werden allgemein asynchrone Kommunikationsmuster realisiert. Durch einen Nachrichtenfluss werden Ereignisse zwischen mehreren Becken ausgetauscht. Assoziationen verknüpfen Datenobjekte und ausführungsirrelevante Elemente wie etwa textuelle Anmerkungen mit Flussobjekten. Datenobjekte repräsentieren prozessrelevante Information. Eine Dateneingabe kommt von außerhalb in den aktuellen Prozess und steht darin global zur Verfügung. Eine Datenausgabe wird vom aktuellen Prozess erzeugt und als Ergebnis der Prozessausführung an das aufrufende System, etwa über einen Webdienst, zurückgeliefert. Gruppierungen fassen bestimmte zusammengehörige Aktivitäten rein visuell über Becken- und Bahngrenzen hinweg zusammen. Aufbauend auf diesen Basiselementen sind eine Vielzahl von Spezialisierungen definiert, die z. B. von Freund und Rücker (2016, S. 27ff) anschaulich erläutert werden.

Neben der grafischen Notation spezifiziert die BPMN ab Version 2.0 (OMG 2011a) eine operationelle Semantik (Kapitel 2.4.1, S. 18). Da diese aktuell durch natürliche

Sprache formuliert ist, eröffnet sich ein gewisser Interpretationsspielraum. In der Konsequenz entstanden verschiedene Ansätze zur formalen Definition der operationellen Semantik, welche sich jedoch auf bestimmte Teilmengen konzentrieren (Dijkman und Gorp 2011). Denn zur Herstellung und Gewährleistung von Interoperabilität zwischen Werkzeugen zur Modellierung, Ausführung und Simulation von Geschäftsprozessen in BPMN ist mindestens ein stabiler Kern einer formalen operationellen Semantik erforderlich. Daher darf man erwarten, dass der formalen Spezifikation der operationellen Semantik von BPMN in zukünftigen Hauptversionen des Standards eine höhere Gewichtung beigemessen wird. Im Hinblick auf die technische Umsetzung gibt es auf Basis der BPMN 2.0 bereits mehrere kommerzielle und frei verfügbare Prozessausführungsmaschinen und darüber hinausgehende Plattformen zum Geschäftsprozessmanagement. Teilweise kommen dabei herstellerspezifische Erweiterungen des Standards zum Einsatz, die zwar eine Interoperabilität einschränken, deren Verwendung aber meist optional ist.

Case Management Model and Notation (CMMN)

Während etwa die BPMN als Prozessmodellierungssprache vorschreibt, wie und in welcher Abfolge etwas zu erledigen ist, so ist ein solch strikter Modellierungsansatz nicht für alle Prozesse praktikabel. Dies trifft insbesondere auf schwach strukturierte Prozesse zu, die über einen stark individuellen Charakter verfügen. Generell handelt es sich hierbei um Prozesse mit einem hohen Anteil an manuell zu verrichtender Tätigkeit und einem größeren menschlichen Entscheidungsspielraum. Beide Aspekte hängen wiederum stark vom Wissen und von der Erfahrung des Menschen sowie äußeren Einflüssen ab, weshalb der genaue Ablauf eines Prozesses nicht von vornherein bestimmt werden kann. Beispiele hierfür sind etwa die Erstellung von Versicherungsgutachten im Rahmen einer Schadensregulierung (Kapitel 10.4, S. 235), die individuelle Planung einer Sonderverkaufsfläche (Kapitel 10.6, S. 243) oder die Durchführung von ärztlichen Behandlungen bzw. Wartungs- und Instandhaltungstätigkeiten an Maschinen. Solche schwach strukturierten Prozesse werden im Rahmen des adaptiven Fallmanagements in einer elektronischen Fallakte beschrieben und dokumentiert. Sie spielen, wie an den obigen Beispielen dargestellt, auch im Rahmen der Smart Service Welt eine wichtige Rolle. Für detaillierte Einblicke in die Thematik des adaptiven Fallmanagements wird an dieser Stelle auf die Arbeiten von Swenson (2010, 2011) verwiesen. Zur standardisierten Repräsentation einer solchen Fallakte hat die Object Management Group die Case Management Model and Notation (CMMN) ins Leben gerufen, die inzwischen in Version 1.1 vorliegt (OMG 2016).

Die CMMN ermöglicht im Gegensatz zur BPMN einen rückwärts verketteten Ansatz zur Durchführung eines Prozesses. Abbildung 3.6 zeigt die dazu zur Verfügung stehenden Basiselemente der CMMN sowie die gültige Anwendung von Dekoratoren zur näheren Bestimmung eines Elements. Die Fallbeschreibung ist das Wurzelelement und enthält alle Elemente des Falls sowie alle Elemente, die die Entwicklung eines konkreten Falls zur Laufzeit unterstützen. Dazu gehören Phasen zur groben Strukturierung der Fallbehandlung in entsprechende Planungskontexte. Meilensteine repräsentieren wich-

Anwendbarkeit der Dekoratoren	Planungs-tabelle	Vor-bedingung	Nach-bedingung	Automatischer Abschluss	Manuelle Aktivierung	Obliga-torisch	Wieder-holung
Fallbeschreibung	☑		☑	☑			
Phase	☑	☑	☑	☑	☑	☑	☑
Aufgabe	nur für Agenten-aufgaben	☑	☑		☑	☑	☑
Meilenstein		☑				☑	☑
Ereignis							
Datenelement							

Abbildung 3.6: Die Basiselemente der CMMN (OMG 2014a, S. 62)

tige Zwischenergebnisse im Zuge einer Fallbehandlung. Ein definierter Meilenstein spezifiziert Aufgaben, welche zur Laufzeit den Umständen entsprechend dynamisch ausgewählt und verknüpft werden, um den Meilenstein zu erfüllen. Aufgaben bezeichnen atomare Tätigkeiten. Diese können entweder andere, nachgelagerte Prozesse oder Fallbehandlungen anstoßen oder von Agenten durchgeführt werden. Während einer Fallbehandlung treten oftmals relevante Ereignisse ein. Neben Ereignissen beeinflusst auch fallrelevante Information transportiert durch Datenelemente die Fallbehandlung. Entsprechende Abhängigkeiten zwischen Ereignissen, Datenelementen, Phasen, Aufgaben und Meilensteinen können durch Vor- und Nachbedingungen formuliert werden. Weitere Dekoratoren geben Auskunft darüber, ob z. B. eine Aufgabe manuell gestartet wird, zwingend durchgeführt werden muss oder mehrfach durchlaufen werden kann.

Die CMMN stellt insbesondere in Bezug auf Ad-hoc-Prozesse eine wichtige Ergänzung zur BPMN dar, welche es ermöglicht, manuelle Tätigkeiten stärker zu formalisieren und so überhaupt erst genauer zu erfassen. In diesem Zusammenhang ist hervorzuheben, dass der CMMN-Ansatz aufgrund seiner starken Verwandtschaft zur etablierten Aktivitätstheorie (Leont'ev 1978) aus Sicht der Benutzerinteraktion (Kaptelinin und Nardi 2006) besonders interessant erscheint.

3.4.4 Cloud Computing

Cloud Computing versucht durch Virtualisierungs- und Auslagerungsbestrebungen, vorhandene IT-Infrastruktur in Form von Hard- und Software besser auszulasten und dennoch bedarfsgerecht und dynamisch zur Verfügung zu stellen. Dadurch ergeben sich in Kombination mit den zuvor vorgestellten Schlüsseltechnologien sowohl innovative Geschäftsmodelle für Anbieter von Cloud-basierter IT-Infrastruktur als auch Kosteneinsparungen und eine größere Flexibilität für Kunden. Mell und Grance (2011) vom amerikanischen National Institute of Standards and Technology formulieren eine inzwischen breit akzeptierte Definition des Cloud Computings, anhand derer existierende IT-Infrastruktur sowie zugehörige Modelle und Methoden leicht und vergleichbar in den Cloud-Kontext eingeordnet werden können.

Demzufolge ist **Cloud Computing** ein Modell zur Ermöglichung eines ubiquitären, bequemen und bedarfsgerechten Zugangs zu einem gemeinsamen Pool von konfigurierbaren IT-Ressourcen über ein Netzwerk. Die IT-Ressourcen können mit minimalem Verwaltungsaufwand oder Eingriff des Dienstanbieters bereitgestellt und wieder freigegeben werden. Beispiele für IT-Ressourcen umfassen Rechenleistung, Speicherplatz, Netzwerkbandbreite, Anwendungen und Dienste. Insbesondere spezifiziert das Cloud Computing mit **Software-as-a-Service** ein Dienstleistungsmodell, welches den Zugang zu hochspezialisierten IT-Ressourcen bietet.

3.5 Fazit

Durch die Verknüpfung vieler bislang orthogonaler Themen- und Forschungsfelder stellt die Smart Service Welt insgesamt einen wichtigen Meilenstein zur Realisierung des *Zukünftigen Internets* dar. In diesem Kapitel wurde die Vision der Smart Service Welt zur Optimierung industrieller Prozesse vorgestellt.

Durch die Betrachtung eines Dienstes über den gesamten Lebenszyklus hinweg konnten effektive Methoden entwickelt werden, um Dienste auf Grundlage einer umfänglichen Dienstbeschreibung im Verbund mit einer entsprechenden technischen Unterstützung einer adäquaten Diensteplattform besser auffindbar, vergleichbar, handelbar und kombinierbar zu machen und bedarfsgerecht zu Verfügung zu stellen. Es wurden die zur Realisierung von digitalen Diensten notwendigen Schlüsseltechnologien Webdienste, dienstorientierte Architekturen, Geschäfts- bzw. Kollaborationsprozesse und Cloud Computing in einem für die vorliegende Arbeit angemessenen Detailgrad erläutert.

Diese Arbeit betrachtet multimodale Dialogschnittstellen für Smart Services. Ordnet man die Beiträge der vorliegenden Arbeit den Phasen des Lebenszyklus zu, so sind hauptsächlich die Phasen Angebot und Nutzung relevant. Durch die in Kapitel 7 (S. 143) beschriebene modellgetriebene Erstellungsmethode konform zum ISE-Rahmenwerk werden in der Angebotsphase multimodale Dialogschnittstellen zur kontextsensitiven, primär mobilen Interaktion mit Diensten entwickelt, die im Rahmen eines Kollaborationsprozesses unterschiedliche Agenten in einen Arbeitsablauf einbinden.

Kapitel 9 (S. 211) beschreibt detailliert die KomBInoS-Laufzeitumgebung als eine verteilte multimodale Dialogplattform speziell zur mobilen Nutzung solcher Dienste. Damit können ein oder mehrere in einen Kollaborationsprozess involvierte Agenten koordiniert werden. Durch die eingesetzten Standards und Web-Technologien stellt die KomBInoS-Laufzeitumgebung eine cloudfähige Dialogplattform-as-a-Service dar, die für die Smart Service Welt u. a. einen generischen, wiederverwendbaren Technologiebaustein *Alerting Management und Kollaboration* bereitstellt.

KAPITEL 4

Multimodale Dialogschnittstellen

Dieses Kapitel behandelt die Grundlagen multimodaler Dialogschnittstellen. Diese ermöglichen in vielen, gerade auch mobilen Nutzungskontexten eine natürliche Benutzerinteraktion und sind darum auch im Rahmen der Smart Service Welt von besonderer Relevanz. Nach einem Exkurs in die Grundzüge der menschlichen Konversation werden für die Arbeit wichtige Konzepte und Verfahren zur Realisierung von multimodalen Dialogsystemen erläutert. Vor dem Hintergrund dieses Wissens wird schließlich die technologische Zielplattform dieser Arbeit, die SiAM-Dialogplattform, vorgestellt.

4.1 Einleitung

Die Verbreitung und Nutzung von mobilen Endgeräten aller Art nimmt beständig und in allen Lebenslagen zu. Während Smartphones heute schon allgegenwärtig sind, werden bei Wearables, also am Körper getragene intelligente Geräte, starke Steigerungsraten in der Durchdringung verzeichnet. So rechnet die International Data Corporation (IDC 2017) mit einer Verdopplung der jährlichen Verkaufszahlen solcher Geräte von heute 125 Millionen auf über 240 Millionen Stück im Jahr 2021. Hierzu zählen hauptsächlich intelligente Uhren (67%), Armbänder (22%) und Kleidung (9%). Die Hewlett-Packard Tochter aruba Networks (2014) sieht in einer Studie entsprechend starke Auswirkungen auf das zukünftige Arbeitsleben. Mobile Geräte, gleich welcher Klasse, unterliegen bestimmten physischen Einschränkungen, die eine angepasste Interaktion und Darstellung bedingen (Heckmann u. a. 2007, S. 21). Außerdem verlangt der ständig wechselnde mobile Nutzungskontext von Benutzern im Vergleich zu einer kontrollierbaren stationären Arbeitsumgebung eine höhere Aufmerksamkeit ab. Dies kann zu kostspieligen Bedienungsfehlern oder gar Unfällen führen.

Hier sind intelligente Benutzerschnittstellen unerlässlich. Sie versprechen laut Maybury und Wahlster (1998, S. 12) eine effizientere, effektivere und natürlichere Benut-

zerinteraktion. Diese Eigenschaften sind auch gerade bei der mobilen Interaktion mit Smart Services von großer Relevanz, steigern sie doch die Prozess- und Transaktionssicherheit und damit letztendlich das Vertrauen in die Technologie. Weiterhin sind diese Eigenschaften integraler Bestandteil von Maybury und Wahlsters (1998, S. 2) Definition von intelligenten Benutzerschnittstellen, die an dieser Stelle sinngemäß wiedergegeben wird.

Definition 7: Intelligente Benutzerschnittstellen (Maybury und Wahlster 1998, S. 2)

Intelligente Benutzerschnittstellen sind Mensch-Maschine-Schnittstellen, die darauf abzielen, die Effizienz, Effektivität und Natürlichkeit der Mensch-Maschine-Interaktion zu steigern. Dazu bilden sie den oder die Benutzer, die Domäne, die spezifischen Aufgaben, den Diskurs, den Kontext und relevante Medien bzw. Modalitäten ab, nutzen diese Modelle aus und schlussfolgern darauf.

Insbesondere eignen sich laut Oviatt (2012, S. 410) multimodale Benutzerschnittstellen, um wechselnden Kontexten in mobilen Anwendungssituationen Rechnung zu tragen. In diesem Kapitel werden daher die für diese Arbeit relevanten Grundlagen multimodaler Dialogschnittstellen erarbeitet. Ein wichtiges Designprinzip, welches bereits im WIP-System (Wahlster u. a. 1993) Anwendung fand, wurde von Wahlster (2003, S. 12) formuliert als „*no presentation without representation*“. Damit ist gemeint, dass eine Repräsentation einer multimodalen Systemausgabe notwendig ist, um darauf referenzierende Benutzereingaben im weiteren Diskurs auflösen zu können. Das System weiß also, was es gesagt hat. Wasinger (2006), Pfleger (2007), und Löckelt (2008) haben dieses Forschungsgebiet bereits ausführlich bearbeitet, sodass sich dieses Kapitel in Teilen auf die darin erarbeiteten theoretischen Grundlagen, wie etwa die Unterscheidung in verbale und nichtverbale Kommunikation, stützt, ohne selbst detailliert darauf einzugehen.

Im Folgenden wird in die wichtigsten Grundlagen und Konzepte menschlicher Konversation (Kapitel 4.2, S. 50) und multimodaler Dialogsysteme (Kapitel 4.3, S. 56) eingeführt. Anschließend wird in Kapitel 4.4 (S. 66) die SiAM-Dialogplattform vorgestellt, die die technologische Zielplattform für diese Arbeit liefert.

4.2 Grundlagen menschlicher Konversation

In diesem Unterkapitel werden in Anlehnung an Jurafsky und Martin (2009, Kap. 24) wichtige Aspekte menschlicher Konversation sowie darauf aufbauende Theorien und Konzepte eingeführt und erläutert, welche direkten Einfluss auf die Implementierung und den Einsatz von natürlichsprachlichen Dialogsystemen haben.

4.2.1 Sprecherinitiative und-wechsel

Gesprochene Sprache ist ein halb-duplex Kommunikationskanal. Das bedeutet, damit eine Konversation zwischen mehreren Gesprächspartnern zustande kommen kann, müssen sich die aktiven Sprecher im Verlauf der Konversation abwechseln. Ein geordneter Sprecherwechsel ist also zur Organisation einer Konversation grundlegend, vor allem weil ein Sprecherwechsel üblicherweise in einem sehr engen Zeitintervall von nur wenigen hundert Millisekunden erfolgt. Sacks u. a. (1974, S. 704) formulieren dazu Regeln, wie an übergangsrelevanten Positionen im Gespräch, z. B. am Ende einer Äußerung, ein Sprecherwechsel stattfinden kann.

(1) Falls der aktuelle Sprecher in seinem Redebeitrag explizit einen anderen Teilnehmer als nächsten Sprecher auswählt, so hat dieser Teilnehmer (im Gegensatz zu allen anderen) das Recht und die Pflicht die Initiative zu ergreifen. In diesem Fall wird der nächste Sprecher durch **Fremdauswahl** bestimmt.

(2) Falls der aktuelle Sprecher in seinem Redebeitrag keinen Nachfolger explizit bestimmt, so steht es jedem Teilnehmer frei, selbstständig die Initiative zu ergreifen. In diesem Fall wird der nächste Sprecher durch **Selbstauswahl** bestimmt.

(3) Falls kein anderer Teilnehmer die Initiative für den nächsten Beitrag ergreift, verbleibt sie beim aktuellen Sprecher.

4.2.2 Sprechakte

Eine Konversation besteht also aus einer Abfolge von Äußerungen unterschiedlicher Sprecher. Wittgenstein (1953) und Austin (1962) postulieren, dass eine Äußerung innerhalb eines Dialogs eine Art Aktion ist, die vom aktuellen Sprecher ausgeführt wird. Nach Austin (1962) stellen jedoch nicht nur performative Äußerungen, d.h. Äußerungen, die eine Handlung ausdrücken (z. B. „*Kraft meines Amtes erkläre ich euch hiermit zu Mann und Frau.*“) Sprechakte dar, sondern alle Äußerungen innerhalb eines Dialogs. Er unterscheidet dabei im Rahmen der **Sprechakttheorie** (Hindelang 2010) zwischen drei Arten von Sprechakten, die jede Äußerung bewirkt.

- Der **lokutionäre Akt** bezeichnet die Handlung, etwas bedeutungsvolles zu sagen. Dieser Akt unterteilt sich in einen **phonetischen Akt** (das Erzeugen von sprachlichen Lauten), einen **phatischen Akt** (das Erzeugen von Äußerungen getreu der Grammatik und unter Verwendung von Wörten und syntaktischer Strukturen einer bestimmten Sprache) sowie einen **rhetischen Akt** (das Erzeugen von Äußerungen mit einer bestimmten Bedeutung).
- Der **illokutionäre Akt** transportiert die eigentliche Funktion, sprich die der Äußerung zugrundeliegende Intention des Sprechers. Eine genauere Differenzierung von illokutionären Akten folgt in der nächsten Aufzählung.
- Der **perlokutionäre Akt** beschreibt das Erzielen einer beabsichtigten Wirkung beim Adressaten einer Äußerung. Ist eine andere als die beabsichtigte Wirkung

beim Adressaten eingetroffen, ist diese Wirkung zwar immer noch ein perlokutionärer Effekt. Der perlokutionäre Akt ist hingegen aber fehlgeschlagen.

Austins Unterscheidung von Sprechakten wird von Searle (1969) verfeinert. Er fasst den phonetischen und den phatischen Anteil von Austins lokutionären Akt zu einem **Äußerungsakt** zusammen. Des Weiteren misst er Austins rhetischem Akt eine größere Bedeutung zu und bezeichnet ihn als **propositionalen Akt**. Dieser besteht aus einem **Referenzakt** und einem **Prädikationsakt**. In ersterem wird auf Objekte der Welt verwiesen. In letzterem wird etwas über das Objekt ausgesagt.

Der Begriff des Sprechaktes wird im Allgemeinen für den illokutionären Akt verwendet, weil dieser im Gegensatz zu den anderen am leichtesten zu fassen ist. In diesem Zusammenhang schlägt Searle (1975b) eine Taxonomie für (illokutionäre) Sprechakte, in die sich alle Äußerungen einordnen lassen, wie folgt vor:

- **Assertiva** bezeichnen Aussagen, mit denen der Sprecher seine Sicht über die Welt darlegt. Diese Aussagen können sich unter objektiven Gesichtspunkten auch als falsch herausstellen.
- **Direktiva** bezeichnen Aussagen, mit denen der Sprecher versucht, den Adressaten zur Durchführung einer bestimmten Handlung zu bewegen.
- **Kommissiva** bezeichen Aussagen, mit denen der Sprecher sich bereit erklärt, etwas zu tun oder zu lassen.
- **Expressiva** bezeichen Aussagen, mit denen der Sprecher seinen mentalen Zustand oder seine Einstellung bzgl. etwas mitteilt.
- **Deklarativa** bezeichnen Aussagen, mit denen der Sprecher den Zustand der Welt ändert bzw. ändern möchte (siehe Austins performative Äußerungen)

4.2.3 Dialogakte

Die im vorherigen Abschnitt vorgestellten Sprechakte beziehen sich auf einzelne Äußerungen eines Sprechers. Ein Dialogakt kann laut Jurafsky und Martin (2009, S. 876) als eine Erweiterung eines Sprechaktes angesehen werden, welche es erlaubt, auf vorangegangene bzw. nachfolgende Sprech- bzw. Dialogakte eines Diskurses als Folge von Äußerungen zu verweisen. Somit wird der Fokus vom Verstehen einzelner Äußerungen auf das Verstehen des größeren Diskurses erweitert. Ein Dialogakt wird also innerhalb des gesamten Diskurskontextes interpretiert. Es entstehen Paare von aufeinanderfolgenden Dialogakten, die eine zusammengehörige Verkettung von Dialogakten eines Diskurses bilden. Für Bunt (1994, S. 21) besteht ein Dialogakt neben der Äußerungsform aus einem **semantischen Inhalt**, der in den Diskurs eingebracht wird, und einer **kommunikativen Funktion**, die die Intention den Sprechers in Bezug auf den semantischen Inhalt näher bestimmt. Dieser Zusammenhang wird auch aus Abbildung 4.1 deutlich.

Insgesamt kann so spezifiziert werden, welche Kontextänderung ein Dialogakt bewirken kann. Es wurden diverse, teils domänenspezifische, Taxonomien für die Annotation von (transkribierten) Konversationen mit Dialogakten entwickelt, z. B. im Verbmobil

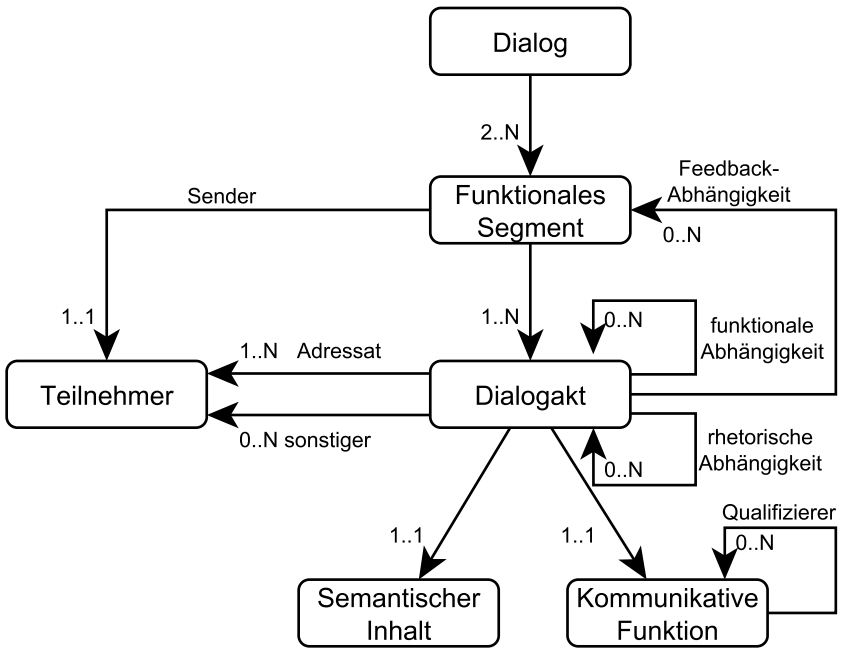


Abbildung 4.1: Das Dialogakt-Metamodell nach ISO (2012)

Projekt von Jekat u. a. (1995) sowie Alexandersson u. a. (1998). Daraus hat sich schließlich eine domänenunabhängige Taxonomie herausgebildet, die von der Internationalen Organisation für Normung (ISO 2012) standardisiert wurde. Abbildung 4.2 zeigt einen Ausschnitt von universell verwendbaren kommunikativen Funktionen. Darüber hinaus existieren dimensionsspezifische kommunikative Funktionen, z. B. für einen geordneten Sprecherwechsel (Kapitel 4.2.1, S. 51) oder zum Austausch von sozialen Obligationen.

Mit Hilfe des Dialogakt-Metamodells sowie der Taxonomie von kommunikativen Funktionen lassen sich einerseits Schegloff und Sacks (1973) Paare aufeinander folgender Dialogakte unterschiedlicher Sprecher (Adjacency Pairs) identifizieren und abbilden, z. B. *Question* → *Answer*, *Offer* → *Accept/Decline Offer* oder *Request* → *Accept/Decline Request*. Andererseits können auch rhetorische Abhängigkeiten von Äußerungen im Sinne einer Unterscheidung zwischen Nukleus und Satelliten gemäß Mann und Thompsons (1988) Theorie rhetorischer Strukturen beschrieben und analysiert werden. Solch ein kohärenter Diskurs wird von Grosz und Sidner (1986) in drei Komponenten strukturiert: die lingusitische Struktur, die intentionale Struktur und den Aufmerksamkeitszustand der Gesprächsteilnehmer.

4.2.4 Natürlichsprachliches Kommunikationsverhalten

Bislang wurde ausschließlich auf die Rolle des Sprechers eingegangen, der in einer Konversation Sprech- bzw. Dialogakte ausführt. In diesem Abschnitt wird der Fokus auf

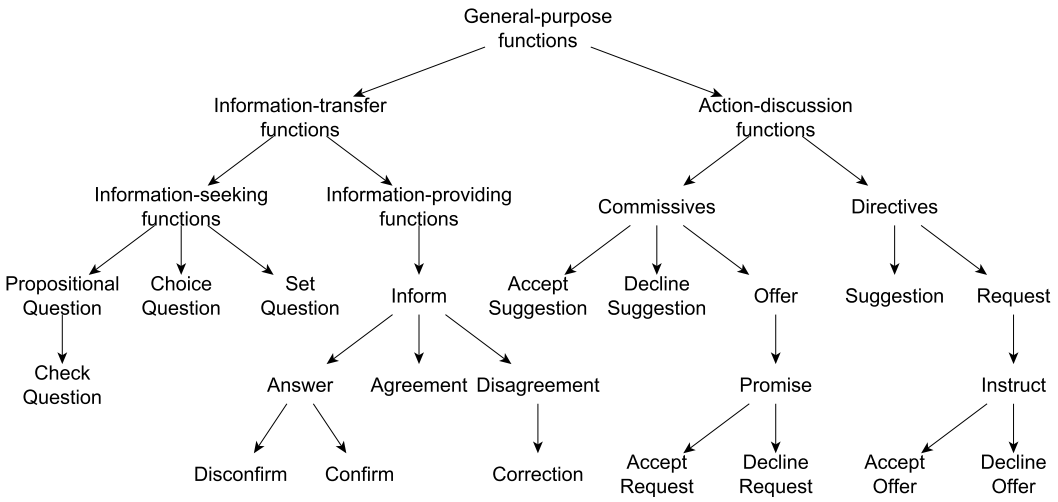


Abbildung 4.2: Taxonomie von allgemeinen kommunikativen Funktionen nach ISO (2012)

die Rolle des Zuhörers bzw. des Adressaten ausgedehnt. Watzlawick u. a. (2011, Erstausgabe 1967) formulierten sechs Kommmunikationsaxiome, auf deren Grundlage von Thun (2016, Erstausgabe 1981) das Vier-Seiten-Modell der Kommunikation entwickelte. Das erste Axiom besagt, dass man nicht nicht kommunizieren kann. In diesem Sinne signalisiert ein Zuhörer dem Sprecher intuitiv verbal oder auch nonverbal, ob er diesen verstanden hat und ob er dessen Meinung teilt. Da eine solche Signalisierung über einen Rückkanal während der eigentlich Äußerung des Sprechers erfolgt, wird dies auch als **Backchanneling** bezeichnet. Sie ist wichtig für den Sprecher, um fortfahren oder gegebenenfalls korrigieren zu können. Clark (1996, S. 222) formuliert dieses Informationsbedürfnis des Sprechers ähnlich zu Normans (2002, S. 27) *Principle of Feedback* als **Prinzip der Abgeschlossenheit**[1] und meint damit, dass Handelnde, die eine Aktion durchführen, für den aktuellen Zweck hinreichende Anhaltspunkte benötigen, dass die Durchführung der Aktion erfolgreich war. Diese Anhaltspunkte fließen ohne Sprecherwechsel durch Backchanneling in die Konversation ein. In diesem Zusammenhang beschreibt Hahn (2017) ein adaptives Verfahren zum Dialogmanagement auf Basis von Backchanneling.

Clark und Schaefer (1989, S. 265) vertreten die Auffassung, dass ein Redebeitrag eine gemeinschaftliche Aktion eines Sprechers und seiner Zuhörer ist. Demzufolge teilen sie einen Redebeitrag in zwei Phasen ein. In der **Präsentationsphase** wird der Adressat vom Sprecher mit einer Äußerung konfrontiert. In der **Akzeptanzphase** interpretiert

[1] Principle of Closure: „Agents performing an action require evidence, sufficient for current purposes, that they have succeeded in performing it.“

der Zuhörer die Äußerung und signalisiert dem Sprecher, ob er (seiner Meinung nach) den Beitrag verstanden hat. Des Weiteren nennen Clark und Schaefer (1989, S. 267) verschieden starke Anhaltspunkte, die dem Sprecher signalisieren, dass der Zuhörer seinen Beitrag verstanden hat. Die folgende Liste sortiert diese Anhaltspunkte aufsteigend.

(1) **Andauernde Aufmerksamkeit.** Der Zuhörer offenbart dem Sprecher keine Änderung in seiner Aufmerksamkeit. Dies signalisiert dem Gegenüber, dass der Zuhörer mit dem Gesagten weiterhin einverstanden ist.
(2) **Nächster Beitrag.** Der Zuhörer fährt mit einem eigenen Redebeitrag fort, der sich schlüssig in die Unterhaltung einfügt.
(3) **Bestätigung.** Der Zuhörer signalisiert dem Sprecher durch nonverbale Kommunikation (Augenkontakt, Nicken) oder kurzen Bestätigungslauten („Ah“, „Mhm“) oft schon während der Äußerung, dass er versteht.
(4) **Bekundung.** Der Zuhörer fasst sein Verständnis eines Teils oder der vollständigen Äußerung in eigene Worte.
(5) **Darlegung.** Der Zuhörer wiederholt Teile der Äußerung oder die vollständige Äußerung.

Während die schwächeren Anhaltspunkte sehr subtil sein können, sind die starken hingegen sehr explizit. Das wirft – gerade im Hinblick auf Dialogsysteme – die Frage auf, wann welche Reaktion des Zuhörers vom Sprecher als hinreichend für ein erfolgreiches Verstehen interpretiert werden kann. Clark und Schaefer (1989, S. 267) überlassen dies empirischen Untersuchungen, weisen aber darauf hin, dass die Stärke der Reaktion des Zuhörers im Allgemeinen von der Art und Weise der Präsentation (je komplexer, desto stärker) sowie der Wichtigkeit des aktuellen Zwecks der Präsentation abhängt. Weiterhin bezeichnen Clark und Schaefer (1989, S. 263) den Diskurs unter Anwendung der genannten Kriterien zur Erreichung einer (vermeintlich) gemeinsamen Auffassung als **Grounding**. Das folgende Beispiel verdeutlicht einen solchen Diskurs.

(1) A: Würden Sie mir bitte noch Ihre Telefonnummer geben?
(2) B: Klar, 0681/85775…[*Bestätigung & nächster Beitrag*]
(3) A: Mhm.[*Bestätigung via Backchanneling*]
(4) B: …5272.
(5) A: Ich habe vermerkt: Saarbrücken 85775-5272. [*Bekundung & Darlegung*]
(6) B: Genau. [*Bestätigung*]
(7) A: Dankeschön. Wann kann ich Sie erreichen? [*Bestätigung & nächster Beitrag*]

Auf Grundlage von Dialogakten beschreibt Traum (1999) ein weiteres Grounding-Modell und vergleicht dieses mit dem von Clark und Schaefer (1989).

4.2.5 Konversationelle Implikaturen

In menschlicher Konversation wird meistens mehr kommuniziert als wörtlich gesagt. So ist etwa ein einfaches „*Ja.*“ auf die Frage, ob man an die Fernbedienung auf dem Wohnzimmertisch heran kommt, sehr wahrscheinlich nicht die erwünschte Reaktion. Vielmehr drückt die Frage indirekt den Wunsch des Sprechers aus, die Fernbedienung gereicht zu bekommen. In der Mensch-Maschine-Kommunikation führen solche verborgenen Bedeutungsaspekte unweigerlich zu Problemen, weshalb gewisse Konversationsregeln eingehalten werden müssen.

Während Searle (1975a) darunter einen indirekten Sprechakt versteht, bezeichnet Grice (1975, S. 44) die eigentliche Funktion einer Äußerung jenseits des wörtlich Gesagten als konversationelle Implikatur. Mit ihrer Hilfe können scheinbar, d.h. im wörtlichen Sinn, zusammenhangslose Äußerungen in einem Gespräch doch interpretiert und verstanden werden. Dazu müssen aber alle Gesprächspartner dem **Kooperationsprinzip** unterliegen, welches Grice (1975, S. 45) wie folgt formuliert: „*Make your conversational contribution such as is required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which you are engaged.*“ Meggle (1979, S. 248) liefert eine entsprechende Übersetzung ins Deutsche: „*Mache deinen Gesprächsbeitrag jeweils so, wie es von dem akzeptierten Zweck oder der akzeptierten Richtung des Gesprächs, an dem du teilnimmst, gerade verlangt wird.*“

Unter der Annahme, dass das Kooperationsprinzip von allen Gesprächsteilnehmern befolgt wird, lassen sich gültige Inferenzregeln ableiten, die sich zur Interpretation bzw. zur Bedeutungsextraktion von Äußerungen heranziehen lassen. Grice (1975, S. 45ff.) (bzw. in der deutschen Übersetzung Meggle (1979, S. 249ff.)) unterscheidet dazu in Anlehnung an die Kantische Kategorientafel vier Kategorien von Konversationsmaximen.

(1) **Maxime der Quantität.** Sei informativ. Gib in deinem Beitrag so viel Information wie (für die gegebenen Gesprächszwecke) nötig preis, aber nicht mehr.

(2) **Maxime der Qualität.** Sei ehrlich. Sag insbesondere nichts, was du für unwahr erachtest oder wofür dir angemessene Anhaltspunkte fehlen.

(3) **Maxime der Relation.** Sei relevant. Trage zum Fortschritt der aktuellen Konversation bei und schweife nicht vom Thema ab.

(4) **Maxime der Modalität.** Sei präzise. Fasse Dich kurz und prägnant, vermeide Unklarheiten und Mehrdeutigkeiten.

4.3 Grundlagen multimodaler Dialogsysteme

Menschen interagieren untereinander grundsätzlich mit allen zur Verfügung stehenden Sinnen unter Berücksichtigung relevanter Kontextfaktoren und Einschränkungen, die der aktuellen Interaktionssituation geschuldet sind. Was zwischenmenschlich intuitiv von allen an der Kommunikation beteiligten Personen verstanden wird, muss für eine natürliche Mensch-Computer-Interaktion möglichst gut nachempfunden werden. In die-

sem Unterkapitel werden daher relevante Grundlagen und Konzepte vorgestellt, die zur Realisierung einer natürlichen multimodalen Mensch-Computer-Interaktion erforderlich sind.

Zuvor muss aber der Begriff der (Sinnes-) Modalität, häufig auch als Modus bezeichnet, näher bestimmt werden. Maybury und Wahlster (1998, S. 4f.) unterscheiden etwa zwischen den Begriffen Modalität und Medium. Der Begriff **Modalität** bezieht sich auf die menschlichen Sinne zur Wahrnehmung und Verarbeitung von eingehender Information (Sehen, Hören, Riechen, Schmecken und Tasten). Der Begriff **Medium** meint das physische Objekt, mit dem die Sinne angesprochen werden. Die durch das Medium transportierte und von einem Sinn wahrgenommene Information unterliegt einer entsprechend verständlichen Codierung, einem System von Symbolen. Ein solcher **Code** kann etwa natürliche Sprache sein. Diese kann, wenn gesprochen (und durch einen Lautsprecher transportiert), gehört werden. Oder sie kann, wenn geschrieben (und von einem Display angezeigt), gelesen werden. Gleichzeitig können durch ein Medium aber auch andere Codes transportiert werden, z. B. können auf einem Display nicht nur Texte, sondern auch Bilder und Videos angezeigt werden. Das Verhältnis zwischen Medium, Modalität und Code ist also nicht trivial. Dennoch werden die Begriffe Medium und Modalität in der Praxis oft ambig benutzt.

Oviatt (2012) argumentiert diesbezüglich aus der Perspektive eines soziotechnischen Systems (Kapitel 2.2.2, S. 12), in welchem der Mensch durch Interaktion mit dem System Information bereitstellt. Für Oviatt (2012, S. 405) kann dies etwa durch Sprache, Stifteingabe, Touch, Gesten oder Kopf- und Körperbewegungen erfolgen. Für Nigay und Coutaz (1993, S. 172f.) sind dazu auf Seiten eines technischen Systems entsprechende Interpretationsmechanismen analog zur menschlichen Wahrnehmung erforderlich, um aus einer über ein bestimmtes Eingabemedium erfolgten Benutzereingabe eine Bedeutung zu decodieren bzw. zu extrahieren.

Mit dem Ziel einer späteren technologieorientierten Definition von multimodalen Benutzerschnittstellen nimmt auch diese Arbeit verstärkt eine technische Sichtweise auf den Modalitätsbegriff ein. Unter Berücksichtigung der Sichtweisen von Maybury und Wahlster (1998), Oviatt (2012) und Nigay und Coutaz (1993) definiert sich der Begriff für diese Arbeit darum folgendermaßen:

Definition 8: Modalität in KomBInoS

Im Rahmen der Mensch-Computer-Interaktion beschreibt eine Modalität m ein Tripel

$$m = (Medium, Code, Interpreter),$$

wobei ein physisches Objekt (Medium) einen (semiotischen) Code an einen Empfänger sendet, der die Bedeutung des Codes mit Hilfe eines geeigneten Interpreters verstehen kann.

Bei der Benutzerinteraktion mit einem technischen System unterscheidet man zwi-

schen einer Sende- und einer Empfangsrichtung und folglich zwischen Eingabe- und Ausgabemodalitäten. **Eingabemodalitäten** transportieren Information vom Benutzer zum System. Das (technische) Medium fungiert hier als Sensor, der den vom Benutzer ausgesendeten Code empfängt und dessen Bedeutung durch einen in Software gegossenen Interpretationsmechanismus extrahieren kann. Umgekehrt transportieren **Ausgabemodalitäten** Information vom System an den Benutzer. Das (technische) Medium kann die Information so darstellen, dass ein Benutzer sie mit Hilfe eines menschlichen Sinnes interpretieren kann. Definition 8 ermöglicht eine feingranulare Betrachtung verschiedener Modalitäten, da sich zwei Modalitäten voneinander unterscheiden, sobald sich lediglich eine Komponente der entsprechenden Tripel unterscheidet.

Nach Wahlster (2003) ist die Symmetrie von Ein- und Ausgabemodalitäten einer multimodalen Benutzerschnittstelle wichtig. Symmetrische Multimodalität meint, dass alle Eingabemodalitäten auch als Ausgabemodalitäten zur Verfügung stehen und umgekehrt. Dies erfordert eine Beachtung des eingangs formulierten Entwurfsprinzips „*keine (multimodale Ausgabe-) Präsentation ohne Repräsentation*“. Symmetrische Multimodalität ist eine Voraussetzung zum Umgang mit vielen wichtigen Diskursphänomenen wie Anaphern und crossmodalen Referenzen im Rahmen der multimodalen Fusion und Fission. Bevor darauf näher eingegangen wird, werden zunächst Modelle zur Klassifikation von multimodalen Benutzerschnittstellen vorgestellt.

4.3.1 Klassifikation multimodaler Benutzerschnittstellen

Laut Oviatt (2012, S. 409) verarbeiten multimodale Benutzerschnittstellen zwei oder mehr kombinierte Eingabemodalitäten eines Benutzers in einer koordinierten Art und Weise in Verbindung mit einer multimedialen Systemausgabe. Weitere feingranulare Unterscheidungen von Oviatt (2012, S. 409), z. B. zeitlich kaskadierende multimodale Benutzerschnittstellen, lassen sich auch durch den Designraum Nigay und Coutaz (1993) zur Klassifikation von multimodalen Benutzerschnittstellen herleiten. Dieser wird auch als **CASE-Modell** bezeichnet und ist in Abbildung 4.3 dargestellt.

Anhand der für Nigay und Coutaz (1993) herausragenden Eigenschaften Modalitätenfusion (und -fission) sowie temporale Nutzung von Modalitäten lassen sich vermeintlich multimodale Benutzerschnittstellen hinreichend aus Systemsicht miteinander vergleichen und grob in vier Klassen einteilen.

- **Exklusive** multimodale UIs stellen die einfachste Form multimodaler UIs dar. Unterschiedliche Modalitäten können lediglich nacheinander genutzt werden. Eine Aufteilung von zusammengehörender Information auf mehrere Modalitäten ist nicht möglich.
- **Alternierende** multimodale UIs erlauben die Nutzung mehrerer Modalitäten nacheinander. Durch eine vorhandene multimodale Fusion können Informationsbruchstücke aus den unterschiedlichen Kanälen zu einer gemeinsamen Anweisung zusammengeführt werden.
- **Nebenläufige** multimodale UIs können mehrere Modalitäten zur gleichen Zeit

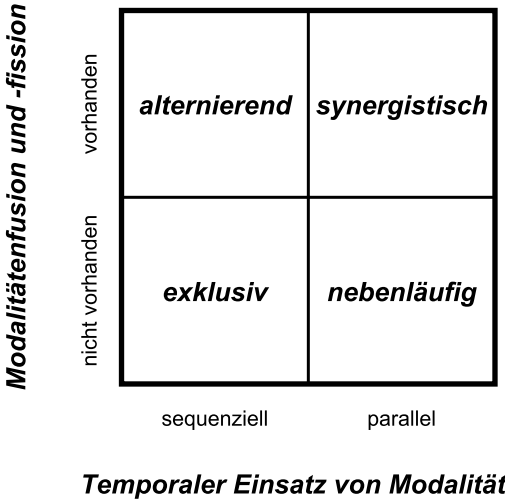


Abbildung 4.3: Designraum zur Klassifikation von multimodalen Benutzerschnittstellen nach Nigay und Coutaz (1993, S. 173)

verarbeiten. Dies geschieht jedoch unabhängig voneinander.

- **Synergistische** multimodale UIs sind hingegen in der Lage, zeitgleich mehrere Modalitäten zu berücksichtigen und die darüber transportierten Informationsbruchstücke zu einer gemeinsamen Anweisung zu formen.

Diese Beobachtungen führen zu Definition 9.

Definition 9: Multimodale Benutzerschnittstellen in KomBInoS

Multimodale Benutzerschnittstellen sind intelligente Benutzerschnittstellen, die es einem Benutzer ermöglichen, mit mehreren Ein- und Ausgabemodalitäten mit einem System zu interagieren.

Dabei unterscheidet man zwischen exklusiven, alternierenden, nebenläufigen und synergistischen multimodalen Benutzerschnittstellen.

Coutaz u. a. (1995) formulieren zudem mit den **CARE-Eigenschaften** ein formales konzeptuelles Rahmenwerk zur Beurteilung der Nutzerfreundlichkeit von multimodalen Benutzerschnittstellen. Dabei löst eine Modalität eine Aktion aus, die von einem Zustand s in einen intendierten Zielzustand s' überführt. In Bezug auf die in Kapitel 4.2.2 (S. 51) erläuterte Sprechakttheorie handelt es sich also um einen perlokutionären Akt. Coutaz u. a. (1995) geben dem Entwickler zwar keine konkreten Umsetzungsempfehlungen. Da unterschiedliche Benutzer in unterschiedlichen Nutzungskontexten unterschiedliche Bedürfnisse in Bezug auf die Interaktion mit einer multimodalen Benutzerschnittstelle haben, sollten von einem multimodalen Interaktionssystem pro durchführbarer

(Inter-) Aktion aber möglichst viele CARE-Eigenschaften adressiert werden, um nutzerfreundlich zu sein.

- **Äquivalenz** (**E**quivalence). Modalitäten, die (ohne temporale Einschränkungen) von Zustand *s* nach *s'* überführen sind äquivalent.
- **Redundanz** (**R**edundancy). Redundante Modalitäten sind äquivalent und können gleichzeitig verwendet werden, ohne die Expressivität zu steigern.
- **Zuordnung** (**A**ssignment). Genau eine Modalität führt von Zustand *s* nach *s'*. Entweder existiert technisch keine Modalitätenwahl oder es wird immer die gleiche Modalität gewählt.
- **Komplementarität** (**C**omplementarity). Komplementäre Modalitäten müssen gemeinsam verwendet werden, um einen Zustandswechsel von *s* nach *s'* durchzuführen.

Die Klasse der synergistischen multimodalen Benutzerschnittstellen stellt auf der einen Seite die natürlichste Ausprägung solcher UIs dar. Auf der anderen Seite stellen sie auch die höchsten Anforderungen an ein darunterliegendes multimodales Interaktionssystem zur Verarbeitung der Ein- und Ausgaben. Hier kommen Dialogsysteme zum Einsatz, die über spezielle Komponenten für die multimodale dialogische Interaktion verfügen. Die weiteren Abschnitte gehen gezielt auf relevante Konzepte zur Realisierung synergistischer multimodaler Dialogschnittstellen ein. Der Aufbau orientiert sich dabei an der Verarbeitungskette eines multimodalen Dialogsystems auf Basis der Systemarchitektur.

4.3.2 Architektur multimodaler Dialogsysteme

Die **konzeptuelle Architektur** von multimodalen Dialogsystemen ist in Abbildung 4.4 dargestellt. Sie entspricht im Wesentlichen der des W3C Multimodal Interaction Frameworks[2] und implementiert eine mehrstufige Verarbeitungskette nach Brennan (1998, S. 17). Jede Modalität wird durch einen unimodalen Erkenner sensorisch aufgezeichnet und von einem Interpreter vorverarbeitet. Diese teils unvollständigen Interpretationen werden durch die multimodale Fusion zu einer vollständigen semantischen Interpretation integriert, welche im Dialogmanager verarbeitet wird. Dieser fragt gegebenenfalls zur Durchführung einer Transaktion oder zum Beziehen von benötigter Information ein domänenspezifisches Back-End an, welches durch entsprechende Wrapper und diverse Programmierschnittstellen (API) zugänglich ist. Dies reicht von nativen Schnittstellen zu cyber-physischen Systemen über leichtgewichtige REST-basierte Schnittstellen (Sonntag u. a. 2010a) bis hin zu komplexen semantischen Diensteplattformen wie im Projekt SmartWeb (Sonntag u. a. 2007) oder im THESEUS Forschungsprogramm (Bergweiler 2014). Nachdem der Dialogmanager eine Antwort konzipiert hat, muss diese bestmöglich auf die zu Verfügung stehenden Ausgabemodalitäten verteilt werden, um sie dem Benutzer situationsgerecht präsentieren zu können. Dies geschieht

[2] siehe http://www.w3.org/TR/mmi-framework/ (letzter Zugriff: 04.11.2017)

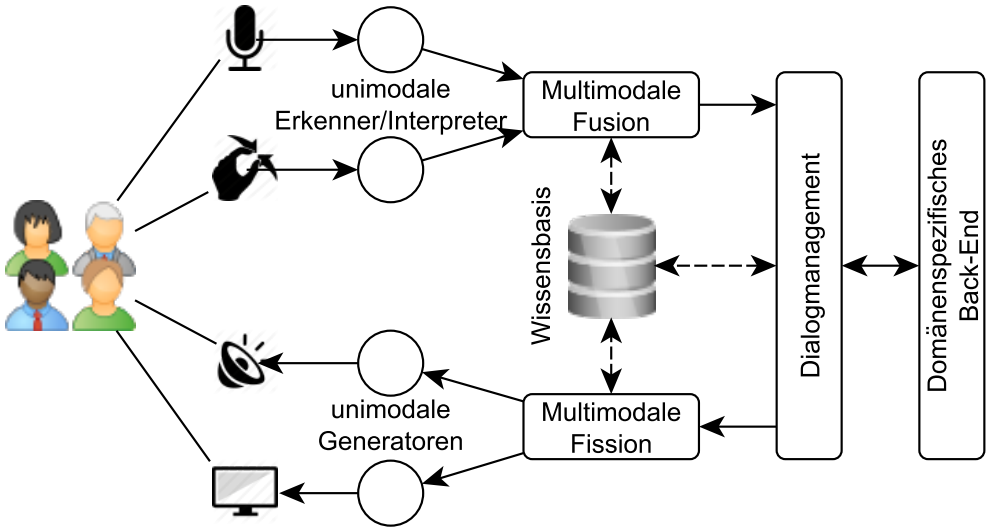


Abbildung 4.4: Konzeptuelle Architektur von multimodalen Dialogsystemen

in der multimodalen Fission. Zuletzt werden einzelne unimodale Generatoren, z. B. für die Synthese von gesprochener Sprache, genutzt, um die multimodale Präsentation für den Benutzer zu erzeugen. Fusion, Fission sowie das Dialogmanagement sind jeweils abhängig von relevanten Kontextfaktoren, die in einer zentralen Wissensbasis vorgehalten werden.

Die **technische Architektur** moderner multimodaler Dialogsysteme forciert im Gegensatz zur konzeptuellen Pipelining-Architektur die lose Kopplung einzelner Module gemäß einer dienstorientierten Architektur (Kapitel 3.4.2, S. 38). Hier kommen Blackboard-Architekturen (Traum und Larsson 2003) oder ereignisbasierte Architekturmuster wie Hub-and-Spoke zum Einsatz - beispielsweise im SmartWeb-System (Reithinger und Sonntag 2005), in ODP (Schehl u. a. 2008) oder im RavenClaw/Olympus-System (Bohus und Rudnicky 2009). Der Hub ist zentraler Bestandteil einer solchen technischen Architektur und verantwortlich für ein schnelles Zustellen von Nachrichten an angemeldete und interessierte fachliche Module. Zur inhaltlichen Kommunikation zwischen einzelnen Modulen haben sich Standards wie z. B. die Extensible MultiModal Annotation Markup Language (Johnston u. a. 2009; Johnston 2009) oder das Media Resource Control Protocol (MRCP) (Burnett und Shanmugham 2012) herausgebildet.

4.3.3 Multimodale Fusion

Die multimodale Fusion integriert (teilweise partielle) Information aus den verschiedenen Eingabekanälen zu einer vollständigen Anweisung. Um die temporalen Abhängigkeiten der anfallenden Informationseinheiten korrekt abzubilden, sind diese mit einem

möglichst exakten Zeitstempel entsprechend ihrer Entstehung zu annotieren. Bei über ein Netzwerk verteilten multimodalen Dialogsystemen stellen nicht synchronisierte Systemuhren und Verzögerungen im Netzwerk subtile Fehlerquellen dar, die es zu vermeiden gilt. Oviatt (2012, S. 419) und Pfleger (2007, S. 88) unterscheiden zwischen früher und später Fusion. Dumas u. a. (2009, S. 12) und Lalanne u. a. (2009, S. 154) untergliedern diese Unterscheidung weiter in Fusion auf der Daten-, Feature-, und Entscheidungsebene. Im multimodalen Dialogsystem SmartKom (Wahlster 2006, S. 14) wird sowohl frühe als auch späte Fusion genutzt.

Eine **frühe Fusion** findet schon auf der Daten- und Signalebene sowie auf der Feature-Ebene zwischen abhängigen und zeitlich sehr eng synchronisierten Eingabemodalitäten statt. So kann sich etwa die zunächst unabhängige Erkennung von Sprache und Lippenbewegungen gegenseitig beeinflussen. Durch die Nutzung von statistischen Verfahren für die Erkennung und Interpretation einzelner Eingabeströme, stellt das Ergebnis eine mit einer Konfidenz behafteten Interpretationshypothese dar.

Die **späte Fusion** integriert unimodale Interpretationshypothesen aus zeitlich und physisch unabhängigen Eingabemodalitäten, wie z. B. Sprach- und Stifteingabe. Dazu müssen diese Interpretationshypothesen bereits so vorverarbeitet worden sein, dass sie in einheitlicher semantischer Form vorliegen. Da diese Interpretationshypothesen prinzipiell unterschiedliche Information transportieren und sich oft ergänzen, kann die späte multimodale Fusion die wechselseitige Disambiguierung der partiellen unimodalen Interpretationshypothesen erreichen. Dazu werden neben schablonenbasierten und hybriden symbolisch/statistischen Verfahren auch unifikationsbasierte Verfahren wie etwa Alexandersson und Beckers (2001, 2003) Overlay eingesetzt. Lalanne u. a. (2009) geben einen guten Überblick über konkrete Implementierungen und weisen darauf hin, dass eine multimodale Anweisung immer kontextabhängig interpretiert werden muss. Das Ergebnis beschreibt eine vollständige semantische Interpretation der Benutzereingabe.

Diskursphänomene

Pfleger (2007, S. 41 ff) beschreibt eine Reihe von Diskursphänomenen, die im Rahmen der (späten) multimodalen Fusion aufgelöst werden können. Solche Phänomene können auch im Rahmen der multimodalen Fission erzeugt werden.

- **Multimodale anaphorische Referenzen** beziehen sich auf einen Satzteil einer vorangegangenen Äußerung, z. B. „*Wann hat Konrad Zuse den Computer erfunden?*“, „*Wo wurde er geboren?*“
- **Multimodale elliptische Referenzen** hingegen lassen eine Information aus, die aus dem Diskurs erschlossen werden muss, z. B. „*Wie wird das Wetter heute?*“, „*Und morgen?*“
- **Multimodale deiktische Referenzen** sind immer kontextabhängig und von einen bestimmten Bezugspunkt aus zu interpretieren. Sie beziehen sich auf vorherige Äußerungen, einen Zeitpunkt bzw. -raum, einen Gesprächspartner oder einen Ort. Wahlster (1991) berücksichtigt außerdem die Genauigkeit von Zeigegesten und beschreibt vor diesem Hintergrund die **Pars-pro-toto**-Deixis. Hierbei

referenziert ein Benutzer durch eine Zeigegeste auf ein (größeres) Objekt. Diese Geste kann entweder auf einen beliebigen Teil des Objekts zeigen oder auf ein bestimmtes eingebettetes Objekt (z. B. eine Schublade in einem Schrank).

- **Cross-modale Referenzen** stellen über eine Modalität Bezug zu Präsentationen in einer anderen Modalität her, z. B. „*Zeige mehr Information zum dritten Listeneintrag.*“

4.3.4 Dialogmanagement

Das Dialogmanagement stellt den Kern eines jeden multimodalen Dialogsystems dar. Traum und Larsson (2003, S. 326) sehen die Aktualisierung des Dialogkontextes, die Bereitstellung von kontextabhängigen Erwartungen für die Interpretation von Benutzereingaben, das Treffen von Entscheidungen welche Inhalte wann dem Benutzer präsentiert werden sollen sowie die Anbindung an ein (domänenspezifisches) Back-End-System als wesentliche Aufgaben des Dialogmanagements an. Je nach Einsatzzweck kann man zwischen verschiedenen Interaktionsstilen und Dialogarten unterscheiden. In Abhängigkeit davon stehen verschiedene Verfahren zum Dialogmanagement zur Verfügung. Ein weiteres Unterscheidungskriterium stellt der Besitz der Dialoginitiative während einer Konversation dar. Diese Aspekte werden im Folgenden erläutert.

Interaktionsstile

Je nachdem, welcher Partner die Konversation steuert, unterscheiden Delgado und Araki (2005, S. 127) zwischen Benutzer- und System-gelenkter Konversation sowie gemischter Initiative.

- **Benutzer-gelenkt**. Der Benutzer ist der aktive Dialogpartner und hat stets die Kontrolle über den Fortgang des Dialogs. Das System ist passiv und reagiert lediglich auf Benutzeranfragen. In diesem Fall ist der Benutzer in der Interaktion mit dem System also weitestgehend frei und wird kaum angeleitet, was aufgrund etwaiger Bedienfehler leicht in ein vermindertes Nutzererlebnis münden kann.
- **System-gelenkt**. Das System leitet den Benutzer durch den Dialog. Dieser reagiert lediglich auf Systemanfragen. Die Gefahr vor Bedienfehlern ist durch eine starke Systemführung reduziert. Ein positives Nutzererlebnis kommt aufgrund der sehr unflexiblen Interaktion trotzdem nur schwer zustande.
- **Gemischte Initiative**. Die Dialogkontrolle und -initiative wechselt zwischen Benutzer und System nach festgelegten Schemata. Erfolgt dieser Wechsel nicht statisch, sondern basierend auf relevanten Kontextfaktoren, Dialoghistorie und erlernten Regeln, so spricht man auch von adaptiver gemischter Initiative. Hierdurch wird eine gute Balance zwischen Nutzererlebnis sowie Effizienz und Effektivität der Benutzerschnittstelle erreicht.

Dialogarten

Anhand des Einsatzzwecks einer multimodalen Dialogschnittstelle kann man im Allgemeinen folgende Dialogarten klassifizieren. In konkreten Systemen sind Mischformen möglich.

- **Kommandoorientierte Dialoge** eignet sich z. B. zur Steuerung von intelligenten Geräten in cyber-physischen Umgebungen, wie z. B. einer instrumentierten, intelligenten Küche (Frey u. a. 2010). Mit den Systemen BabbleTunes (Schehl u. a. 2008) und CoMET (Sonntag u. a. 2009a) hat ein Benutzer Zugriff auf private und öffentliche Mediensammlungen, kann Playlisten erstellen und Inhalte abspielen.
- **Frage-Antwort-Dialoge** dienen zur Befriedigung eines Informationsbedürfnisses des Benutzers. Hierbei werden oft große und komplexe Wissensbasen integriert. Darüber hinaus besteht die Notwendigkeit, Klärungsdialoge zur Disambiguierung oder Einschränkung des Suchraums führen zu können. Beispiele sind die Systeme SmartWeb (Sonntag u. a. 2007) oder CIRIUS (Porta u. a. 2014a). Janzen u. a. (2016) beschreiben unter Anwendung der Theorie rhetorischer Strukturen (Kapitel 4.2.3, S. 52) einen Planungsansatz zur Generierung fairer Antworten in Frage-Antwort-Dialogen mit gemischten, kongruenten und nicht kongruenten Motiven im Kontext von Beratungs- und Verkaufsgesprächen.
- **Formularbasierte Dialoge** ermöglichen es einem System, vom Benutzer strukturiert Information abzufragen und so ein eigenes Informationsbedürfnis vor dem Hintergrund einer im Auftrag des Benutzers durchzuführenden Transaktion zu befriedigen. Hiermit können einfache Aufgaben bewältigt werden.
- **Aufgabenbasierte Dialoge** realisieren komplexe Aufgaben und physische Handlungen, die unterbrochen, wiederaufgenommen oder parallel mit anderen Aufgaben durchgeführt werden können. Ein multimodales Dialogsystem benötigt Fähigkeiten für die Planerkennung, Planverfolgung und Fortschrittsüberwachung.

Dialogmanagementverfahren

Bui (2006) fasst unterschiedliche Ansätze zum Dialogmanagement zusammen.

- **Graphbasierte Ansätze** bilden die Dialogstruktur direkt in Form von Knoten und Kanten ab. Knoten repräsentieren Systemaktionen und -reaktionen, Kanten entsprechen den Benutzereingaben. Die Dialogstruktur ist zur Laufzeit unveränderlich. Zudem werden im Graphen Dialogablauf und Aufgabenmodell miteinander vermischt. Dieser Ansatz wird auch von der in Kapitel 4.4 (S. 66) vorgestellten SiAM-Dialogplattform unterstützt.
- **Ansätze zum Schablonenfüllen** (Slot-Filling) verallgemeinern graphbasierte Ansätze, indem nicht mehr alle Kanten explizit modelliert werden müssen. Stattdessen werden im Rahmen eines formularbasierten Dialogs die zu spezifizierenden Positionen (Slots) einer (domänenspezifischen) Datenstruktur vom System in Kooperation mit einem Benutzer gefüllt. Die Reihenfolge, in der dies geschieht, wird dynamisch, z. B. anhand von Slot-Prioritäten, bestimmt. Schablonenfüllen eignet

sich besonders für transaktionale Dialoge und erlauben gemischte Initiativen, was sie für den Einsatz im Rahmen von KomBInoS qualifiziert. Block u. a. (2004) haben einen Verfahren entwickelt, welches mehrere Slots einer Schablone in einem Zug befüllen kann (Multi-Slot-Filling), wodurch etwa Überbeantwortung ermöglicht wird.

- **Information-State-Ansätze** nach Traum und Larsson (2003) bilden den mentalen Zustand der Gesprächspartner nach. Der sogenannte Informationszustand umfasst folglich u. a. formale Repräsentationen der Gesprächsteilnehmer samt Benutzermodellen, Meinungen, Absichten und Verpflichtungen sowie des bisher erreichten gemeinsamen Verständnisses über die Domäne. Eine Menge von Dialogzügen initiiert eine Aktualisierung des Informationszustandes auf Basis von Regeln, wie sie etwa von Bunt (2011) spezifiziert wurden. TrindiKit (Larsson und Traum 2000) stellt eine Implementierung von Traum und Larssons (2003) Information-State-Ansatz dar.
- **Probabilistische Ansätze** erweitern den Information-State-Ansatz, indem Dialogstrategien in einem statistischen Modell erfasst sind. Dies basiert, wie etwa von Williams u. a. (2008) geschildert, auf Markow-Entscheidungsprozessen sowie auf einem entsprechenden Kostenmodell zur Bepreisung der Zustandswechsel.
- **Planbasierte Ansätze** basieren auf der Annahme, dass Menschen kommunizieren, um bestimmte Ziele zu erfüllen. Die Aufgabe des Zuhörers besteht dabei in der Planerkennung, um eine passende Anwort geben zu können. Dadurch lassen sich noch komplexere Mensch-Maschine-Dialoge gemäß der konversationellen Implikaturen (Kapitel 4.2.5) realisieren. Das Dialogmanangement im SmartKom-System (Löckelt 2006) operiert auf einem Planungsansatz.
- **Agentenbasierte Ansätze** sehen Dialoge als kollaborativen Prozess zwischen intelligenten Agenten an. Agenten arbeiten zusammen, um ein gemeinsames Verständnis über den Dialog zu erlangen und ein gemeinsames Ziel zu erreichen. Dadurch werden Diskursphänomene wie Klärungs- und Bestätigungsdialoge motiviert. Dieses Verfahren wurde im VirtualHuman-System (Reithinger u. a. 2006) eingesetzt.

4.3.5 Multimodale Fission

Sobald der Dialogmanager eine Antwort auf eine Benutzeranfrage konzipiert hat oder von sich aus mit dem Benutzer in Kontakt treten möchte, muss die entsprechende semantische Repräsentation in Form eines Dialogakts an den Benutzer kommuniziert werden. Dies geschieht im Rahmen der multimodalen Fission. Genau wie die Fusion muss auch die Fission den aktuellen Nutzungskontext, Benutzerpräferenzen und -einschränkungen sowie den aktuellen Aufgabenkontext berücksichtigen. Foster (2002) gibt im Rahmen des COMIC-Projekts einen konzeptuellen Überblick über die Aufgaben und die unterschiedlichen Ansätze zur multimodalen Fission. Sie identifiziert drei Hauptaufgabengebiete der Fission.

(1) **Auswahl und Strukturierung des zu präsentierenden Inhalts** auf Basis der Intention des Systems und der zu erzielenden Wirkung beim Benutzer, sprich des illokutionären Anteils des Dialogaktes. Hierzu werden schema- oder planbasierte Verfahren eingesetzt, z. B. auf Basis der Theorie rhetorischer Strukturen (Wahlster u. a. 1993).

(2) Kontextsensitive **Auswahl der Ausgabemodalitäten**.

(3) **Koordination der Ausgabe** bzgl. Layoutplanung von grafischen Ausgaben, Synchronisierung von Modalitäten, die eine fortschreitende Präsentation erfordern, z. B. Sprachausgabe und Animationen eines virtuellen Charakters, sowie die Erzeugung und Synchronisierung von multimodalen und cross-modalen Referenzen.

4.3.6 Wissensbasis und -repräsentation

Die Wissensbasis enthält alle zur Dialogverarbeitung relevanten kontextuellen Informationen. Diese können den persönlichen, sozialen und physischen Kontext umfassen, die auch von Heckmann u. a. (2007) im Rahmen des Ubiquitous User Modeling betrachtet werden. In jedem Fall ist ein Diskursmodell (Pfleger 2007, S. 141ff.) zur kontextbasierten Auflösung von (multimodalen) Referenzen notwendig. Die entsprechenden Metamodelle müssen hierbei mit einem geeigneten Formalismus erstellt worden sein. Generell kann man zwischen domänenspezifischen und dialogspezifischen Wissensbasen unterscheiden. Erstere können durchaus auf Semantic Web Standards wie RDF-Schema oder OWL beruhen. Im Allgemeinen ist eine Mediation heterogener Datenquellen in ein einheitliches dialogspezifisches Format und einen konsistenten Wissensstrom notwendig (Janzen u. a. 2010) (Bergweiler 2014). Hierfür eignen sich diese Standards aufgrund der niedrigen Verarbeitungsgeschwindigkeit, die in der Expressivität der Modellierungssprachen zu suchen ist, nur bedingt. Für eine schnelle Dialogverarbeitung und, damit verbunden, ein positives Nutzererlebnis ist ein guter Kompromiss aus Ausdrucksmächtigkeit und Einfachheit der Modellierungssprachen geboten. Hier haben sich z. B. Carpenters (1992) getypte Merkmalstrukturen bewährt. Pfleger (2007, S. 173) ergänzt getypte Merkmalstrukturen um eindeutige Indentifizierer. In Verbindung mit leichtgewichtigen Algorithmen für Unifikation und Overlay (Alexandersson und Becker 2001, 2003) ergibt sich so ein effizienter Modellierungsansatz von ausreichender Ausdrucksmächtigkeit, der in der anschließend vorgestellten SiAM-Dialogplattform auf Ecore-basierte Metamodelle übertragen werden konnte.

4.4 Die SiAM-Dialogplattform

Die multimodale SiAM-Dialogplattform (Neßelrath und Feld 2014; Neßelrath 2016) stellt eine multimodale Dialogplattform auf Basis moderner Softwareplattformen und Architekturmuster dar. Die technische Basis liefert die Java-basierte Integrationsplattform der Open Service Gateway Initiative (OSGi) (Wütherich u. a. 2008), die aufgrund

ihrer Serviceorientiertheit eine lose Kopplung von Komponenten begünstigt. Die konzeptuelle Architektur der SiAM-Dialogplattform ist in Abbildung 4.5 dargestellt.

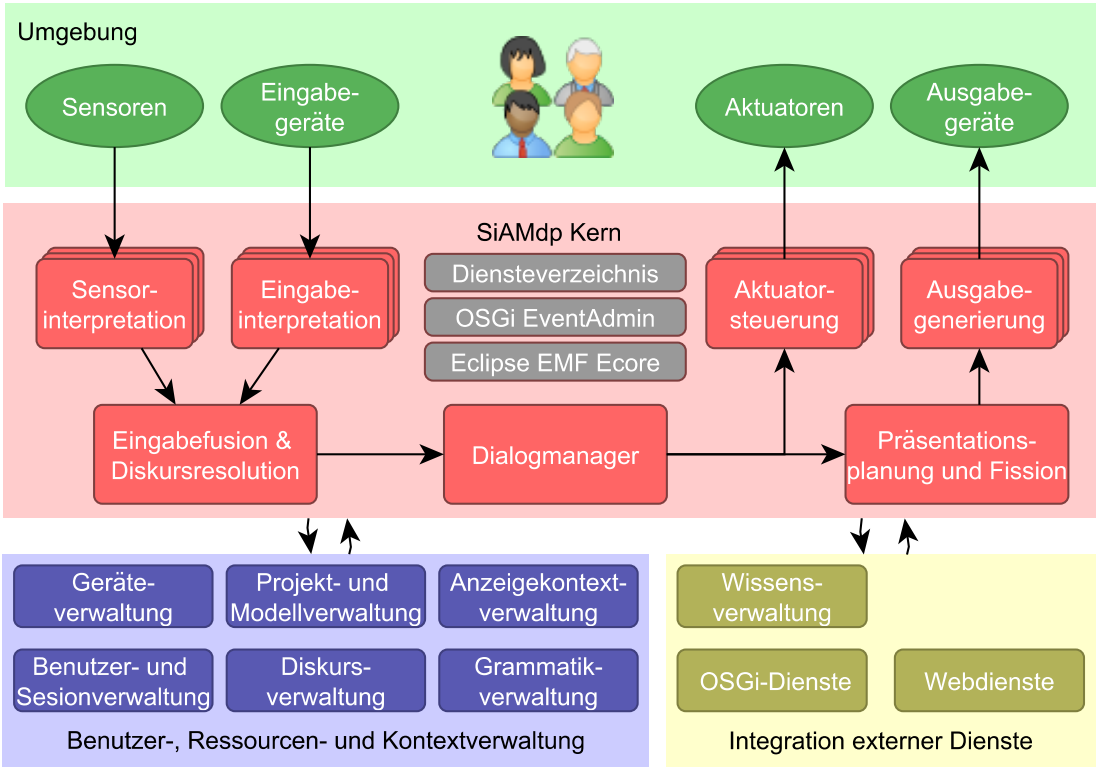


Abbildung 4.5: Die konzeptuelle Architektur der SiAM-Dialogplattform

Alle SiAM-Module sind als OSGi-Komponenten implementiert und über das Diensteverzeichnis auffindbar. Die Kommunikation erfolgt ereignisbasiert über den OSGi-EventAdmin, der in Analogie zum Hub-and-Spoke-Architekturmuster die Rolle des Hub einnimmt. Neue Komponenten lassen sich sehr einfach sogar zur Laufzeit hinzufügen, starten, stoppen und wieder entfernen. SiAM-dp fokussiert auf eine sehr große Anzahl unterschiedlicher Ein- und Ausgabemodalitäten sowie die semantische Verarbeitung von Sensoreingaben und die Ansteuerung unterschiedlicher Aktoren im Rahmen cyber-physischer Systeme. Hierzu ist eine einheitliche und umfassende Basismodellierung notwendig. Betrachtet man die Forschungshistorie, steht sie in einer Reihe mit den Systemen SmartKom (Wahlster 2006), SmartWeb (Sonntag u. a. 2007) und ODP (Löckelt u. a. 2014), deren Konzeptionen in die Entwicklung eingeflossen sind. Dazu zählt auch das von Neßelrath und Porta (2011) sowie Porta u. a. (2014a) beschriebene Vorgehen zur modellbasierten Entwicklung multimodaler Dialogschnittstellen auf Basis von ODP im Rahmen eines Rapid-Prototyping-Ansatzes. Eine einheitliche Basismodellierung wird durch die Integration des EMF in die OSGi-Umgebung sowie die Verwendung des Meta-Metamodells Ecore als Grundlage für SiAM-spezifische Metamo-

delle erreicht. Abbildung 4.6 gibt einen Überblick über die Metamodell-Architektur von SiAM-dp.

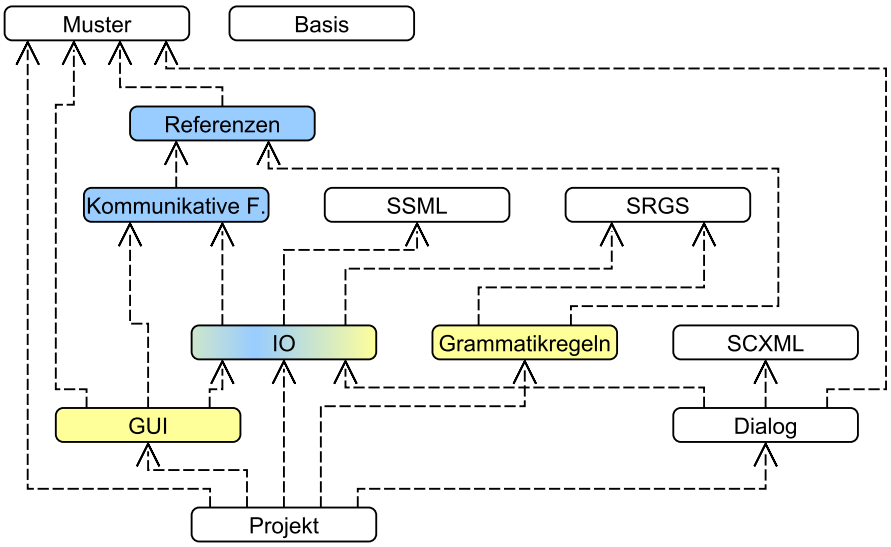


Abbildung 4.6: Die Metamodell-Architektur der SiAM-Dialogplattform

Im Vorgriff auf das in Kapitel 5.2.2 (S. 74) vorgestellte Cameleon Referenzrahmenwerk umfassen blau gekennzeichnete Metamodelle Konzepte für die Dialogaktmodellierung und dienen dieser Arbeit als Basis für das abstrakte UI. Gelb markierte Metamodelle stellen Konzepte des konkreten UI bereit. Das IO-Metamodell enthält Konzepte beider Abstraktionsebenen. Ein Dialogmodell wird zur Laufzeit auf State Chart XML (Barnett u. a. 2015) abgebildet und in einem entsprechenden Verarbeitungsmodul ausgeführt. Die operationelle Semantik (Kapitel 2.4.1, S. 18) eines Dialogmodells ist also auf Basis von Zustandsautomaten definiert. Ebenso werden das Spracheingabemodell in die Speech Recognition Grammar Specification (Hunt und McGlashan 2004) und das Sprachausgabemodell in die Speech Synthesis Markup Language (SSML) (Burnett u. a. 2004) transformiert.

Mit Hilfe von selbst erzeugten Eclipse Editoren können graphbasierte Dialoge sowie umfassende Modelle für die Spracheingabe, die Sprachausgabe und die grafische Benutzerschnittstelle spezifiziert werden. Diesen liegt eine Modellierung von Dialogakten und kommunikativen Funktionen gemäß ISO (2012) (Kapitel 4.2.3, S. 52) zugrunde. Alle Modellartefakte werden zentral in einem Projektmodell zusammengefasst, in welchem auch Benutzer und Ressourcen wie z. B. Interaktionsgeräte spezifiziert werden können.

Ecore ist in Bezug auf die Expressivität äquivalent zu getypten Merkmalstrukturen. Entsprechende Algorithmen für die Unifikation und Overlay wurden auf Basis der Modell-API von Ecore implementiert und werden in Anhang B (S. 271) gesondert erläutert.

4.5 Fazit

In diesem Kapitel wurden die Grundlagen menschlicher Konversation und darauf aufbauend multimodaler Dialogsysteme erläutert, welche die Realisierung synergistischer multimodaler Benutzerschnittstellen ermöglichen. Die Entwicklung einer solchen Benutzerschnittstelle ist aufgrund des darunterliegenden theoretischen Fundaments anspruchsvoll und entsprechend ressourcenintensiv. Durch die Entwicklung und Bereitstellung wiederverwendbarer, generischer multimodaler Dialogplattformen reduziert sich diese Komplexität. Entsprechende Entwicklungsmethoden samt Werkzeugkette, wie von Neßelrath und Porta (2011) und Porta u. a. (2014a) beschrieben, erhöhen den Automatisierungsgrad und lassen die Komplexität auch für Nicht-Experten beherrschbar werden. Zudem steigert sich die Qualität der Benutzerschnittstelle bzgl. Konsistenz und Vollständigkeit. Dies bietet eine gute Ausgangsposition für den Einsatz im Rahmen der Smart Service Welt. Im Rahmen dieser Arbeit stellt die SiAM-Dialogplattform die technologische Zielplatform (Kapitel 2.4.1, S. 19) bereit, welche um zusätzliche Metamodelle, Modelltransformationen sowie Werkzeuge und Laufzeitkomponenten für die (mobile) Nutzung von Smart Services ergänzt wird.

KAPITEL 5

Verwandte Arbeiten

Dieses Kapitel diskutiert verwandte Arbeiten aus den drei paarweisen Querschnittsthemen, der zuvor vorgestellten Forschungsgebiete. Auf dieser Grundlage wird ein Anforderungskatalog für diese Arbeit abgeleitet, der auch zum Vergleich der verwandten Arbeiten dient.

5.1 Einleitung

Dieses Kapitel diskutiert ausgewählte verwandte Arbeiten aus den Querschnittsthemen der in den vorherigen Kapiteln vorgestellten Forschungsgebiete, wie einleitend in Kapitel 1.2.2 (S. 3) erläutert. Die *modellgetriebene Entwicklung von intelligenten Benutzerschnittstellen* stützt sich häufig auf das Cameleon Referenzrahmenwerk. Aber auch andere Arbeiten auf diesem Gebiet liefern wertvolle und teils komplementäre Lösungsansätze, die für diese Arbeit von Bedeutung sind. Die *Komposition von Benutzerschnittstellen für Dienste* wird in verwandten Arbeiten unterschiedlich interpretiert. Hier müssen zuerst die prinzipiellen Möglichkeiten und Ausprägungen eines Kompositionsansatzes erörtert werden, bevor sich diesen Arbeiten zugewandt werden kann. Die ausgewählten *ganzheitlichen Methoden zur Dienstentwicklung* setzen die ingenieurmäßige Dienstentwicklung einerseits in einen größeren arbeitsorganisatorischen Kontext und zeigen auf technischer Ebene, wie eine integrierte, nahtlose Werkzeugkette aussehen kann.

Anhand der Stärken und Schwächen der verwandten Arbeiten wird ein Anforderungskatalog für die eigene Arbeit abgeleitet. Gleichzeitig dienen die Anforderungen als Kriterien für einen Vergleich der verwandten Arbeiten.

5.2 Modellgetriebene Entwicklung von intelligenten Benutzerschnittstellen

Dieses Unterkapitel beschäftigt sich mit der ingenieurmäßigen Erstellung von multimodalen Dialogschnittstellen. Für Vanderdonckt (2005, S. 18) ist dieses *User Interface Engineering* ein Querschnittsthema aus Softwareentwicklung, Mensch-Computer-Interaktion und menschlichen Faktoren bzgl. der Ergonomie. Oberstes Ziel sei die Entwicklung einer Methodik, mit der sich Benutzerschnittstellen entlang eines Lebenszyklus und mit Hilfe von etablierten Konzepten der Softwareentwicklung realisieren lassen. Zu dieser Methodik gehören

(1) eine Reihe von **Metamodellen**, die die verschiedenen Aspekte von Benutzerschnittstellen adäquat repräsentieren können.

(2) ein strukturierter **Entwicklungsprozess**, der beschreibt, wie die Metamodelle einzusetzen und zu erstellen sind. Dieser berücksichtigt auch Abhängigkeiten zwischen einzelnen Metamodellen.

(3) **Werkzeugunterstützung** zur Anwendung der Methodik.

Dies sind Kernelemente der modellbasierten Entwicklung von Benutzerschnittstellen und für rein graphische Benutzerschnittstellen, die nach dem klassischen Bedienkonzept *Fenster, Icon, Menüs und (Maus-) Zeiger* funktionieren, bereits gut erforscht, etwa im Rahmen einer entsprechenden W3C-Inkubatorgruppe[1]. Punkt (2) wird im nächsten Abschnitt anhand der benutzerzentrierten *Elements of User Experience* vertieft. Anschließend wird das Cameleon Referenzrahmenwerk vorgestellt, welches die oben angeführten Punkte (1) und teilweise (2) von einem konzeptionellen Standpunkt aus abdeckt und heutzutage meist als Ausgangspunkt für konkrete ganzheitliche Methodiken dient, die auch insbesondere Punkt (3) berücksichtigen. Entsprechende Methodiken werden zum Ende des Unterkapitels behandelt. Ein besonderes Augenmerk wird auch auf Arbeiten zur Erstellung von multimodalen Dialogschnittstellen speziell für mobile Endgeräte und Dienste liegen.

5.2.1 Garrett's Elements of User Experience

Garretts (2010) *Elements of User Experience* beschreiben einen systematischen und benutzerzentrierten Ansatz zur Erstellung von Benutzerschnittstellen. Im Rahmen des TEXO-Projekts wurde diese Herangehensweise als Methode zur Entwicklung von mobilen multimodalen Dialogschnittstellen für das Internet der Dienste adaptiert und angewendet (Sonntag u. a. 2010b). Garrett gliedert die für ein positives Nutzererlebnis wichtigen Aspekte in fünf aufeinander aufbauende Ebenen. Diese sind in Abbildung 5.1 dargestellt. Innerhalb dieser Ebenen kann die Benutzerschnittstelle schrittweise verfeinert werden. Die unterste und zugleich abstrakteste Ebene ist die **Strategieebene**.

[1] siehe http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/ (letzter Zugriff: 04.11.2017)

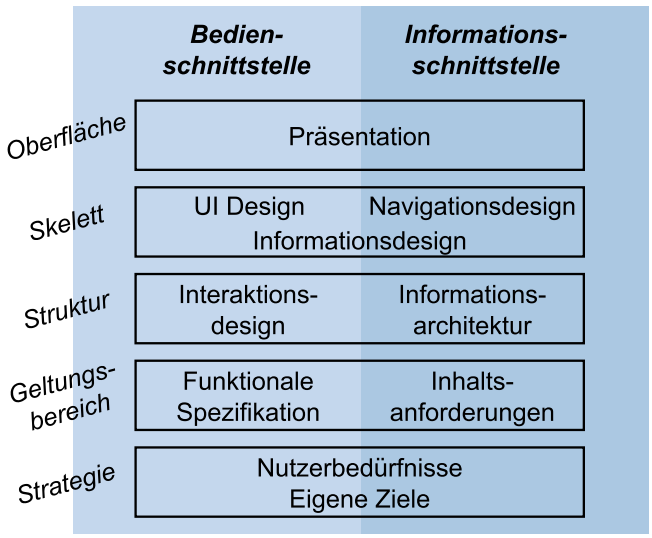


Abbildung 5.1: Die Elemente für ein positives Nutzererlebnis nach Garrett (2010)

Darin werden Unternehmensziele und Benutzerbedürfnisse identifiziert. In der Ebene des **Geltungsbereichs** werden diese für eine spätere Umsetzung priorisiert und gegebenenfalls fallengelassen. Es folgen die **Strukturebene** und die **Skelettebene**. Mit jeder neuen Ebene nimmt die ursprünglich abstrakte Idee konkretere Züge an. Schließlich beschreibt die **Oberflächenebene** detailliert das Aussehen der Anwendung. Da eine konkretere Ebene von den darunter liegenden abhängt, werden alle Entscheidungen auf diesen Ebenen immer konsequent nach oben propagiert. Dies bedeutet, dass die Entscheidungsfindung auf einer Ebene immer konform zu den bereits getroffenen Entscheidungen auf den Ebenen darunter sein muss.

Eine Benutzerschnittstelle kann einerseits als reine **Bedienschnittstelle** betrachtet werden, die die Steuerung einer Maschine oder die Durchführung von Aufgaben ermöglicht. Andererseits kann eine Benutzerschnittstelle aber auch als **Informationsschnittstelle** aufgefasst werden, die etwa den aktuellen Status einer Maschine oder einer Aufgabe kommuniziert. Diese Standpunkte sind gegenüber dem Benutzer untrennbar miteinander verknüpft. Während der Entwicklung der Benutzerschnittstelle kann man diese Sichten jedoch teilweise getrennt betrachten. Aufgrund der beschriebenen Dualität teilt Garrett jede Ebene in zwei Teile, was durch die unterschiedliche Farbgebung in Abbildung 5.1 zum Ausdruck gebracht wird. Der linke Teil betrachtet eine Ebene aus Sicht der Bedienschnittstelle, der rechte aus Sicht der Informationsschnittstelle. Die Strategieebene sowie das Informationsdesign innerhalb der Skelettebene müssen aus beiden Blickwinkeln gleichermaßen betrachtet werden.

Die Strukturierung einer Benutzerschnittstelle in mehrere aufeinander aufbauende Ebenen und dazu orthogonalen Sichten gemäß dem Teilen-und-Herrschen-Prinzip ist (nicht nur) für eine methodische Herangehensweise bei der modellgetriebenen Entwick-

lung einer UI sehr sinnvoll. Auf diese Weise lässt sich der Problemraum bzw. die Entwicklung in mehrere handhabbare Teilprobleme und -modelle zerlegen, die obendrein von Spezialisten koordiniert bearbeitet werden können. Dieses Muster wird darum in den folgenden verwandten Arbeiten häufig erkennbar sein und ist auch für die in dieser Dissertation beschriebene Methode grundlegend.

5.2.2 Das Cameleon Referenzrahmenwerk

Als konzeptuelle Spezialisierung der MDA (Kapitel 2.5, S. 23) definieren Calvary u. a. (2003) mit dem Cameleon Referenzrahmenwerk (CRF, Cameleon Reference Framework) auf konzeptioneller Ebene eine breit akzeptierte Methode zur modellbasierten Entwicklung von kontextsensitiven, plastischen Benutzerschnittstellen. Eine plastische Benutzerschnittstelle gewährleistet nach Calvary u. a. (2003, S. 4) eine adäquate Benutzerfreundlichkeit in zur Entwurfszeit berücksichtigten Nutzungskontexten. Die Benutzerfreundlichkeit zeichnet sich durch eine Menge von Eigenschaften bzw. Bedingungen aus, die zur Entwurfszeit ausgewählt wurden und zur Laufzeit in verschiedenen Nutzungskontexten überwacht und eingehalten werden können. Der Nutzungskontext selbst setzt sich auf der einen Seite aus einem zur Entwurfszeit vorhersehbaren Anteil und einer Laufzeitkomponente zusammen. Auf der anderen Seite unterscheidet der Nutzungskontext stereotype Benutzer, Faktoren bzgl. der eingesetzten Hard- und Softwareplattform sowie Faktoren bzgl. der physikalischen Nutzungsumgebung. Daraus motiviert sich u. a. die Wahl der zu unterstützenden und zur Laufzeit angebotenen Ein- und Ausgabemodalitäten. Abbildung 5.2 stellt das CRF in einer vereinfachten Form grafisch dar.

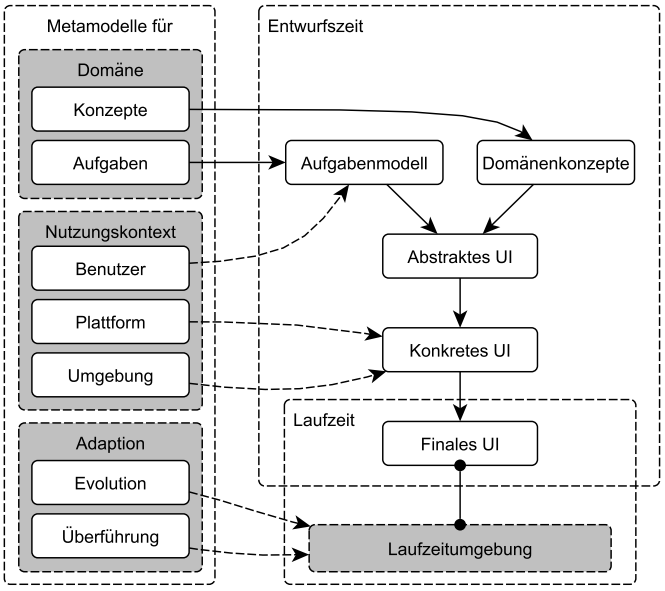


Abbildung 5.2: Das Cameleon Referenzrahmenwerk nach Calvary u. a. (2003, S. 294) und Stanciulescu u. a. (2005, S. 260)

Auf der linken Seite werden die eingesetzten ontologischen Metamodelle (Kapitel 2.2.2, S. 14) anhand der Gruppen Domäne, Nutzungskontext und Laufzeitadaption gegliedert. Die Gruppe der Domänen-Metamodelle ermöglicht die Domänenmodellierung bzgl. der auftretenden Konzepte und der vom Benutzer zu erledigenden Aufgaben. Die Metamodelle zur Formalisierung des Nutzungskontextes leiten sich direkt aus den im vorherigen Absatz erläuterten Bestandteilen des Nutzungskontextes ab. Metamodelle für die Adaption ermöglichen die Beschreibung von Adaptionsregeln und Hilfestellungen, um den Kontextwechsel und die damit verbundene Adaption der Benutzerschnittstelle für den Benutzer möglichst transparent und reibungslos zu gestalten.

In Abbildung 5.2 befinden sich rechts verschiedene Instanziierungen der Metamodelle für verschiedene Nutzungskontexte samt skizziertem Entwicklungsprozess. Ausgehend von einer formalisierten benutzerzentrischen Aufgabenmodellierung und einer Domänenmodellierung erhält man im Rahmen eines bestimmten Nutzungskontextes über Modelltransformationen ein plattformunabhängiges abstraktes UI (AUI) und schließlich ein plattformabhängiges konkretes UI (CUI), aus welchem letztlich durch Codegenerierung eine finale Benutzerschnittstelle erzeugt werden kann. Die Modelltransformationen können automatisiert, manuell oder semiautomatisch durchgeführt werden.

Eine Laufzeitumgebung, welche in Abbildung 5.2 unten dargestellt ist, überwacht relevante Kontextfaktoren und kann auf dieser Entscheidungsgrundlage die Adaption der Benutzerschnittstelle anstoßen und durchführen.

Auf Basis des CRF wurden konkrete Instanziierung entwickelt, z. B. von Mori u. a. (2004) und Limbourg u. a. (2005). Üblicherweise resultieren aus den Transformations- und Generierungsprozessen in sich geschlossene, proprietäre Benutzerschnittstellen. Einige Ansätze betrachten aber auch browsergestütze UIs (Stanciulescu u. a. 2005) und UIs für Dienste (Paternò u. a. 2009) und Prozesse (Sousa u. a. 2008, 2010). Parallel dazu wird auch verstärkt die Einbeziehung mehrerer Modalitäten berücksichtigt (Heinrich u. a. 2007; Paternò u. a. 2008; Ertl 2009). Synergistische Multimodalität sowie Diskursphänomene werden im Gegensatz zur vorliegenden Arbeit jedoch nicht betrachtet. Hierzu fehlen in diesen Ansätzen entsprechende Verarbeitungskomponenten für die multimodale Fusion und Fission sowie ein adäquates Dialogmanagement auf Basis von Dialogakten.

Die folgenden Abschnitte behandeln UsiXML, MARIA, EMODE und Useware Engineering als (teilweise multimodale) Instanziierungen des CRF.

5.2.2.1 UsiXML und MultiXML

Auf Grundlage von XML haben Limbourg u. a. (2004) die User Interface Extensible Markup Language (UsiXML) zur vollständigen deklarativen Beschreibung von Benutzerschnittstellen in unterschiedlichen Nutzungskontexten mit entsprechender Unterstützung für verschiedene Modalitäten konzipiert. UsiXML basiert dabei explizit auf dem CRF.

Im Kontext von UsiXML betrachten Stanciulescu u. a. (2005) insgesamt vier Moda-

litätsklassen, also diskrete Abstufungen der zur Verfügung stehenden Modalitäten, von rein grafischen über vorwiegend grafische bzw. sprachliche bis hin zu rein sprachlichen Benutzerschnittstellen, wie in Abbildung 5.3 gezeigt.

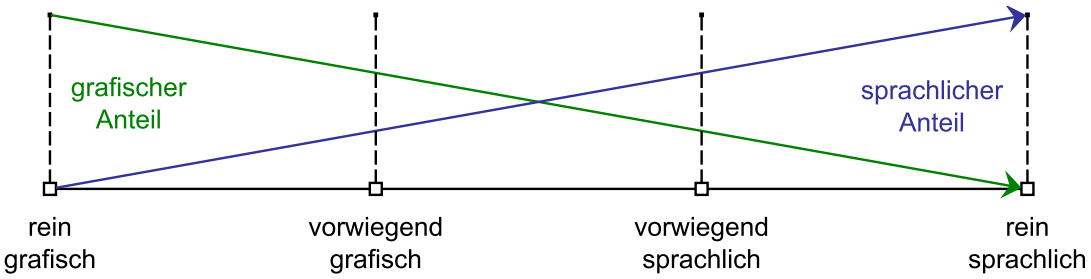


Abbildung 5.3: Modalitätsklassen in MultiXML nach Stanciulescu u. a. (2005, S. 263)

Diese Modalitätsklassen finden sich auch in MultiXML (Stanciulescu und Vanderdonckt 2007) wieder, einer Erweiterung von UsiXML zur Entwicklung von browserbasierten multimodalen Benutzerschnittstellen. Der Entwicklungsprozess durchläuft grob die vier in Abbildung 5.2 gezeigten Entwicklungsstadien des CRF. Das finale UI wird in UsiXML zwar nicht mehr betrachtet (Sousa u. a. 2007, S. 116). Dennoch lassen sich mit den um UsiXML herum entwickelten Werkzeugen alle Entwurfs- und Entwicklungstätigkeiten bis zum konkreten UI durchführen. Abbildung 5.4 zeigt eine Übersicht über den Entwicklungsprozess in MultiXML und die verwendeten Werkzeuge.

IdealXML dient initial zum Erstellen eines Aufgabenmodells und eines Domänenmodells. MultiXML stellt dem Designer verschiedene Abbildungen und Transformationen zur Verfügung, um Konzepte in unterschiedlichen Abstraktionsebenen aufeinander abzubilden. Umwandlungen von einer Abstraktionsebene zur nächsten werden als Graph-Transformationen auf dem UsiXML-Modell ausgeführt. Dazu werden Regeln definiert, die bestimmen, wie ein UI sukzessive konkretisiert wird. Diese Regeln werden in der Transformationssprache TransformiXML spezifiziert. Das abstrakte UI besteht aus mehreren abstrakten Containern, in denen abstrakte individuelle Komponenten des UIs zusammengefasst werden. Diese Komponenten repräsentieren grundlegende Interaktionsfunktionen, die jedoch noch modalitätsunabhängig definiert sind, z. B. die Auswahl von Elementen. Auf konkreter Ebene werden die abstrakten Komponenten nun modalitätsspezifisch. In MultiXML wird zwischen grafischen und natürlichsprachlichen Interaktionsobjekten unterschieden. Neuere Versionen betrachten auch stiftbasierte Gesten (Beuvens und Vanderdonckt 2012, 2014). Abhängig vom jeweiligen Nutzungskontext wird ein abstraktes UI in mehrere konkrete UIs überführt. Für den letzten Schritt zum finalen UI müssen Abbildungen zu existierenden Präsentations- und Interaktionstechnologien geschaffen oder teils kommerzielle Werkzeuge verwendet werden. So erzeugt GrafiXML aus konkreten grafischen UI-Modellen HTML-Code, der in einem Webbrowser gerendert werden kann. Mithilfe des im IBM WebSphere Voice Toolkit enthaltenen

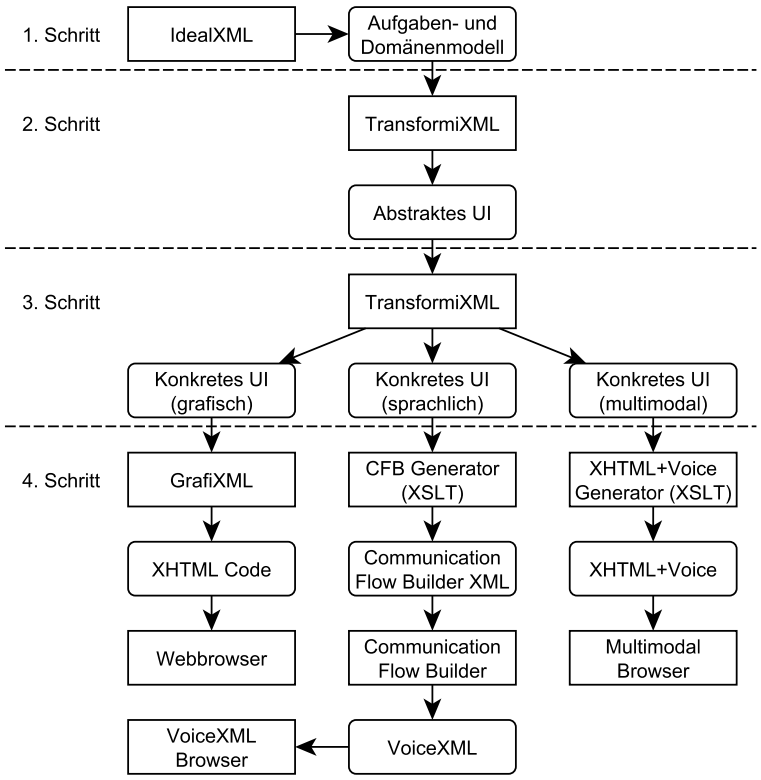


Abbildung 5.4: Entwicklungsprozess und Werkzeugunterstützung in MultiXML nach Stanciulescu und Vanderdonckt (2007, S. 52)

Communication Flow Builders[2] können aus konkreten sprachlichen UI-Modellen nach vorhergehender Extensible Stylesheet Language (XSL)-Transformation sprachbasierte Benutzerschnittstellen auf der Basis von VoiceXML (McGlashan u. a. 2004) erzeugt werden. Aus multimodalen UI-Modellen können durch einen Generator anhand von XSL-Transformationen exklusive bzw. nebenläufige multimodale Benutzerschnittstellen auf Basis von XHTML+Voice (Axelsson u. a. 2001) generiert werden.

Sousa u. a. (2008, 2010) nutzen UsiXML zur modellgetriebenen Entwicklung von Prozess-UIs. Die zugrundeliegende Methode ist in Abbildung 5.5 dargestellt.

Die Methode nutzt initial in (1) ein Prozessmodell und das entsprechende Datenmodell des Prozesses, welche aus UsiXML-Perspektive als externe Modelle angesehen werden. Mit Hilfe von Werkzeugen werden diese Modelle in (2) auf das Aufgabenmodell und das Domänenmodell abgebildet. Fortan stehen in (3) - (6) die in Abbildung 5.4 gezeigten Werkzeuge zur UI-Entwicklung auf Basis der Prozessbeschreibung zur Verfügung. Vorbehaltlich einer adäquaten Werkzeugunterstützung sehen Sousa u. a. (2008, S. 558)

[2]siehe http://pic.dhe.ibm.com/infocenter/pvcvoice/51x/index.jsp?topic=/com.ibm.voicetools.callflow.doc/ccfcommflow.html (letzter Zugriff: 04.11.2017)

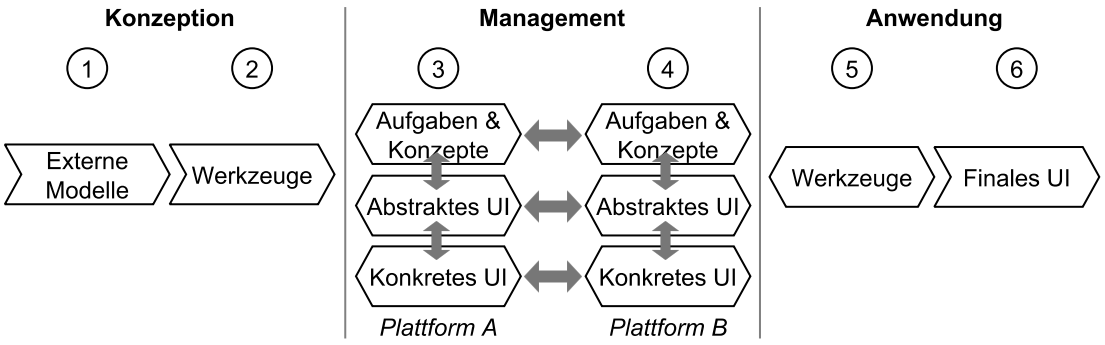


Abbildung 5.5: Entwicklungsmethode zur Erstellung von Prozess-UIs mit Hilfe von UsiXML nach Sousa u. a. (2008, S. 557)

bei diesem Vorgehen Vorteile bzgl. der konsistenten und schnellen Propagierung von mit der Zeit erforderlichen Prozessänderungen in die Benutzerschnittstelle. Im Gegensatz zur vorliegenden Arbeit lassen sie die Komposition solcher Benutzerschnittstellen bisher jedoch außer Acht.

Aufgrund der Popularität von UsiXML sind zahlreiche Werkzeuge entstanden. Auf eine gemeinsame, leistungsfähige Integrationsplattform, wie die in dieser Arbeit verwendete Eclipse-Plattform, wurde dennoch verzichtet. Zudem werden keine im Rahmen der MDA standardisierten Sprachen zur Modellbildung und -transformation, wie in Kapitel 2.5 (S. 23) beschrieben, verwendet, obwohl Vanderdonckt (2005) UsiXML als konform zur MDA sieht.

Aufgrund der Graphstruktur einer UsiXML-Anwendung und des Fehlens sonstiger Strukurierungsmittel sind Inhalte der einzelnen Abstraktionsebenen über den gesamten Graphen verstreut. Die Transformationsregeln werden gerade in späteren Entwicklungsschritten schnell unübersichtlich. Da die Transformationsregeln auch Teile des bestehenden Graphen ersetzen können, ist schwer nachvollziehbar, welche Elemente in welchem Schritt hinzugefügt oder geändert wurden.

Die starre Einteilung in vier Modalitätsklassen, wie in Abbildung 5.3 dargestellt, erschwert die Unterstützung weiterer Modalitäten in einem massiv multimodalen Anwendungsfall. Entweder wird schon zur Entwurfszeit eine Modalitätsklasse festgelegt oder es müssen gleich vier verschiedene konkrete UI-Modelle und finale Benutzerschnittstellen erzeugt werden. Die Notwendigkeit von Laufzeitkomponenten zur multimodalen Fusion und Fission wird zwar von Stanciulescu und Vanderdonckt (2007, S. 43) erkannt, aber im Ansatz nicht weiter berücksichtigt, weshalb die so geschaffene Multimodalität lediglich zwei der vier CARE-Eigenschaften (Kapitel 4.3.1, S. 59), Äquivalenz und Zuordnung, realisiert. Die in dieser Arbeit entwickelte Laufzeitumgebung basiert auf der massiv multimodalen SiAM-Dialogplattform, welche es einem Benutzer im Rahmen des Projekts MADMACS[3] ermöglichte, bis zu neun Modalitäten nach Belieben zu

[3] siehe https://madmacs.dfki.de/ (letzter Zugriff: 04.11.2017)

kombinieren (Neßelrath 2016, S. 214).

5.2.2.2 MARIA und TERESA

Paternò u. a. (2009) beschreiben MARIA als eine universelle modellbasierte Sprache auf mehreren Abstraktionsebenen für interaktive dienstorientierte Anwendungen in ubiquitären Umgebungen. MARIA wurde im EU-Projekt ServFace auf der Basis von TERESA (Mori u. a. 2004) weiterentwickelt und kann nun Operationen von WSDL-basierten Webdiensten aus der Benutzerschnittstelle heraus ansprechen und diese durch UI-Annotationen im WSDL-Dokument in Form eines Dienste-Front-Ends einbinden. Des Weiteren erweitert MARIA TERESA um ein flexibles Datenmodell sowie ein Ereignismodell. MARIA verfolgt prinzipiell einen CRF-konformen Entwicklungsansatz. Auf Ebene des Aufgabenmodells kommen ConcurTaskTrees (Paternò u. a. 1997) zum Einsatz. Darunter spezifiziert MARIA eigene Metamodelle für AUI und CUI. Diese Modelle können durchgängig in einer integrierten Autorenumgebung, genannt MARIAE[4] (Paternò u. a. 2011b) und gezeigt in Abbildung 5.6, erstellt werden. MARIA ist wie TERESA eine XML-basierte deklarative Sprache. Folglich werden XSL-Transformationen zur Modelltransformation eingesetzt. MDA-Standards werden nicht benutzt.

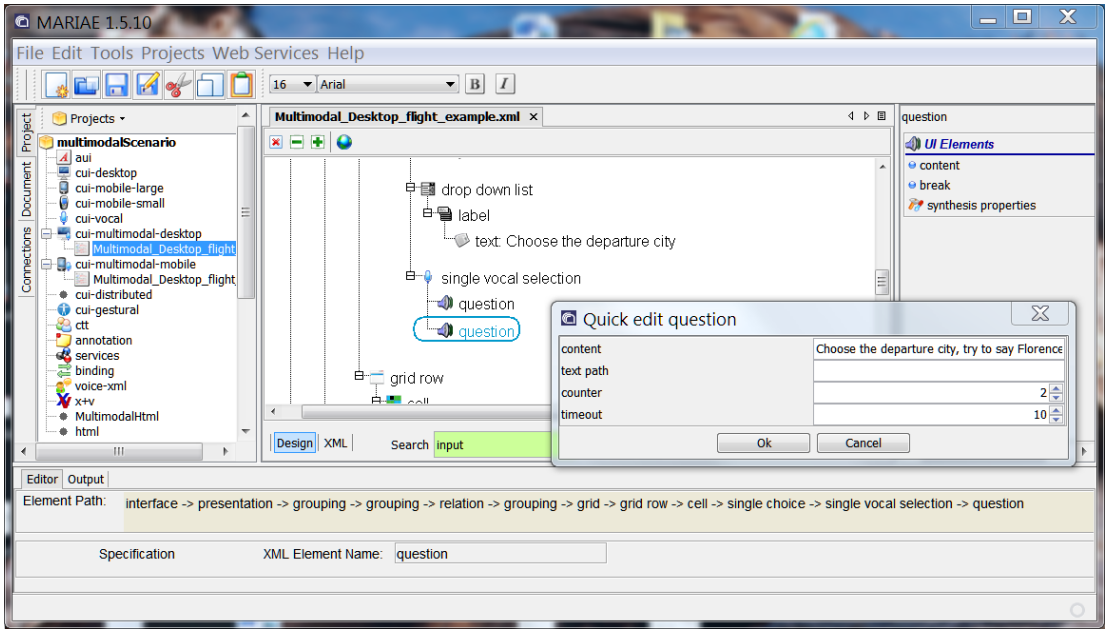


Abbildung 5.6: MARIAE, ein integrierter Editor für MARIA-Anwendungen

Die Nutzbarkeit von Modalitäten ist abhängig von der avisierten Zielplattform der finalen Benutzerschnittstelle. Neben grafischen und natürlichsprachlichen UIs werden

[4]siehe http://hiis.isti.cnr.it:4500/research/MARIAE/home (letzter Zugriff: 04.11.2017)

auch multimodale UIs im Sinne einer Kombination aus Grafik und Sprache betrachtet. Paternò und Giammarino (2006, S. 331) behaupten, alle CARE-Eigenschaften (Kapitel 4.3.1, S. 59) abdecken zu können. Bei genauerer Betrachtung stellt sich jedoch heraus, dass durch einen definitorischen Kniff die Redundanz-Eigenschaft[5] zur Äquivalenz-Eigenschaft und die Äquivalenz-Eigenschaft zur Zuordnungs-Eigenschaft[6] degradiert werden. Außerdem fällt auf, dass die Komplementarität lediglich auf Seite der Systemausgabe berücksichtigt wird. Ohne den Einsatz von Komponenten für die multimodale Fusion und Fission wird eine etwaig synergistisch genutzte Multimodalität auf Ausgabeseite lediglich durch plattformspezifische Muster erzeugt.

Aufgrund der Einbindung bereits existierender Funktionalität in Form von Webdiensten merken Paternò u. a. (2009, S. 19:21) an, dass ein bisheriger Top-Down-Ansatz nicht zweckmäßig sei. Stattdessen müsse dieser mit einem Bottom-Up-Ansatz zur Formalisierung von zu verwendender Dienstfunktionalität in der eigenen Notation zu einem hybriden Ansatz verknüpft werden. Dazu werden generelle Anforderungen an eine Komposition gestellt, die anhand eines entsprechenden Designraums in Kapitel 5.3.1 (S. 90) erläutert wird.

MARIA unterstützt den vorgestellten Designraum durch verschiedene Operatoren, um Benutzerschnittstellen zu komponieren. Gruppierungsoperatoren fassen Interaktionselemente in Gruppen zusammen, Verbindungsoperatoren verbinden mehrere Ansichten oder Präsentationen miteinander. Dabei unterscheidet man wiederum zwischen elementaren Verbindungen und konditionalen Verbindungen bei denen abhängig vom Wert eines bestimmten Parameters unterschiedliche Präsentationen aufgerufen werden. Durch diese Operatoren können Entwickler eine Komposition immerhin manuell erzeugen. Ein semiautomatischer Ansatz ist durch die Werkzeugkette nicht realisiert.

5.2.2.3 EMODE

Hamann u. a. (2008) beschreiben einen zum CRF konformen modellgetriebenen Ansatz zur Entwicklung multimodaler kontextsensitiver Anwendungen im Rahmen des EMODE-Projekts, der technisch primär auf MDA-Standards (MOF-basierte Metamodelle und QVT) sowie einer Werkzeugkette auf Basis von Eclipse beruht. Abbildung 5.7 zeigt ein beispielhaftes Aufgabenmodell in EMODE.

In einem solchen werden Benutzer-, System- und Interaktionsaufgaben spezifiziert, die wiederum mit Hilfe von abstrakten Aufgaben zu komplexeren Aufgaben zusammengefasst werden können. Durch Ein- und Ausgabe-Pins können Datenflüsse über Aufgaben hinweg modelliert werden. Externe Ereignisse und Kontextinformation können, genau wie fachliche Information, in einzelnen Aufgaben berücksichtigt und verarbeitet

[5] Die Gleichzeitigkeitsbedingung des Modalitäteneinsatzes der Redundanz-Eigenschaft wird vernachlässigt.

[6] Es werden nur zwei Modalitäten betrachtet, von denen eine exklusiv („*either ... or*“) vorgesehen wird. Somit ist die Kardinalität der Menge der äquivalenten Modalitäten eins, was in diesem Fall der Zuordnungs-Eigenschaft entspricht.

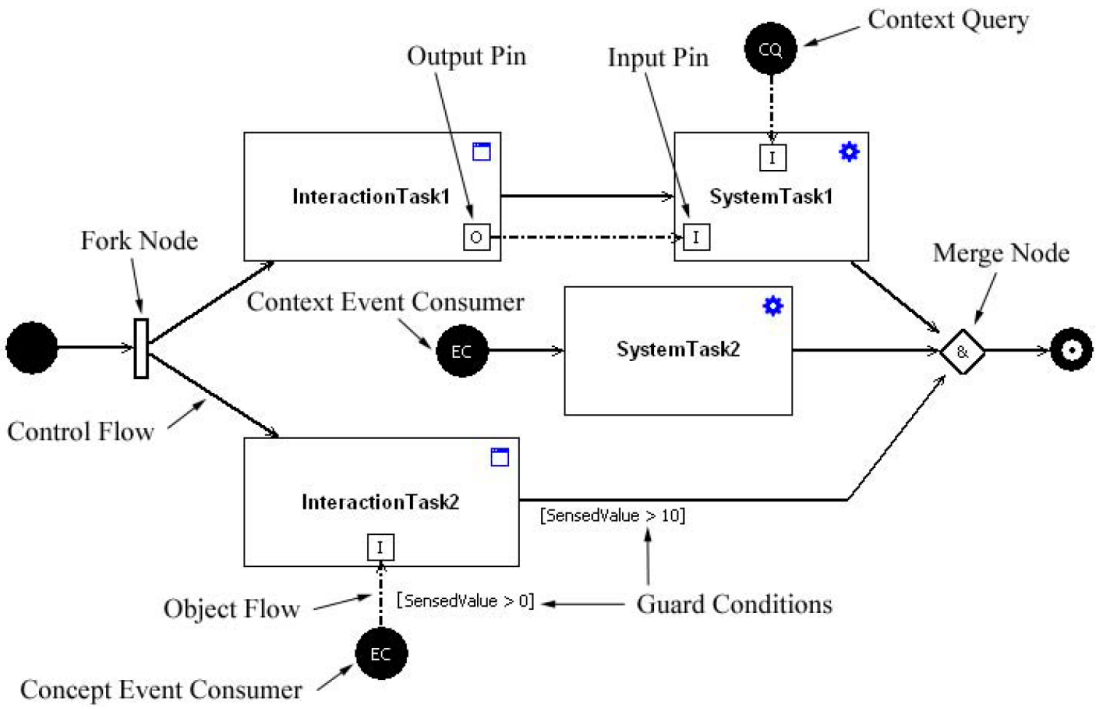


Abbildung 5.7: Ein Aufgabenmodell in EMODE (Hamann u. a. 2008, S. 202)

werden. Aufgabendefinitionen als Analogie zu elementaren Aktionstypen in der UML-Activity verfeinern und bestimmen Aufgaben auf semantischer Ebene näher (Heinrich u. a. 2007, S. 18). Auf Basis der im Aufgabenmodell enthaltenen Systemaufgaben werden zu implementierende Adapter generiert, die die Geschäftslogik der UI einbinden können. Anhand der spezifizierten Interaktionsaufgaben wird ein abstraktes UI-Modell abgeleitet, welches als DialogueSpace-Modell bezeichnet wird. Dieses wird nun, im Gegensatz zu anderen CRF-basierten Verfahren, nicht in ein konkretes UI-Modell transformiert. Stattdessen wird das Modell direkt sukzessive modalitätsspezifisch verfeinert. Laut Behring u. a. (2007, S. 36) stellt EMODE durch die Auflösung der strikten Trennung von AUI und CUI in das DialogueSpace-Modell eine Möglichkeit bereit, UIs auf beliebigem Abstraktionsniveau zu beschreiben, und diese schrittweise über mehr als eine Stufe zu verfeinern. Dieses Vorgehen findet sich auch in der Dialog and Interface Specification Language (DISL) (Mueller u. a. 2004) wieder. Heinrich u. a. (2007, S. 22) führen an, dass zur Laufzeit Fusions- und Fissionsalgorithmen bereitstünden. Einen Anwendungsfall im Rahmen des geschilderten Beispiels bleiben die Autoren jedoch schuldig. Multimodalität bezieht sich laut Behring u. a. (2007, S. 31) lediglich auf unterschiedliche unimodale Ausprägungen einer UI auf unterschiedlichen Zielplattformen. Alle durch die Werkzeugkette erstellten Modelle werden zentral in einer Modellablage abgelegt, eine Wiederverwendung der Modelle im Rahmen einer Komposition wird jedoch nicht adressiert. Die Aufgabenmodelle spezifizieren Geschäftsprozesse, die in einer eigens entwickelten Pro-

zessausführungsmaschine ablaufen. Die BPMN 2.0 wurde zwar erst nach Projektende von EMODE verabschiedet, dennoch wurde die Nutzung anderer ausführbarer Prozessmodellierungsstandards nicht in Erwägung gezogen. Know-How aus EMODE floss auch in die Entwicklung des ISE-Rahmenwerks (Kapitel 5.4.1, S. 101) ein.

5.2.2.4 Useware Engineering

Useware ist laut Zühlke (2004, S. 2) ein Sammelbegriff für Hard- und Softwarekomponenten eines technischen Systems, die der Benutzung dienen. Folglich versteht man unter Useware-Engineering die standardisierte ingenieurmäßige Herstellung von Useware und die damit verbundenen Prozesse. Vor allem in Bezug zu industrienahen Benutzer- und Maschinenbedienschnittstellen stehen Fragen der Effizienz und der Integration der Bedienschnittstellen in die physische Arbeitsumgebung der Anwender im Fokus. Der Useware Engineering Prozess (Meixner u. a. 2011; Zühlke 2012) ist in Abbildung 5.8 dargestellt.

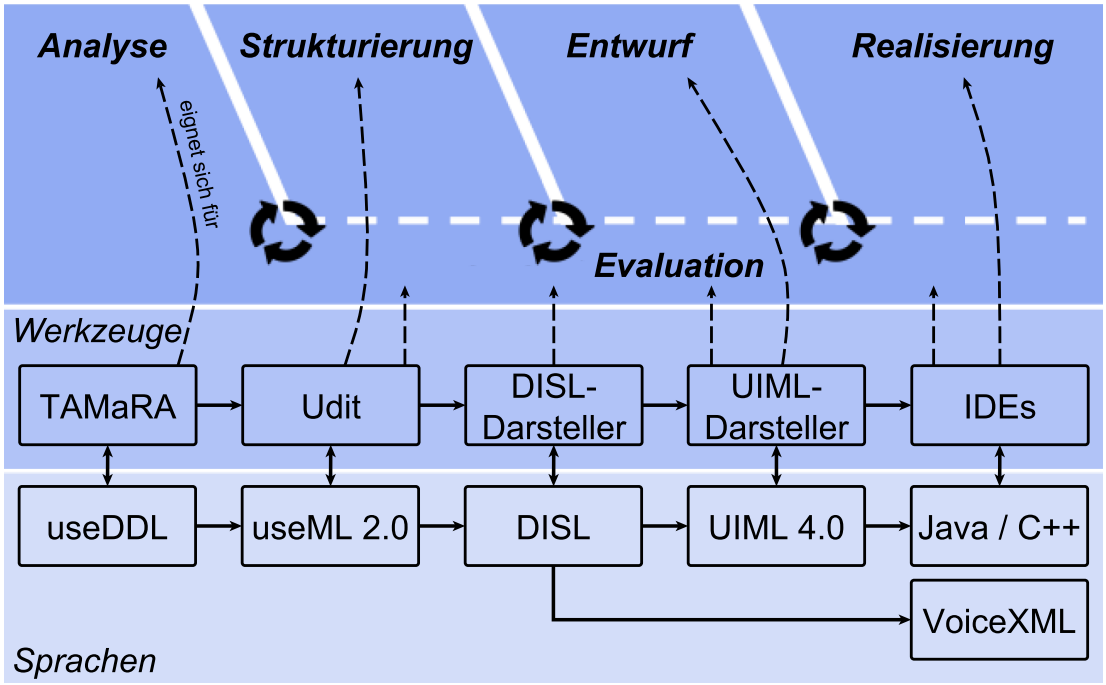


Abbildung 5.8: Der Useware Engineering Prozess mit modellbasierter Werkzeugkette nach Meixner (2010, S. 2)

Der Prozess gliedert sich in die vier überlappenden Schritte (1) Analyse, (2) Strukturierung, (3) Entwurf und (4) Realisierung, welche frühzeitig und permanent evaluiert werden. Außerdem verfolgt das Useware Engineering einen modellgetriebenen Ansatz konform zum CRF. Lediglich die Auswahl der Formalismen und Werkzeuge zur

Modellbildung unterscheiden sich zu den bisher vorgestellten Ansätzen. So wird die Useware Markup Language zur detaillierten Repräsentation von Aufgaben und deren temporale Abhängigkeiten eingesetzt. Auf dieser Basis wird wie in EMODE eine DISL-Repräsentation des abstrakten UIs abgeleitet, welches anschließend in der User Interface Markup Language (UIML) als konkretes UI transformiert wird. Hierzu stehen proprietäre Werkzeuge wie der useML-Editor Udit oder ein DISL2UIML-Transformator zur Verfügung. Die konkrete, rein grafische Benutzerschnittstelle wird mit Hilfe eines UIML-Renderers plattformspezifisch visualisiert. Eine Komposition von Benutzerschnittstellen ist nicht möglich.

5.2.3 DomainEditor

Gandhe u. a. (2009) beschreiben mit dem DomainEditor ein integriertes Autorenwerkzeug zum schnellen Erstellen von dialogbasierten taktischen Vernehmungssimulationen. Diese Simulationen dienen Militärangehörigen zum Erlernen und Trainieren von Vernehmungstechniken zur Informationsgewinnung. Dabei interagiert ein menschlicher Dialogteilnehmer mit einem oder mehreren virtuellen Charakteren, die sich in der Regel erst durch Anwendung einer Vernehmungstaktik kooperativ verhalten. Dazu zählen neben normalen Fragen zur Informationsgewinnung und zum Grounding (Kapitel 4.2.4, S. 53) durchaus auch das Komplimentieren, Beschwichtigen, Belohnen, Beleidigen und Bedrohen der virtuellen Charaktere, um an die letzten wichtigen Details zu gelangen. Das Verhalten und die Reaktionen der virtuellen Charaktere kann in einem leichtgewichtigen Information State-basieren Dialogmanagementansatz auf Basis von SCXML-basierten Zustandsgraphen kulturspezifisch modelliert werden. Der DomainEditor, dessen Benutzerschnittstelle in Abbildung 5.9 gezeigt wird, richtet sich ausdrücklich an Autoren ohne fachlichen Hintergrund im Bereich Dialogsysteme und -management.

In einem Top-Down-Ansatz erstellen Autoren initial ein Modell der Domäne, welches auf Propositionen *<Objekt, Attribut, Wert>* heruntergebrochen wird. Jeder virtuelle und menschliche Charakter hat Kenntnis über eine bestimmte Menge an Propositionen. Dies bildet die Ausgangsbasis zur Generierung von Dialogakten. Ein entsprechender Algorithmus wird von Gandhe u. a. (2009, S. 4) skizziert. Pro Faktum werden Dialogakte mit unterschiedlichen kommunikativen Funktionen erzeugt, die es etwa einem unwissenden Charakter erlauben, danach zu fragen und einem wissenden entsprechend wahrheitsgemäß zu antworten oder zu lügen. Auf logischer Ebene des AUI werden so die **Konsistenz** und die **Vollständigkeit** der vom System zu verarbeitenden Dialogakte sichergestellt. Konsistenz bedeutet in diesem Zusammenhang, dass nur Dialogakte generiert werden, die vom Dialogsystem verarbeitet werden können. Der Begriff Vollständigkeit meint, dass alle für die Domäne relevanten Dialogakte erzeugt werden. Für jeden Dialogakt muss auf Ebene des CUI mindestens eine Verbalisierung im Sprachmodell formuliert werden, um die kommunikative Funktion und den semantischen Inhalt des Dialogaktes transportieren zu können. Dabei wird unterschieden zwischen Dialogakten des menschlichen Nutzers und Dialogakten der virtuellen Charaktere.

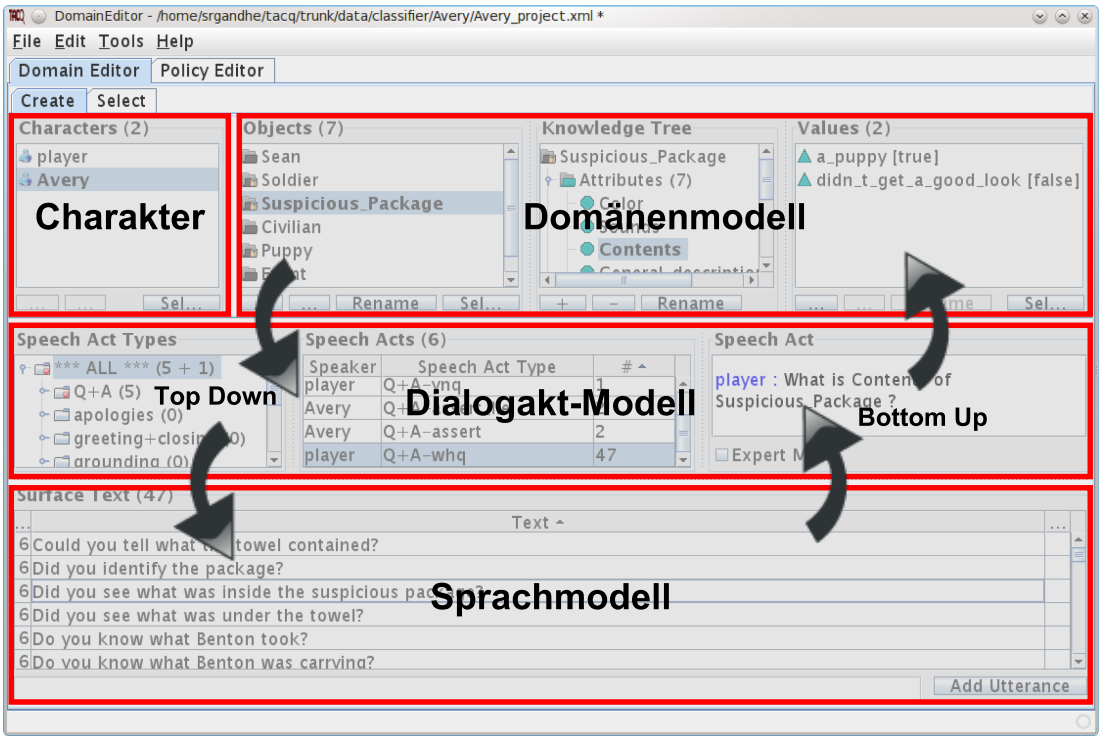


Abbildung 5.9: Die Benutzerschnittstelle des DomainEditors nach Gandhe u. a. (2011)

Verbalisierungen für erstere dienen zum Trainieren eines Modells zum Sprachverstehen. Mehrfache Verbalisierungsoptionen eines Charakter-Dialogaktes erzeugen eine Varianz in der Sprachausgabe des entsprechenden Charakters. Die so im Top-Down-Verfahren erstellte Simulation dient im Rahmen von Benutzerstudien zum Sammeln von bislang unbedachten Nutzeräußerungen, die dann durch Dialogaktannotation in einem Bottom-Up-Ansatz zur Verfeinerung der Simulation dienen. Dies kann auch zu Erweiterungen am Domänenmodell führen.

Gandhe u. a. (2011) liefern einen Anhaltspunkt, wie lange die Entwicklung einer Simulation durch Nicht-Experten dauern kann und wie groß die entsprechenden Modelle werden können. In Abhängigkeit von der Größe des Domänenmodells liegt die Entwicklungszeit typischerweise bei vier Monaten. Im Falle der Simulation von Victor und Amber (Lane u. a. 2010) wurden 480 Dialogakte und 4109 Äußerungen für den menschlichen Benutzer sowie 339 Dialogakte und 332 Äußerungen[7] für die virtuellen Charaktere erstellt. Die sehr große Anzahl der Benutzeräußerungen umfasst alle möglichen grammatikalischen und synonymen Varianten und dient zum Trainieren eines Sprach-

[7] Warum die Anzahl von Ambers Äußerungen kleiner ist als die Anzahl der Dialogakte, ist nicht ersichtlich. Evtl. konnten Synergieeffekte durch den Charakter Viktor ausgenutzt werden, oder es handelt sich hier lediglich um ein Versehen.

erkennungsmodells. Ein grammatikbasierter Ansatz, wie von der SiAM-Dialogplattform verfolgt, erreicht in der Regel eine ähnliche Varianz bei einer viel kleineren Menge an hierarchisch strukturierten Äußerungen. Während die Erkennerrate also prinzipiell abhängig vom verwendeten Spracherkennungssystem ist (Yao u. a. 2010; Morbini u. a. 2013), geben Artstein u. a. (2011) die automatisch erzeugte Abdeckung der Simulation auf Benutzerseite anhand von Benutzerstudien mit 72-76% an. Dieser Wert kann durch den Bottom-Up-Ansatz um ca. 12% verbessert werden. Das Problem liegt hier aber prinzipiell und verkürzt dargestellt darin, dass dem Benutzer eine Domänenoffenheit suggeriert wird. Folglich kann eine 100%-Abdeckung aller Benutzeräußerungen nicht erreicht werden.

Allerdings ist der vorgestellte Ansatz durchaus für diese Arbeit interessant. Im Gegensatz zu einer vermeintlich offenen und potentiell unkooperativen Frage-Antwort-Situation findet sich der Benutzer bei der (natürlichsprachlichen) Interaktion mit Smart Services im Rahmen eng umgrenzter Aufgaben vielmehr in einer kooperativen Situation wieder, in der Dialogmanagementverfahren zum Schablonenfüllen (Kapitel 4.3.4, S. 64) zum Einsatz kommen können. Dies kann die Komplexität auf der einen Seite merklich reduzieren. Auf der anderen Seite werden neue Anforderungen gestellt, die es zu berücksichtigen gilt. Hierzu zählt etwa die Möglichkeit, mehrere Slots einer Schablone gleichzeitig zu füllen. Ein generierter Dialogakt im DomainEditor bezieht sich lediglich auf den Wert eines einzigen atomaren Objektattributs. Die vorliegende Arbeit folgt dem Ansatz von Zanten (1998), der einen hierarchischen Ansatz der Dialogakterzeugung beschreibt, um in einem kooperativen Frage-Antwort-Szenario unspezifische Fragen sowie Über- und Unterbeantwortung adaptiv abzubilden. Mit einer entsprechend hierarchischen Domänenmodellierung könnte dieser Ansatz auch im DomainEditor angewandt werden.

Maass u. a. (2011) beschreiben im Kontext von Verkaufsgesprächen eine methodisches Vorgehen zur dynamischen Erzeugung von Frage-Antwort-Dialogen samt prototypischer Umsetzung in einem mobilen Verkaufsassistenzdienstes. Verkaufsgespräche sind nicht immer kooperativ und oft geprägt durch gegensätzliche Interessen und Intentionen, auf deren Grundlage verkaufsrelevante Frage-Antwort-Muster identifiziert wurden. Übertragen auf den DomainEditor kann das beschriebene Vorgehen dazu angewandt werden, trainingsrelevante Frage-Antwort-Muster in Vernehmungssituationen abzuleiten, die die Erstellung von Vernehmungssimulationen durch Wiederverwendung der Muster beschleunigen.

5.2.4 Universal Remote Console

Die Universal Remote Console (URC) ist eine Kombination von Hard- und Software, welche es Benutzern erlaubt, mit Haushaltselektronik und digitalen Diensten personalisiert und barrierefrei zu interagieren. Eine Benutzerschnittstelle kann für eine Person bzw. für eine Personengruppe adaptiert werden und steht fortan für alle Instanzen einer Geräteklassen als universelle Bedienschnittstelle zur Verfügung. So kann etwa eine

URC-basierte TV-Fernbedienung herstellerübergreifend TV-Funktionen steuern. Gerade älteren Menschen und Menschen mit kognitiven Einschränkungen wird so ein leichter Zugang zu Haushaltselektronik ermöglicht, z. B. bei einem Gerätewechsel. Ein breiter gefasster Anwendungsbereich liegt allgemein auf der Steuerung von vernetzten Geräten im Rahmen der Hausautomation. URC ist durch die ISO/IEC (2014) standardisiert, Aktivitäten im Bereich URC werden durch die openURC Alliance (Alexandersson u. a. 2011) gesteuert.

Die URC-Plattformarchitektur ist in Abbildung 5.10 dargestellt. Diese verknüpft netzwerkfähige Haushaltselektronik und digitale Dienste (in der URC-Terminologie als Target bezeichnet) mit nahezu beliebigen Endgeräten zur Steuerung und Zustandsüberwachung der Targets.

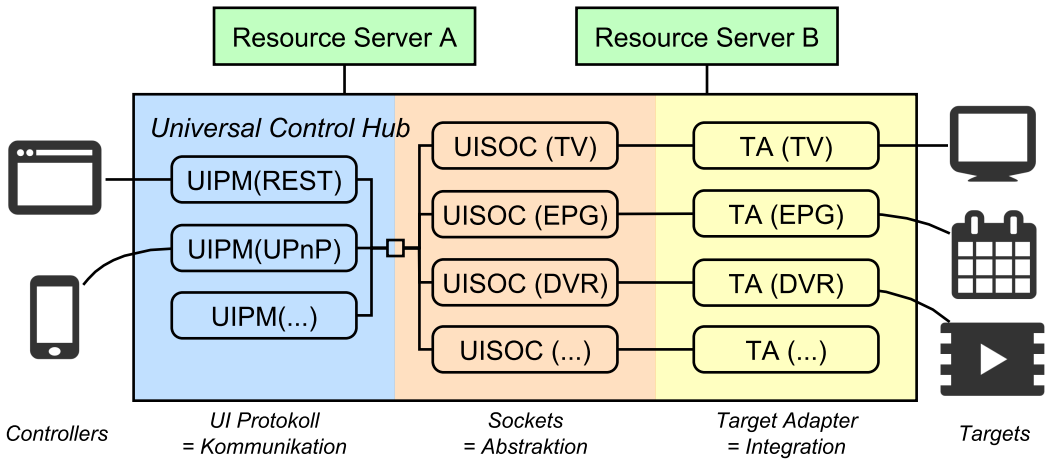


Abbildung 5.10: Die URC-Plattformarchitekur nach Frey (2015, S. 104)

Dazwischen befindet sich der Universal Control Hub (Zimmermann und Vanderheiden 2007) als zentrale Middleware-Komponente. Der Aufbau der Komponente erfolgt in drei Schichten. Die Target-Adapter-Schicht integriert auf technischer Ebene ein Target in die Middleware. Die Socket-Schicht stellt eine Abstraktionsschicht dar, welche von den technischen Target-Adaptern abstrahiert. Dazu werden Zustände, Funktionen und Ereignisse eines Targets in Form eines User Interface Socket abstrakt beschrieben. Ein Endgerät rendert eine konkrete Bedienschnittstelle für eine Geräteklasse. Dazu kommuniziert das Endgerät über ein User Interface Protocol Module der UI Protokollschicht mit den entsprechenden Target-Adaptern auf Basis der abstrakten Schnittstellenbeschreibungen. Über Ressourcenserver können alle beschriebenen Artefakte zur Wiederverwendung und (manuellen) Komposition im Sinne von Pluggable User Interfaces (Zimmermann u. a. 2011) bereitgestellt werden.

Frey (2015) beschreibt eine modulare Implementierung der Middleware auf Basis von OSGi. Diese wird im Rahmen seiner agentenbasierten Integrationsplattform zur Bereitstellung von Smart Services in intelligenten Umgebungen eingesetzt (Kapitel 3.2, S. 30).

Die Modellierung von Smart Services als auch von intelligenten Umgebungen erfolgt mit Hilfe von EMF. Zur Steigerung der Benutzerakzeptanz wird von Britz u. a. (2016) eine sichere Implementierung des Universal Control Hub gemäß der Vertrauenswürdigkeitsstufe *methodisch entwickelt, getestet und durchgesehen* (EAL-4) der Common Criteria Richtlinien (ISO/IEC 15408) angestrebt.

URC kann als ein zu dieser Arbeit komplementärer Ansatz aufgefasst werden, der sich primär mit der Bereitstellung von Haushaltselektronik und digitalen Diensten sowie dem einheitlichen Zugriff darauf über eine personalisierte und barrierefreie Bedienschnittstelle auseinandersetzt. Die modellgetriebene Entwicklung einer Bedienschnittstelle für Endgeräteklassen lag bislang nicht im Fokus. Hierzu können aber Konzepte und Methoden der vorliegenden Arbeit eingesetzt werden. Dennoch wurde der URC-Ansatz bereits mit multimodalen Dialogsystemen gekoppelt (Neßelrath und Porta 2011).

5.2.5 Intelligenter Web-Roboter

Bierwas u. a. (2014) übertragen den URC-Ansatz zur logischen Abstraktion isolierter Funktionalitäten vernetzter Geräte (-klassen), um diese über eine einheitliche Bedienschnittstelle nutzbar zu machen, auf die Prozessebene. Hierzu werden eine Menge von transaktionalen Webseiten einer Klasse (z. B. Reisebuchungsportale oder Bankwebseiten) untersucht, anhand derer abstrakte Referenzprozesse zur automatischen Durchführung einer Transaktion im Sinne einer robotergesteuerten Prozessautomatisierung erstellt werden. Abbildung 5.11 verdeutlicht den Zusammenhang relevanter Modelle.

Die einzelnen Prozessschritte abstrahieren von einer konkreten Webseite und nutzen Klassifikatoren zur automatischen Identifikation semantischer Konzepte, die aus (Benutzer-) Interaktionssicht für den Prozess relevant sind. Um dies zu ermöglichen, müssen innerhalb eines Prozessschritts relevante Konzepte eines Aufgabenmodells, eines Navigationsmodells sowie eines GUI-Modells miteinander in Beziehung gesetzt werden. So kann ein und derselbe Prozess auf unterschiedlichen Webseiten einer Klasse unverändert operieren. Ein Benutzer tritt nun nicht mehr direkt mit der transaktionalen Webseite in Kontakt, sondern nutzt eine einheitliche Bedienschnittstelle zur Durchführung des Referenzprozesses. Diese Bedienschnittstelle kann auf Grundlage des Prozesses mit Hilfe von KomBInoS erstellt werden.

5.3 Komposition von Benutzerschnittstellen für Dienste

Der Fokus dieses Unterkapitels liegt auf Benutzerschnittstellen für Dienste sowie deren Komposition. Dienste sind prinzipiell zu Mehrwertdiensten kombinierbar. Während reine Back-End-Dienste, z. B. in Form eines übergeordneten WS-BPEL-Prozesses (Kapitel 3.4.3.2, S. 42) miteinander orchestriert werden können, ist dieses Vorgehen bei eventuell beteiligten und benötigten Benutzerschnittstellen so nicht möglich. Hier kommen Mashup-Technologien (Kapitel 3.4.1.4, S. 38) zum Einsatz, die eine manuelle Komposition von Diensten und Benutzerschnittstellen ermöglichen.

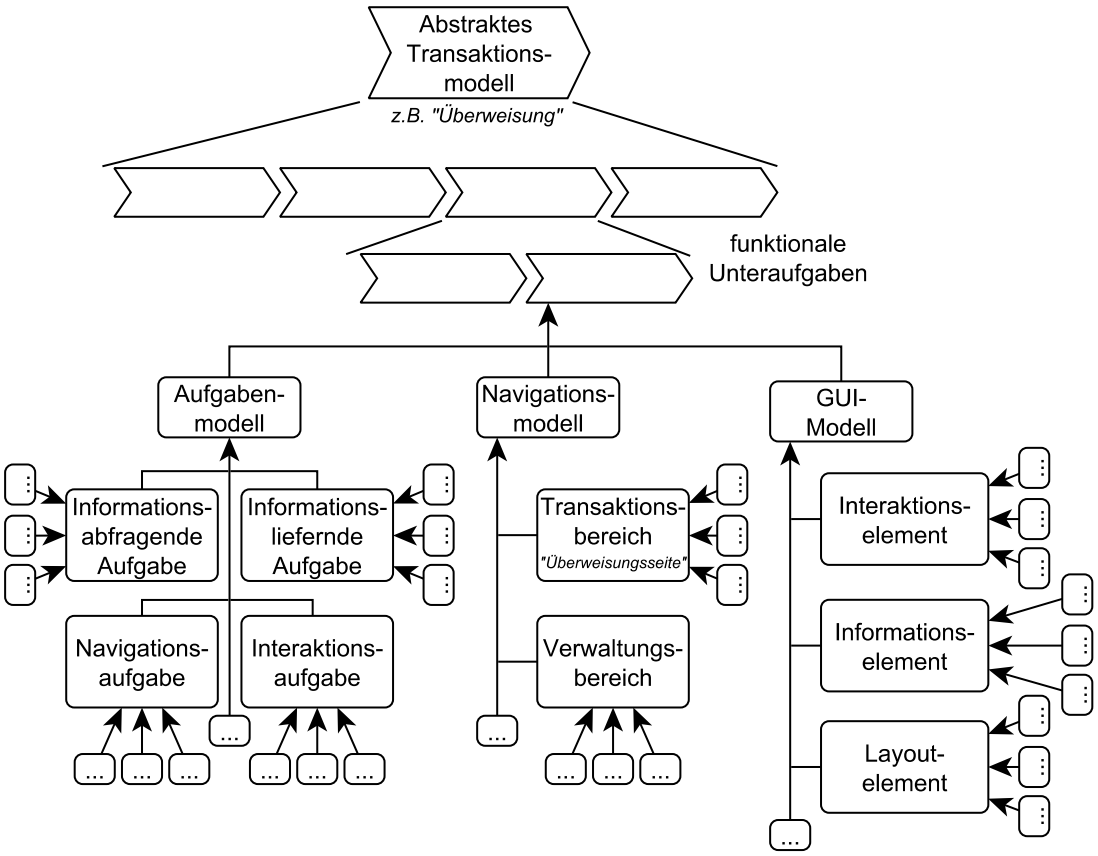


Abbildung 5.11: Relevante Modelle des intelligenten Web-Roboters nach Bierwas u. a. (2014, S. 58)

Das Konzept der Benutzerschnittstellenkomposition wird häufig im Zusammenhang mit Anwendungsintegration genannt. Abbildung 5.12 illustriert die unterschiedlichen Ebenen der Anwendungsintegration.

In einer klassischen Drei-Schichten-Architektur, die man auch in Diensten wiederfindet, kann eine Anwendungsintegration nach Daniel u. a. (2007, S. 61) auf Datenebene (Abbildung 5.12, a), auf Ebene der Geschäftslogik (Abbildung 5.12, b) oder auf Ebene der Benutzerschnittstelle (Abbildung 5.12, c) erfolgen. Dies wird durch eine zusätzlich eingefügte Integrationsschicht erreicht, oberhalb derer die eigentliche Anwendung auf Basis von zugänglichen Programmierschnittstellen meist komplett neu implementiert wird. Im Zuge einer Dienstekomposition ist eine ausschließliche Integration auf Datenebene (Abbildung 5.12, a) nicht anwendbar, da eine Dienstekomposition ja gerade darauf abzielt, Geschäftslogik einzelner atomarer Dienste über eine wohldefinierte API wiederzuverwenden (Abbildung 5.12, b).

Eine Integration auf UI-Ebene (Abbildung 5.12, c) hält Daniel u. a. (2007, S. 61)

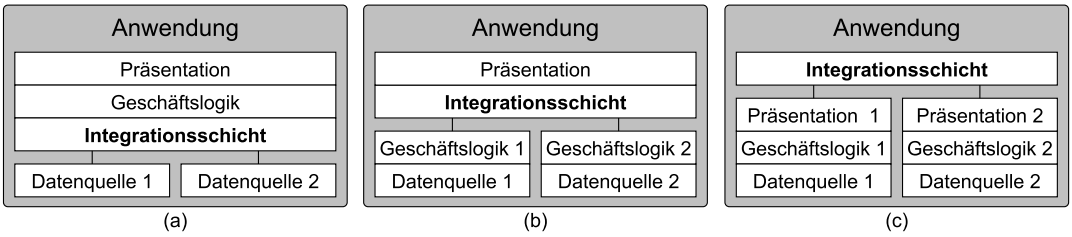


Abbildung 5.12: Anwendungsintegration auf verschiedenen Ebenen (Daniel u. a. 2007, S. 61)

insbesondere dann für sinnvoll, wenn diese auf den darunterliegenden Ebenen, z. B. aufgrund fehlender APIs, nicht möglich oder eine Neuentwicklung der Benutzerschnittstelle, z. B. aufgrund ihrer Komplexität, zu teuer ist. Bei multimodalen Dialogschnittstellen trifft letzteres zu. Die Entwicklung benötigt im Vergleich zu klassischen Benutzerschnittstellen mehr Zeit, mehr Spezialwissen und allgemein mehr Ressourcen.

AbuJarour u. a. (2009, S. 7) analysieren die grundsätzlichen Herangehensweisen zur UI-Aggregation[8]. Abbildung 5.13 zeigt den Zusammenhang und die Abhängigkeiten der dabei beteiligten Artefakte.

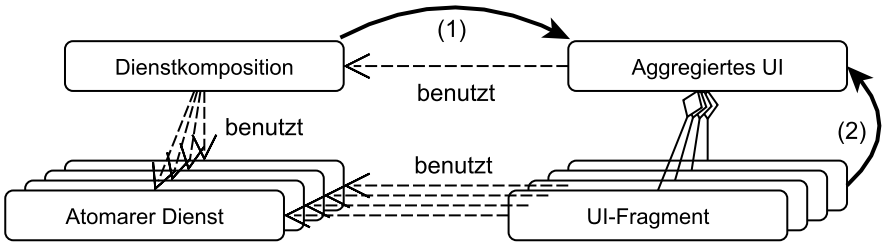


Abbildung 5.13: Beteiligte Artefakte und deren Zusammenhang bei der UI-Aggregation nach AbuJarour u. a. (2009, S. 7)

Ein komponierter Dienst ruft im Rahmen seiner internen Kompositionskette mehrere atomare Dienste über deren öffentliche API auf. Ebenso existiert zu jedem der atomaren Dienste je ein UI-Fragment, welches mit dem entsprechenden atomaren Dienst interagieren kann. Folglich können in einer aggregierten Benutzerschnittstelle sowohl der komponierte Dienst als auch die Gruppe der beteiligten UI-Fragmente herangezogen werden, um die atomaren Dienste zu konsumieren. Dies eröffnet insgesamt drei mögliche Wege zur Erstellung einer aggregierten Benutzerschnittstelle.

(1) **Neuimplementierung der UI auf Basis des komponierten Dienstes.** Obwohl von AbuJarour u. a. (2009) nicht explizit und hier nur der Vollständigkeit

[8] In dieser Arbeit werden die Begriffe UI-Aggregation und UI-Integration synonym verwendet, auch wenn streng genommen ein Unterschied dahingehend besteht, dass Bestandteile einer Aggregation auch ohne ihr Aggregat existieren können, was die Wiederverwendbarkeit steigert.

halber aufgeführt, kann, wie in Abbildung 5.12 (b) gezeigt, eine Dienstekomposition allein auf logischer Ebene unter Verwendung bereitgestellter APIs erfolgen. Der so neu erschaffene, komponierte Dienst stellt die Integrationsschicht dar sowie eine eigene API zur Verfügung, auf deren Basis eine die atomaren Dienste aggregierende Benutzerschnittstelle von Grund auf neu entwickelt wird. In Abbildung 5.13 wird dieser Weg durch den Pfeil (1) und in Abbildung 5.12 durch den Fall (b) dargestellt.

(2) **Dienstekomposition ausschließlich auf UI-Ebene.** Alternativ kann eine Komposition auch rein auf UI-Ebene durchgeführt werden. Dies stellt auf der einen Seite eine relativ leichtgewichtige Integration der beteiligten atomaren Dienste dar. Auf der anderen Seite bedeutet diese Form der Komposition aber auch, dass dennoch notwendige Rechenaufwände für Datentransformation und Dienstaufrufe direkt in der Benutzerschnittstelle stattfinden, welche dafür in der Regel nicht ausgelegt ist. Da die Dienstekomposition lediglich durch die Benutzerschnittstelle zusammengehalten wird, existiert kein wiederverwendbarer komponierter Dienst, der wiederum von anderen Diensten über eine API konsumiert und somit selbst wieder integriert bzw. zu einem weiteren Mehrwertdienst komponiert werden kann. In Abbildung 5.13 wird dieses Vorgehen durch den Pfeil (2) dargestellt. Weiterhin entspricht dieses Vorgehen dem Fall (c) in Abbildung 5.12.

(3) **Kombinierte Dienstekomposition mit UI-Aggregation.** In ihrem PoSR-System (Kapitel 5.3.3, S. 97) favorisieren AbuJarour u. a. (2009, S. 7) letztlich eine kombinierte Dienstekomposition sowohl auf logischer als auch auf UI-Ebene zur Erzeugung einer aggregierten Benutzerschnittstelle. Dabei wird Kompositionswissen der logischen Ebene auf UI-Ebene zur semiautomatischen Erzeugung einer aggregierten Benutzerschnittstelle ausgenutzt. So entsteht ein vom UI unabhängig nutzbarer (Back-End-) Dienst. Statt der APIs der atomaren Dienste, muss in der aggregierten Benutzerschnittstelle stets die API des komponierten Dienstes angesprochen werden.

Im Folgenden wird zunächst ein konzeptioneller Designraum zur UI-Komposition vorgestellt. Anschließend werden relevante Arbeiten und konkrete Systeme bzgl. Komposition von Benutzerschnittstellen für Dienste anhand der gerade beschriebenen Unterscheidung behandelt. Auch wenn für diese Arbeit Fall (3), also die kombinierte Dienstekomposition mit UI-Aggregation, im Vordergrund steht, so kann man anhand verwandter Arbeiten, die den Fall (2) betreffen, einige wichtige Konzepte lernen und entsprechende Modelle ableiten.

5.3.1 Designraum zur UI-Komposition

Paternò u. a. (2011a) beschreiben einen Design- bzw. Problemraum für die Komposition von dienstbasierten Benutzerschnittstellen und adressieren dabei vertieft die Fälle (2) und (3) der obigen Aufzählung. Anhand des Problemraums, der in Abbildung 5.14 dargestellt ist, wird erläutert, welche Aspekte einer Benutzerschnittstelle bei einer Kom-

position berücksichtigt werden können und wie dies geschehen mag.

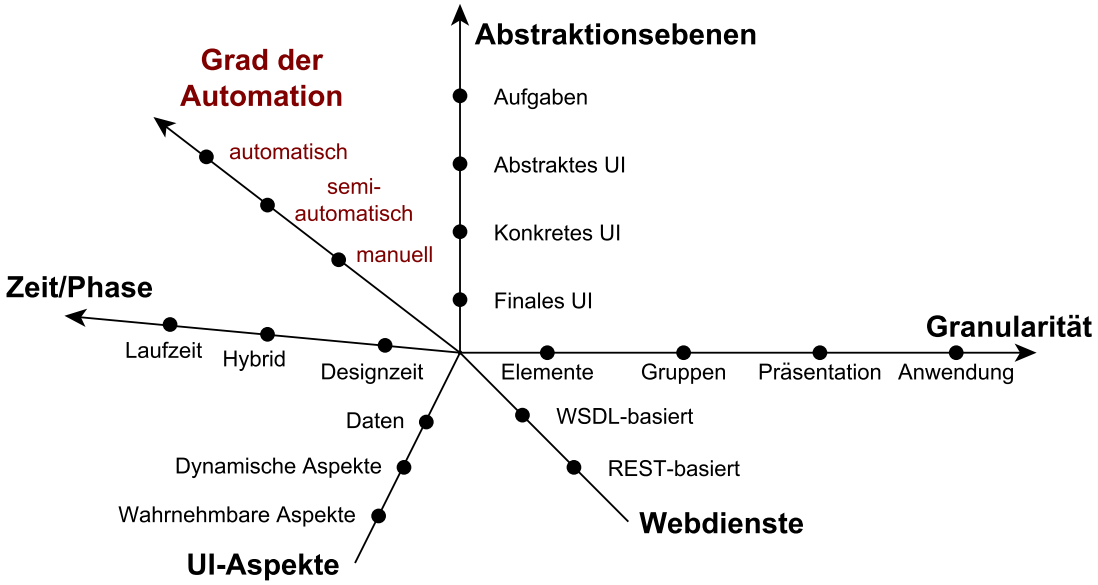


Abbildung 5.14: Unterschiedliche Dimensionen im Designraum für UI-Komposition nach Paternò u. a. (2011a, S. 45)

Der Problemraum besteht ursprünglich aus fünf Dimensionen, die unterschiedliche Aspekte der UI-Komposition berücksichtigen. Die Autoren merken hierzu an, dass lediglich die drei Dimensionen *Abstraktionsebene*, *Granularität* und *Zeit/Phase* eine Ordnungsrelation auf ihrer entsprechenden Achse besitzen, was durch einen Pfeil am Ende der Achse gekennzeichnet ist. Diese Relation gibt an, ab bzw. bis wann die Komposition in der jeweiligen Dimension erfolgt.

- **Abstraktionsebenen.** Das CRF unterteilt eine Benutzerschnittstelle in die Abstraktionsebenen Konzepte und Aufgabenmodell, abstraktes UI, konkretes UI sowie finales UI. Diese Gliederung findet sich auch in dieser Dimension wieder und drückt aus, dass eine Komposition auf all diesen Ebenen stattfinden kann.
- **Granularität.** Diese Dimension unterscheidet die Granularität der Komposition auf Präsentationsebene. So können in einer Komposition einzelne (grafische) Elemente, Elementgruppen, vollständige Präsentationen oder gar vollständige Anwendungen betrachtet werden. Eine Präsentation bezeichnet hierbei alle für einen Benutzer zu einem bestimmten Zeitpunkt wahrnehmbaren Bestandteile einer Benutzerschnittstelle einer Anwendung. Die Komposition vollständiger Anwendungen kann beispielsweise durch Mashup-Technologien wie Portalserver und Portlets (Polgar 2012) erfolgen, bei der jedoch keine tiefere inhaltliche UI-Komposition

stattfindet.

- **UI-Aspekte.** Die Unterscheidung, die in dieser Dimension getroffen wird, ist im Groben vergleichbar mit Garretts (2010) Ebenenstruktur (Kapitel 5.2.1, S. 72). Paternò u. a. (2011a) berücksichtigen, (1) ob eine Komposition hauptsächlich auf Ebene der Daten, die vom UI manipuliert werden, stattfindet, (2) ob dynamische Aspekte bezogen auf die Reihenfolge der vom Benutzer durchzuführenden Aktionen betroffen sind oder (3) ob die Komposition auf wahrnehmbaren UI-Aspekten beruht. Während bei Garrett (2010) die entsprechenden Ebenen Struktur, Skelett und Oberfläche im Entwicklungsprozess aufeinander aufbauen, sehen Paternò u. a. (2011a) diese Aspekte im Rahmen der UI-Komposition als orthogonal an.
- **Zeit/Phase.** Diese Dimension drückt aus, wann eine Komposition stattfindet. Sie unterscheidet eine statische Komposition zur Entwurfszeit, eine dynamische Komposition zur Laufzeit oder eine Mischform, in der Teile sowohl statisch als auch dynamisch komponiert werden.
- **Webdienst.** Diese Dimension liefert eine Aufzählung von verschiedenen Webservice-Technologien (Kapitel 3.4.1, S. 36), die bei der UI-Komposition berücksichtigt werden sollten. Obwohl Paternò u. a. (2011a) hier lediglich zwei Technologien nennen, ist diese Aufzählung bei Weitem nicht erschöpfend. Ein Kompositionsansatz ist in Bezug auf diese Dimension umso umfänglicher, je größer die Anzahl der berücksichtigten Webservice-Technologien ist. Neben der Problematik der Datenmediation auf syntaktischer oder semantischer Ebene, ist hier die Heterogenität von Webdiensten auf technischer Ebene ausschlaggebend.

Zusätzlich zu den gezeigten fünf Dimensionen von Paternò u. a. (2011a) ist für diese Arbeit eine weitere Dimension von Relevanz: der Grad der Automation einer Komposition. Diese Dimension ist in Abbildung 5.14 in rot gekennzeichnet.

- **Grad der Automation.** Diese Dimension drückt aus, wie eine Komposition durchgeführt wird. Dies kann rein manuell, vollautomatisch oder in einer Mischform semiautomatisch geschehen. Diese Dimension ist orthogonal zur Dimension Zeit/Phase, da eine Komposition zur Entwurfszeit durchaus vollautomatisch und eine Komposition zur Laufzeit mit Auswirkungen auf das Nutzererlebnis ebenso manuell durchgeführt werden kann.

5.3.2 Dienstekomposition auf UI-Ebene: ServFace, CRUISe und mashArt

Pietschmann u. a. (2010a) vergleichen ihre eigenen Ansätze zur Anwendungs- bzw. Dienstekomposition auf der UI-Ebene unter Zuhilfenahme eines eigens erstellten Evaluationsrahmenwerks. Dabei untersuchen sie den aktuellen Lösungsraum und decken noch offene Herausforderungen auf, um das generelle Vorgehen der Dienstekomposition auf UI-Ebene zu etablieren. Das Evaluationsrahmenwerk umfasst aus Sicht der Autoren wichtige funktionale Aspekte bzgl. der Dienstekomposition bzw. -integration auf UI-Ebene. Die Evaluation und der Vergleich eines Kompositionsansatzes erfolgen da-

mit in erster Linie qualitativ und nicht quantitativ. Ohne im Detail auf die einzelnen Evaluationskriterien einzugehen, lässt sich daran ein prinzipielles Vorgehen zur Dienstekomposition auf UI-Ebene erkennen. Ein **Komponentenmodell** abstrahiert in einem Bottom-Up-Ansatz von heterogenen funktionalen Komponenten, welche die atomaren Bestandteile einer (komponierten) Anwendung darstellen, und ermöglicht so deren Wiederverwendung im Rahmen eines **Kompositionsmodells**. Zur Erstellung der Modelle stehen in der Regel geeignete **Werkzeuge** zur Verfügung. Eine **Laufzeitumgebung** sorgt für die tatsächliche Durchführung der zuvor spezifizierten Komposition. Vor diesem Hintergrund werden nun die Arbeiten ServFace, CRUISe und mashArt vorgestellt.

ServFace

Nestler u. a. (2009, 2010) beschreiben mit dem ServFace Builder einen grafischen Editor, der es einem Kompositionsersteller ermöglicht, Webdienste auf Ebene der Benutzerschnittstelle über mehrere Ansichten hinweg zu komponieren. Dazu sind neben der rein technischen WSDL-basierten Beschreibung der verwendeten Dienste auch entsprechende UI-Annotationen einzelner Dienstoperationen notwendig, die die Dienstekomposition auf UI-Ebene erleichtern sollen. Solche Annotationen werden von den Dienstentwicklern auf Basis eines Metamodells spezifiziert und stehen dem Kompositionsersteller anschließend zur Verfügung. Die Annotationen modellieren statische UI-Aspekte, Verhalten von UI-Elementen und Abhängigkeiten zwischen anderen Diensten bzw. Dienstoperationen.

Das Metamodell zur Beschreibung einer komponierten Anwendung ist in Form des Composite Application Models (Feldmann u. a. 2009) auszugsweise in Abbildung 5.15 dargestellt.

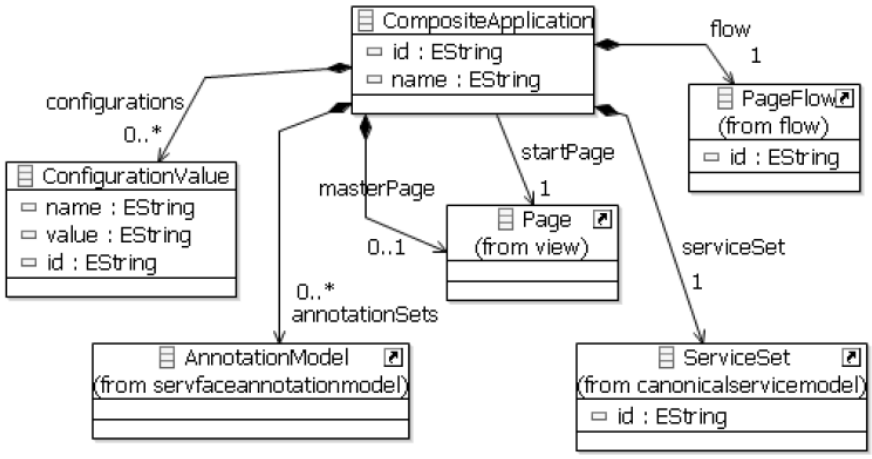


Abbildung 5.15: Das Composite Application Metamodell in ServFace (Feldmann u. a. 2009, S. 25)

In einem ersten Schritt zur Erstellung einer komponierten Anwendung (CompositeApplication) werden zu verknüpfende Dienstoperationen unterschiedlicher annotierter

Dienste ausgewählt (ServiceSet). Anschließend können aus den technischen Dienstbeschreibungen und den UI-Annotationen (AnnotationModel) der ausgewählten Operationen automatisiert UI-Fragmente abgeleitet werden. Diese sind logisch korrekt mit den entsprechenden Ein- und Ausgabeparameter der Dienstoperationen verknüpft. Ebenso werden in diesem Generierungsprozess regelbasierte Usability-Richtlinien berücksichtigt. Schließlich kann der Kompositionsersteller im ServFace Builder UI-Elemente von Aus- und Eingabeparametern unterschiedlicher Operationen miteinander verknüpfen, um so den Datenfluss innerhalb der Anwendung zu spezifizieren. UI-Elemente können zur besseren Strukturierung der Anwendung auf mehrere Ansichten (Page) verteilt werden. Diese Ansichten können im Rahmen des zu spezifizierenden Navigationsflusses entweder in eine einfache lineare Abfolge oder alternativ in einen komplexeren, verzweigenden Graphen eingeordnet werden (PageFlow).

Das Composite Application Model dient als Ausgangspunkt für die automatische Generierung einer Benutzerschnittstelle. Zwei konkrete Zielplattformen stehen zur Auswahl. Es kann entweder eine Webanwendung oder eine vollständige mobile Anwendung für Android erzeugt werden. Das Schreiben von Integrationscode ist nicht erforderlich. Der rein grafische Kompositionsansatz nimmt zugunsten der Endnutzerzentrierung bewusst Nachteile in der Ausdrucksmächtigkeit in Kauf. Nestler u. a. (2011, S. 201) betrachten hauptsächlich Dienstoperationen, die eine Nutzerinteraktion und somit eine Benutzerschnittstelle erfordern. Dienstekompositionen im Hintergrund der Benutzerschnittstelle sind nicht möglich.

CRUISe - Composition of Rich User Interface Services

Pietschmann u. a. (2009) argumentieren, dass zukünftige webbasierte Anwendungen alleine aus Diensten bestehen, die entweder Daten, Geschäftslogik im Sinne von höherwertiger Funktionalität oder Benutzerschnittstellen bereitstellen. Vor diesem Hintergrund wenden sie das Paradigma der Dienstorientiertheit auf die Präsentationsebene von Webanwendungen an, um auf diese Weise eine leichtgewichtige Dienstekomposition zu ermöglichen. Eine Benutzerschnittstelle in CRUISe wird aus generischen und wiederverwendbaren UI-Komponenten zusammengesetzt, die als **Benutzerschnittstellendienste** bezeichnet werden. Ein Benutzerschnittstellendienst ist ein Technologieadapter oder Wrapper mit einer einheitlichen Schnittstelle für konkrete webbasierte UI-Komponenten. Ein Karten-UIS abstrahiert etwa von der konkreten Google Maps API und deren konkreter Umsetzungstechnologie (JavaScript) und stellt die Kartenvisualisierungsfunktionalität konfigurierbar zur Integration über eine einheitlich ansprechbare API zur Verfügung. In CRUISe kann ein Benutzerschnittstellendienst dynamisch und kontextsensitiv ausgewählt, konfiguriert und homogen in eine Benutzerschnittstelle integriert werden. Diese Integration soll zur Laufzeit direkt in der Client-Anwendung des Benutzers erfolgen. Zuvor wird von einem Entwickler manuell eine Kompositionsbeschreibung erstellt. Das Metamodell zur Beschreibung einer Komposition in CRUISe ist in Abbildung 5.16 dargestellt.

Die Kompositionsbeschreibung enthält Referenzen auf genutzte Basiskomponenten,

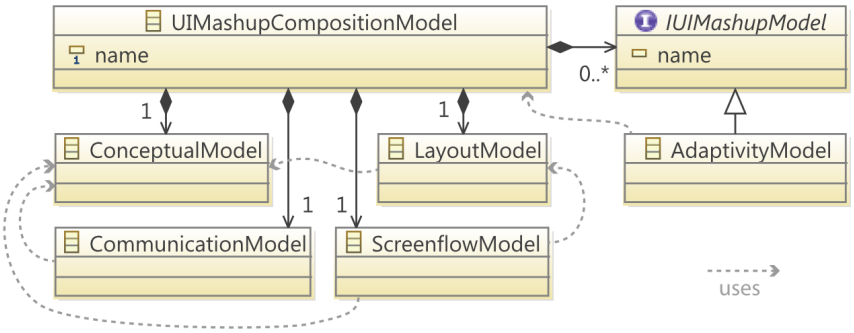


Abbildung 5.16: Bestandteile des Kompositionsmodells in CRUISe (Pietschmann u. a. 2010b, S. 416)

deren kompositionsspezifische Konfiguration, den Kommunikationsfluss zwischen den Komponenten (CommunicationModel), die Verknüpfung einzelner Präsentationen (ScreenflowModel), deren Layout (LayoutModel) sowie Adaptionsverhalten (AdaptivityModel) (Pietschmann u. a. 2010b, S. 415ff.). Aus der deklarativen Beschreibung wird anschließend eine Webanwendung generiert, die mit Platzhaltern für zur Laufzeit injizierte UI-Komponenten versehen ist. Die Nutzung der komponierten Webanwendung erfolgt schließlich in der eigens entwickelten CRUISe-Laufzeitumgebung. Diese führt die notwendige UI-Integration zur Laufzeit auf Basis von bereitgestellten Konfigurationen und dem aktuellen Nutzerkontext durch. Durch die Technologieabstraktion in Form von Benutzerschnittstellendiensten und der deklarativen Kompositionsbeschreibung samt Codegenerierung verfolgt Pietschmann (2012) schließlich die modellgetriebene Entwicklung adaptiver, komponentenbasierter Mashup-Anwendungen.

Eine Komposition in CRUISe wird vom Entwickler manuell ohne semiautomatische Unterstützung modelliert. Zum Feinschliff der Komposition ist JavaScript zwingend erforderlich. CRUISe berücksichtigt keine multimodalen Benutzerschnittstellen, sondern fokussiert auf eher großformatige, rein grafische UIs.

mashArt

mashArt (Daniel u. a. 2009) zielt auf einen universellen Integrationsansatz sowohl auf UI- als auch auf Dienstebene ab. Atomare und wiederverwendbare Komponenten werden in der Mashart Description Language beschrieben, deren Metamodell in Abbildung 5.17 gezeigt wird.

Ein Beschreibung in mashArt spezifiziert relevante Zustände (StateVariable), Ereignisse (Event) zum Kommunizieren von Zustandsänderungen und Operationen (Operation) zur Reaktion auf Zustandsänderungen. Anhand dieser ereignisgestützten Komponentenlogik können mit der ebenfalls durch mashArt spezifizierten Universal Composition Language Kompositionen als ereignis- und datenflussbasierte Verschaltung von mashArt-Komponenten unter Einsatz eines einfachen Publish-Subscribe-Kommunika-

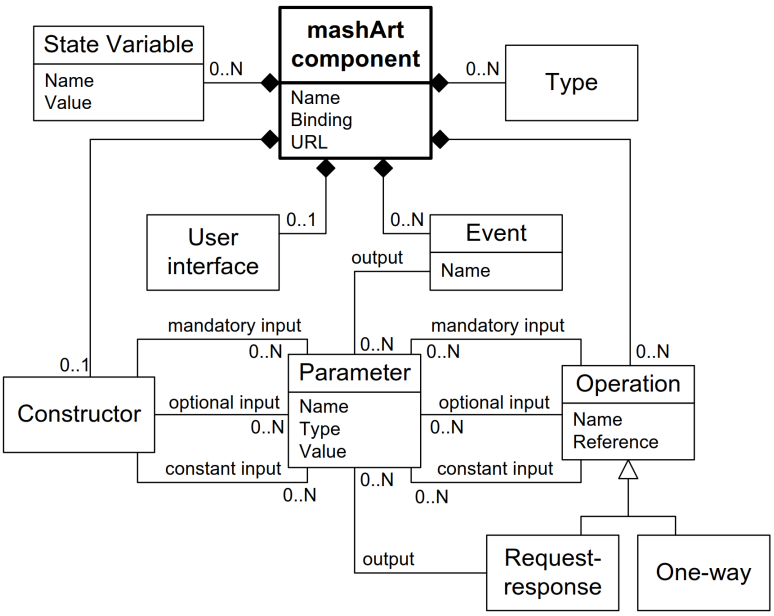


Abbildung 5.17: Das mashArt-Komponentenmodell (Daniel u. a. 2009, S. 434)

tionsmusters beschrieben werden. Dazu stehen die Operationen *Parallele Aufspaltung*, *Konjunktive* und *Disjunktive Zusammenführung* und *Bedingte Ausführung* zur Verfügung. Im Grunde analog zu CRUISe, werden heterogene UI-Komponenten einheitlich beschrieben und können so in einer Komposition leichter wiederverwendet werden. Hierzu steht ein browserbasierter Editor zur Verfügung. Der Aspekt der losen Kopplung durch ereignisgestützte APIs kommt hier jedoch stärker zum Vorschein, wenngleich Ereignisse nicht semantisch repräsentiert werden. Eine serverseitige Laufzeitumgebung kann in mashArt beschriebene Kompositionen in Verbindung mit einem mashArt-Client ausführen.

Diskussion

Zugunsten einer starken Endnutzerzentrierung bei der Kompositionserstellung verzichten ServFace, CRUISe und mashArt auf Abstraktionsebenen im Sinne von Paternòs Designraum (Kapitel 5.3.1, S. 90). Vielmehr gehen alle drei Arbeiten nach dem Reverse-Engineering-Prinzip (Kapitel 2.4.2, S. 20) vor. Zur Verfügung stehende Dienste oder UI-Komponenten werden durch Annotationen oder Technologieadapter auf eine einheitliche semantische Ebene gehoben, von der aus im Anschluss eine Komposition erfolgen kann. Diese Komposition findet direkt in der konkreten Benutzerschnittstelle statt. Weitere Abstraktionsebenen darüber hinaus fehlen. Pietschmann u. a. (2010a, S. 468) merken dazu an, dass es im Allgemeinen schwer sei, die richtige Balance zwischen der Vollständigkeit an Funktionalität und der Einfachheit des Kompositionsansatzes zu finden. Diese Balance müsse jeweils individuell für entsprechende Zielgruppen herausgearbeitet werden. Weiterhin müsse neben der Vereinfachung der Technologie auch die Wiederver-

wendbarkeit nicht nur auf der Ebene der Implementierung unterstützt werden, sondern auch in Bezug auf Domänen- und Kompositionswissen. Kompositionswissen meint hier Wissen über bekannte, zum Teil auch gelernte Kompositionsmuster, welches etwa dazu genutzt werden kann, einen Kompositionsprozess durch automatisierte Vorschläge zu unterstützen. Dieser Aspekt wird im nächsten Abschnitt nochmals genauer diskutiert.

5.3.3 Kombinierte Dienstekomposition mit UI-Aggregation: PoSR

Die Potsdam Services Registry (PoSR) (AbuJarour u. a. 2009) ist ein umfassendes System zum Finden, Erzeugen, Aggregieren und Nutzen von Webdiensten. Dies schließt neben der technischen Ebene auch die Ebene der Benutzerschnittstelle ein. Dazu unterscheidet PoSR zwei ineinandergreifende Nutzungsphasen, die in Abbildung 5.18 dargestellt sind.

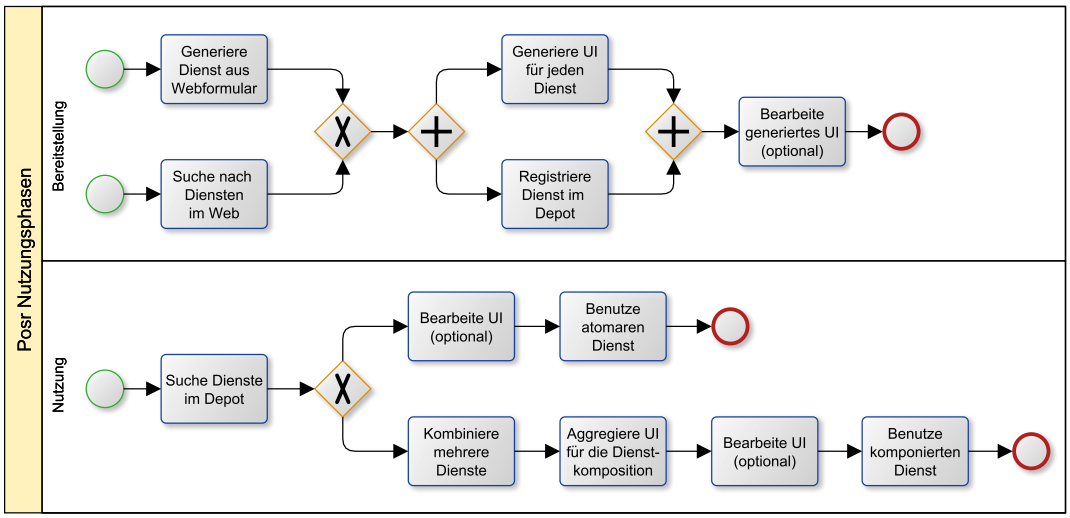


Abbildung 5.18: Verschiedene Nutzungsphasen des PoSR-Systems nach AbuJarour u. a. (2009, S. 3)

In der **Bereitstellungsphase** kann PoSR anhand einer beobachteten Interaktionssequenz eines Benutzers beim Ausfüllen eines Formulars in einer Webanwendung einen elektronisch verfügbaren Dienst mit formaler WSDL-Beschreibung (Kapitel 3.4.1, S. 36) generieren und in seinem eigenen Dienstedepot zur späteren Verwendung hinterlegen. Ein solches Formular kann über mehrere Webseiten verteilt sein, weshalb u. a. neu auftretende von sich wiederholenden Formularfeldern unterschieden werden müssen. Zusätzlich sucht PoSR nach bereits veröffentlichten und formal beschriebenen elektronischen Diensten. Diese werden ebenfalls im Depot hinterlegt und auffindbar gemacht. Für hinterlegte Dienste werden automatisiert UI-Fragmente für die Dateneingabe und die Anzeige von Ergebnissen eines Dienstaufrufs erzeugt. Diese Generierung erfolgt

durch eine Transformation von technischer WSDL-Dienstbeschreibung in eine abstrakte UI-Beschreibung auf Basis des W3C-Standards XForms (Boyer 2009). XForms trennt Daten, Logik und Präsentation voneinander, folgt also dem Model-View-Controller-Architekturmuster. Dies wirkt sich positiv auf die Wiederverwendbarkeit der so beschriebenen UI-Fragmente aus. Automatisch erzeugte UI-Fragmente können in einem nachgelagerten Schritt manuell durch den Benutzer im Webbrowser verfeinert werden, um bei Bedarf die Bedienbarkeit zu verbessern.

Die untere Bahn zeigt die **Nutzungsphase** von zuvor hinterlegten Diensten. Ein Interessent erhält auf Basis seiner Suchanfrage eine Liste von passenden Diensten. Nach einer optionalen Anpassung einer Dienstnutzungsschnittstelle kann der entsprechende Dienst genutzt werden. Er kann alternativ aber auch mehrere Dienste miteinander verknüpfen. Im Anschluss daran muss eine Aggregation der zu den verknüpften Diensten gehörenden UI-Fragmente erfolgen. Dies geschieht unter Zuhilfenahme eines browserbasierten Editors. Die Dienstekomposition wird zusammen mit der aggregierten Benutzerschnittstelle im Depot für etwaige andere Interessenten hinterlegt. Nach einer optionalen Anpassung der aggregierten Benutzerschnittstelle ist die Dienstekomposition schließlich einsatzbereit.

Das Vorgehen zur Dienstekomposition und UI-Aggregation bedarf noch einer genaueren Betrachtung. Die Dienstekomposition auf logischer Ebene wird (ähnlich wie bei mashArt, jedoch nicht auf der UI-Ebene) durch die Anwendung einer Teilmenge der BPMN (Kapitel 3.4.3.2, S. 42) erreicht. Basierend auf den Ergebnissen von zur Muehlen und Recker (2008) schränken AbuJarour u. a. (2009, S. 7) die Sprachvielfalt der BPMN auf die Elemente Aktivität, Sequenzfluss, exklusives (XOR) und inklusives (AND) Gateway, sowie Start- und Ende-Ereignisse ein. Dies führe zu einer deutlichen Komplexitätsreduktion für den Kompositionsersteller. Die Ausdrucksmächtigkeit werde dabei in der Praxis nur gering reduziert. Eine solche Kompositionsbeschreibung kann nach AbuJarour u. a. (2009, S. 7) mit Hilfe semiautomatischer Kompositionstechniken zu einem funktionsfähigen komponierten Dienst weiterentwickelt werden. Eine reine Dienstekomposition steht für sie aber nicht im Fokus. Stattdessen favorisieren sie, wie einleitend erwähnt, eine kombinierte Dienstekomposition sowohl auf logischer Prozessebene als auch auf UI-Ebene zur Erzeugung einer aggregierten Benutzerschnittstelle. Auf logischer Ebene wird die Dienstekomposition, deren Erstellung nicht im Fokus steht, durch einen BPMN-Prozess unter Verwendung lediglich einer fixen Teilmenge der zur Verfügung stehenden BPMN-Sprachelemente beschrieben. Dies bietet den Vorteil eines echten, wiederverwendbaren (Back-End-) Dienstes zum Preis einer zusätzlich benötigten Softwarekomponente zum Ausführen der im Modell repräsentierten Komposition. Unter Zuhilfenahme des im BPMN-Prozess vorhandenen Kompositionswissens der logischen Ebene, werden für die Komposition relevante UI-Fragmente der beteiligten atomaren Dienste automatisiert kombiniert und rekonfiguriert. Die Kombination der UI-Fragmente erfolgt dabei anhand definierter Kompositionsmuster und -regeln basierend auf den in der erlaubten BPMN-Teilmenge enthaltenen Kontrollstrukturen, welche in Tabelle 5.1 aufgelistet sind. Die Rekonfiguration der einzelnen UI-Fragmente bezieht

sich vornehmlich auf den Austausch der in der UI hinterlegten Dienstaufrufe. Statt der API des atomaren Dienstes muss in der aggregierten Benutzerschnittstelle stets die API des komponierten Dienstes angesprochen werden.

Kontrollstruktur	Kompositionsregel
Sequenzfluss	Die UI-Fragmente, der in Reihe verschalteten Dienste werden mithilfe der XForms-Konstrukte *switch* und *case* in der komponierten Benutzerschnittstelle ebenfalls in Reihe geschaltet.
XOR Split	Dem Benutzer wird die Möglichkeit gegeben, manuell zur Laufzeit zu entscheiden, welcher Weg eingeschlagen werden soll. Alternativ kann der Weg zur Entwurfszeit auch im Prozessmodell vorbestimmt werden.
AND Split	Die UI-Fragmente der beteiligten Dienste werden in einer Ansicht aggregiert. Technisch geschieht dies, indem die XForms *switches* der UI Fragmente in einen vom Benutzer kontrollierbaren *switch* verschachtelt werden.

Tabelle 5.1: Kompositionsmuster und -regeln in PoSR

Die automatische Komposition von Benutzerschnittstellen beschränkt sich im PoSR-System auf die einfache Zusammenfassung von XForms-basierten Webformularen. Dabei werden potentiell mehrfach vorhandene, bedeutungsgleiche UI-Elemente für ein und dasselbe Datum unterschiedlicher Webdienste nicht automatisch aggregiert. Diese müssen vom Kompositionsersteller manuell zusammengeführt werden. Unter Umständen sind dabei auch Datentransformationen umzusetzen. Trotzdem stellt die UI-Komposition anhand eines zuvor erstellten logischen Prozessmodells eine wichtige konzeptuelle Erweiterung der bereits vorgestellten Arbeiten dar, welche auch für diese Arbeit relevant ist.

Der Generierungsvorgang bei PoSR arbeitet auf der potentiell unvollständigen Interaktionssequenz sowie auf der syntaktischen Struktur einer formularbasierten Webseite. Im Gegensatz dazu operieren Bierwas u. a. (2014) (Kapitel 5.2.5, S. 87) auf einer abstrakten semantischen Repräsentation sowohl der möglichen Interaktion als auch der darunterliegenden Webseite. Die Bedeutung der Semantik bei der Komposition wird im nächsten Abschnitt weiter thematisiert.

5.3.4 Ontologiebasierte UI-Komposition: UI^{2}Ont

Paulheim und Probst (2013) sehen einen fundamentalen Unterschied zwischen einer domänenspezifischen Sprache (Kapitel 2.4, S. 16) und einer Ontologie in der Hinsicht, dass eine domänenspezifische Sprache vor dem Hintergrund eines bestimmten Zwecks, etwa der einfachen Codegenerierung, erstellt wurde, wohingegen eine Ontologie versucht, die Realität allgemeingültig und nicht an einen konkreten Zweck gebunden möglichst genau abzubilden. Sie analysieren eine Vielzahl an Beschreibungssprachen für

Benutzerschnittstellen, die jeweils unterschiedliche Abstraktionsebenen gemäß Paternòs Designraum (Kapitel 5.3.1, S. 90) abdecken, auf Gemeinsamkeiten und Unterschiede. Dazu zählen unter anderem UsiXML und MARIA, die auch in Kapitel 5.2 (S. 72) erläutert werden. Diese Sprachen werden zu einer umfassenden, allgemeingültigen Ontologie, genannt UI^{2}Ont, für die semantische Beschreibung von Benutzerschnittstellen aggregiert. Auf dieser Basis, so Paulheim und Probst (2013), lassen sich nun durch den Einsatz semantischer Technologien wie etwa Inferenzmaschinen leichter Verfahren für Benutzerschnittstellen mit Erklärungsfunktionalität oder zur Adaption und Integration von Benutzerschnittstellen entwickeln. Auch Konverter von einer konkreten Beschreibungssprache in die Ontologie und zurück seien denkbar, was die Ontologie zu einer allgemeinverständlichen Zwischensprache werden ließe. Bezüglich des letzten Punktes, der UI-Integration, werden anhand der Ontologie zu integrierende UI-Komponenten semantisch beschrieben. Außerdem werden Ereignisse, die zwischen UI-Komponenten ausgetauscht werden können, semantisch annotiert. Zuletzt werden auf Basis der Ontologie Integrationsregeln definiert. UI^{2}Ont muss zusätzlich immer von einer Domänenontologie ergänzt werden, welche relevante Konzeptmodellierungen der Domäne enthält.

Zur Laufzeit steuert ein regelbasierter Ereignisprozessor die Verteilung von Ereignissen an entsprechende UI-Komponenten. Durch die Nutzung des UI^{2}Ont-Rahmenwerks (Paulheim und Probst 2010) können vielfältige konkrete UI-Komponenten unterschiedlichster Implementierungstechnologien auf eine semantische Abstraktionsebene gehoben und miteinander integriert werden, ohne dass Interaktionsmetaphern wie etwa Drag&Drop zwischen einzelnen UI-Komponenten neu modelliert oder implementiert werden müssen (Paulheim und Erdogan 2010). Dieses Rahmenwerk wurde auch im SoKNOS-Projekt (Babitski u. a. 2011) eingesetzt.

Paulheim und Erdogan (2010, S. 307) legen den Fokus des UI^{2}Ont-Rahmenwerks auf die Integration von UI-Komponenten innerhalb von desktopbasierten Anwendung. Multimodale und mobile Benutzerschnittstellen werden nicht betrachtet. Wie auch die zuvor vorgestellten Arbeiten, verfolgt das UI^{2}Ont-Rahmenwerk ein Bottom-Up-Prinzip, sprich existierende UI-Komponenten werden durch Technologieadapter gekapselt oder durch semantische Annotationen beschrieben und so auf eine einheitliche Ebene gehoben.

5.4 Ganzheitliche Methoden zur Dienstentwicklung

In diesem Unterkapitel liegt das Augenmerk auf ganzheitlichen Methoden zur facettierten Dienstentwicklung. Einzelne Facetten können so in entsprechend qualifizierten Teams verteilt und zum Teil parallel entwickelt werden. Dieses Vorgehen bietet sich auch bei der modellgetriebenen Erstellung von multimodalen Dialogschnittstellen an und ist dementsprechend zu bevorzugen. Anhand zweier wesentlicher Beiträge des TEXO-Projekts, die diese Arbeit in ihrer Entstehung maßgeblich beeinflusst haben, wird erläutert, wie (1) unternehmensintern ein facettenreicher Dienst verteilt entwickelt wird und wie (2) dieser Dienst nach außen hin als Dienstleistung beschrieben wird. Die

Dienstleistungsmodellierung liegt zwar nicht im Fokus dieser Arbeit, gehört zu einem vollständigen Bild aber dennoch ins Blickfeld. Die gleichnamige Buchreihe Dienstleistungsmodellierung von Thomas und Nüttgens (2014) beleuchtet dieses Thema vertieft aus wirtschaftsinformatischer Sicht.

5.4.1 Integrated Service Engineering (ISE)

Cardoso u. a. (2009, S. 22) verstehen unter Service Engineering einen Ansatz, der mit Hilfe von Modellen und Methoden alle wesentlichen Aspekte eines Dienstes entlang des Dienstlebenszyklus (Kapitel 3.3, S. 33) unterstützt und abbildet. Das ISE-Rahmenwerk (Kett u. a. 2014) liefert dazu eine konzeptuelle Methodik auf Basis von Zachmans (1987) Rahmenwerk zur Architektur von Informationssystemen im Unternehmensumfeld.

Das ISE-Rahmenwerk berücksichtigt orthogonale Facetten eines Dienstes auf unterschiedlichen Abstraktionsebenen und kann somit auf die typischen Arbeitsabläufe der Dienstentwicklung in größeren, hierarchisch strukturierten Unternehmen und Organisationen abgebildet werden. Auf strategischer Ebene ist die Beschreibung einer Dienstfacette sehr abstrakt, wenig formal und teilweise unstrukturiert. Diese wird auf tieferen Ebenen bis hin zur technischen Ebene konkretisiert und kann schließlich in einer Laufzeitumgebung zur Ausführung gebracht werden. Die unterschiedlichen Dienstfacetten umfassen die externe Dienstbeschreibung, den internen Arbeitsablauf unter Verwendung existierender Dienste im Rahmen eines komponierten Dienstbündels, die verwendeten Datenmodelle, sowie notwendige UI-Beschreibungen und Geschäftsregeln. Abbildung 5.19 zeigt den so aufgespannten Matrix-Aufbau des ISE-Rahmenwerks, in dem jeder Zelle ein spezialisiertes Metamodell zugewiesen ist, welches die Anforderungen an Abstraktionsgrad und Perspektive erfüllt.

Die abgebildete Zuordnung erfolgte hierbei beispielhaft und kann in konkreten Instanziierungen analog zum CRF je nach Erfordernis variieren. Konzeptuell existieren vertikale Modelltransformationen, um Information aus einer höheren Abstraktionsebene in eine konkrete zu überführen. Idealerweise sind auch Transformationen in die entgegengesetzte Richtung möglich, um iterative Modellverfeinerungen im Rahmen eines Roundtrip-Engineering (Kapitel 2.4.2, S. 20) zu erlauben. Abhängigkeiten zwischen verschiedenen orthogonalen Facetten eines Diensten werden durch horizontale Modellintegrationen behoben.

Auf Basis der Eclipse-Plattform stellt die ISE-Werkbank (Scheithauer u. a. 2009) eine prototypische Implementierung des ISE-Rahmenwerks dar. Hierbei kommen konsequent MDA-Standards (Kapitel 2.5, S. 23) zum Einsatz. Metamodelle werden mit Hilfe des MOF-basierten Meta-Metamodells Ecore erstellt. Modelltransformationen werden in relationalem QVT implementiert, sodass potentiell ein Roundtrip-Engineering möglich ist. Darüber hinaus bietet die Werkbank eine Anbindung an die TEXO-Diensteplattform. So können Dienste mit nur einem Klick zu einem Dienstarchiv, in dem alle zur Nutzung notwendigen Artefakte enthalten sind, gepackt und auf die Diensteplattform ausgeliefert werden. Mit entsprechender Authentifizierung und Autorisierung können

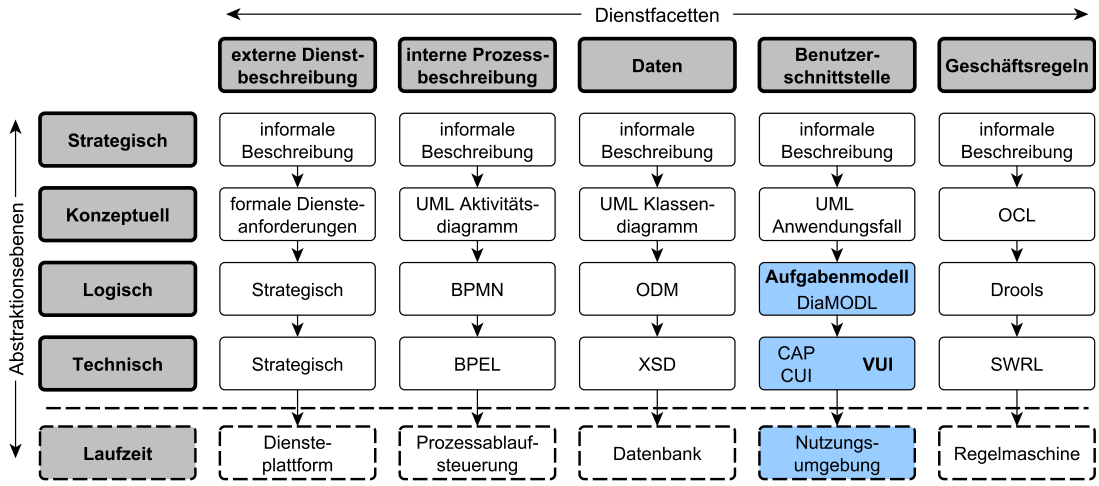


Abbildung 5.19: Die Metamodel-Matrix des ISE-Rahmenwerks nach Cardoso u. a. (2009, S. 23)

außerdem alle Modelle eines ausgelieferten Dienstarchivs aus der ISE-Werkbank heraus abgerufen werden. Das ISE-Dashboard als Teil der Werkbank gießt die Metamodell-Matrix des Rahmenwerks (Abbildung 5.19) in eine grafisches Werkzeug zum Anstoßen von Modelltransformationen. Dadurch werden einem Entwickler die inhärenten Abhängigkeiten in der Matrix als auch die notwendigen Arbeitschritte im Zusammenhang bewusst.

Zur Entwicklung einer Benutzerschnittstelle eines Dienstes wird so etwa die Beschreibung eines UML-Anwendungsfalls auf konzeptueller Ebene in Trættebergs (2003, 2007) DiaMODL-Notation überführt. DiaMODL ist auf der logischen Ebene angesiedelt und erlaubt die feingranulare abstrakte Modellierung von Benutzerinteraktionen. Dazu werden Aspekte des Aufgabenmodells mit denen des AUI verknüpft. Auf technischer Ebene realisieren Constantines (2003) Canonical Abstract Prototypes das konkrete UI, welches sich auf klassische GUI-Interaktion beschränkt. Porta (2010) verfeinert nun das ISE-Rahmenwerk konform zum CRF. So werden Metamodelle und Transformationen zum Erstellen von multimodalen Dialogschnittstellen für Smart Services homogen in die ISE-Matrix integriert. Diese tauschen die zuvor genannten Notationen DiaMODL und Canonical Abstract Prototypes aus. Betroffene Zellen in Abbildung 5.19 sind farblich hervorgehoben, eingefügte Metamodelle sind fett gedruckt.

Die Anwendung der ISE-Methode zur Entwicklung eines Dienstes kann eingebettet sein in eine umfassendere Methode zur Entwicklung eines komplexen (ubiquitären) Informationssystems. So beschreiben Maass und Varshney (2012) ein entsprechendes Entwurfsmodell und wenden dieses zur Erstellung eines Entscheidungsunterstützungssystems im Gesundheitsbereich zur Vermeidung unbeabsichtigter Medikationsfehler an. Auf Basis erzählhafter Schilderungen von Anwendungskontext und konkreter Anwen-

dungssituation wird eine abstrakte schematische Darstellung des komplexen Informationssystems entworfen, die nach den Facetten Informationsbereich, soziales System, Dienstsystem und physischer Objektraum aufgefächert ist.

5.4.2 Unified Service Description Language

Die Unified Service Description Language (USDL) (Barros und Oberle 2012) beschreibt IT-gestützte Dienstleistungen umfänglich gegenüber potentiellen Nutzern. Aus Sicht des Anbieters irrelevante oder schützenswerte Interna eines Dienstes brauchen nicht abgebildet zu werden, obgleich dies die Genauigkeit des Angebotsabgleichs bei der Dienstsuche verringern kann. Eine Dienstbeschreibung in USDL umfasst neben rein technischen auch operationale und geschäftliche Stammdaten eines Dienstes.

Zu den **technischen Stammdaten** zählen analog zu einer WSDL-Beschreibung eines Dienstes etwa konkrete technische Schnittstellen und Protokolle zum Aufrufen eines Dienstes über das Internet. Für Dienste oder Dienstleistungen, die lediglich über das Internet vermittelt werden, z. B. eine Autoreparatur, kann dieser Aspekt vernachlässigt werden. **Operationale Stammdaten** modellieren semantisch die Funktionalität eines Dienstes anhand benötigter Ressourcen sowie Abhängigkeiten und Interaktionen zwischen nachgelagerten Diensten in einer etwaigen Kompositionskette. **Geschäftliche Stammdaten** enthalten Preismodelle in Abhängigkeit von definierten Dienstgütevereinbarungen, rechtliche Aspekte wie Lizenzvereinbarungen und allgemeine Geschäftsbedingungen sowie Information über den Anbieter des Dienstes.

Abbildung 5.20 zeigt die modulare Architektur der USDL in Form von Teilmodellen und deren Abhängigkeiten.

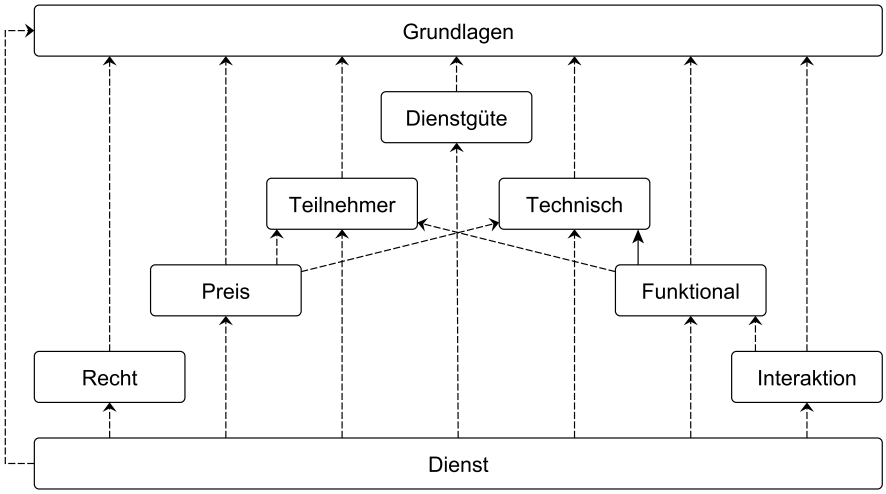


Abbildung 5.20: Die Architektur der USDL nach Barros und Oberle (2012, S. 206)

Jedes Teilmodell ermöglicht die Beschreibung eines Dienstaspektes, wie z. B. Preismo-

delle, rechtliche Rahmenbedingungen oder funktionale und technische Aspekte. Konzepte, die in mehreren Teilmodellen wiederverwendet werden, werden an zentraler Stelle im Grundlagen-Teilmodell erfasst. Im Teilmodell Dienst fließen alle modellierten Aspekte der Dienstbeschreibung zusammen.

Der erste Entwurf der USDL basierte auf einem einfachen XML-Schema (Oberle 2014), welches aufgrund zu geringer Ausdrucksmächtigkeit in MDA-konforme Metamodelle auf Basis von Eclipse-Ecore (Kapitel 2.5, S. 24) überführt und weiterentwickelt wurde. Auf dieser Grundlage wurden Modellierungswerkzeuge und Softwarekomponenten zur Verwaltung von Dienstbeschreibungen entwickelt (Heller u. a. 2012). Um die Nutzerbasis der USDL zu vergrößern und um die Erweiterbarkeit sowie den Grad der Automation im Rahmen des digitalen Verhandelns zu steigern, wurde die Ecore-basierte Variante gemäß Linked-Data-Prinzipien in Linked-USDL (Pedrinaci u. a. 2014) überführt. Die gezeigte Modellarchitektur bleibt trotz Technologiewechsel weiterhin gültig. Jedoch wird die Expressivität der Vorgängerversion nicht erreicht. Pedrinaci u. a. (2014, S. 79) dokumentieren eine Abdeckung von durchschnittlich 74% für die Teilmodelle, die hauptsächlich im Fokus von Linked-USDL stehen.

5.5 Fazit

Anhand der Inhalte der Grundlagenkapitel 2 bis 4 sowie der Stärken und Schwächen der in diesem Kapitel diskutierten verwandten Arbeiten lässt sich ein Anforderungskatalog für die Konzeption und die Implementierung der in dieser Arbeit vorgestellten KomBInoS-Plattform erstellen, um die in Kapitel 1.2 (S. 2) formulierten Ziele zu erreichen. Die Anforderungen dienen gleichzeitig als Kriterien zum Vergleich der verwandten Arbeiten und werden im Folgenden thematisch anhand der wissenschaftlichen und ingenieurtechnischen Fragestellungen gruppiert und erläutert.

5.5.1 Definition eines Anforderungskatalogs

Modellgetriebene Entwicklung von multimodalen Dialogschnittstellen

Eine konzeptuelle und flexible Modellarchitektur konform zum CRF wird in dieser Arbeit als wichtig erachtet, weil so einerseits eine klare Trennung von unterschiedlichen Abstraktionsebenen erreicht wird und andererseits eine Vergleichbarkeit mit ähnlichen Ansätzen möglich ist. Da IT-gestützte Dienste Menschen als Agenten in einen Kollaborationsprozess einbinden, sollte eine formale Repräsentation solcher Prozesse die Grundlage zur modellgetriebenen UI-Entwicklung darstellen. Die Verwendung von Dialogakten ist vor dem Hintergrund der späteren Verwendung eines multimodalen Dialogsystems als Laufzeitumgebung vorteilhaft. Aus dem gleichen Grund kann die starre Aufteilung in Modalitätsklassen entfallen, woraus sich eine größere Flexibilität für den späteren Benutzer der UI ableitet. Das (plattformabhängige) finale UI sollte technologisch auf modernen Webstandards beruhen, um dennoch eine gewisse Unabhängigkeit von einer konkreten Zielplattform zu erreichen.

M_1 Metamodell-Architektur konform zum CRF
M_2 Aufgabenmodell auf Basis von formalen Kollaborationsprozessen
M_3 Abstraktes UI auf Basis von Dialogakten und kommunikativen Funktionen
M_4 Flexibles konkretes UI, keine Unterscheidung von starren Modalitätsklassen
M_5 Finales UI auf Basis von Webstandards

Modellgetriebene Komposition von multimodalen Dialogschnittstellen

Der Begriff der modellgetriebenen Komposition wird in dieser Arbeit als Wiederverwendung existierender UI-Artefakte in unterschiedlichen Situationen interpretiert. Hierzu ist ein kombinierter Kompositionsansatz hilfreich, der gegebenenfalls verfügbare Kompositionsinformation auf Prozessebene ausnutzt. Außerdem wird ein semiautomatischer Kompositionsansatz zur Entwurfszeit bevorzugt, weil nur so ein Entwickler die Möglichkeit hat, zu Gunsten eines positiven Nutzererlebnisses steuernd in die Komposition einzugreifen. Gleichzeitig kann eine nachgelagerte manuelle Verfeinerung als wertvolle Information im Rahmen eines lernenden Ansatzes zur Verbesserung der vorausgehenden automatischen Komposition ausgenutzt werden. Die durch Wiederverwendung bestehender UI-Artefakte erstellte Benutzerschnittstelle muss selbst ebenso wiederverwendbar sein.

C_1 Kombinierter Kompositionsansatz
C_2 Semiautomatische Kompositionserstellung zur Entwurfszeit
C_3 Lernender Kompositionsansatz
C_4 Ergebnis der Komposition wieder komponierbar

Integrierte Entwicklungs- und Kompositionsmethode

Ein strukturiertes, methodisches Vorgehen zur Erstellung von Benutzerschnittstellen definiert in erster Linie eine aufeinander abgestimmte Abfolge verschiedener Entwicklungs-, Transformations- und Kompositionstätigkeiten mit dem Ziel, eine nutzbare Benutzerschnittstelle zu erhalten. Außerdem soll das strukturierte Vorgehen die Verteilung der Tätigkeiten auf mehrere Personen erlauben, etwa durch Berücksichtigung der Abhängigkeiten in der Modellarchitektur. So kann das Vorgehen im Kontext einer umfassenden Dienst(-leistungs-)entwicklung innerhalb einer Organisation eingesetzt werden.

P_1 Strukturierter Erstellungsprozess
P_2 Verteilte Erstellung möglich

Nahtlose Werkzeugunterstützung

Eine integrierte, nahtlose Werkzeugkette ist essentiell zur Umsetzung des strukturierten Erstellungsprozesses. Dies bedeutet die Zusammenführung unterschiedlicher Werkzeuge unter einer gemeinsamen IDE. Durch die Verwendung existierender MDA-Standards wird auf technischer Ebene eine besonders tiefe Integration zwischen der Metamodell-Architektur, Modelltransformationen und Werkzeugen erreicht. Zudem können existierende industrieerprobte Werkzeuge modular wiederverwendet werden. Gleichzeitig

steigen die Verbreitungsmöglichkeit und die Akzeptanz der eigenen Werkzeugkette. Auf methodischer Ebene muss die Werkzeugunterstützung den Entwickler im Erstellungsprozess leiten. Das bedeutet, der Entwickler muss jederzeit den aktuellen Stand der Entwicklung als auch die als nächstes durchzuführenden Arbeitsschritte in Erfahrung bringen können. Die Verwendung einer zentralen Modellablage ermöglicht auf technischer Ebene einerseits die Umsetzung eines verteilten Entwicklungsprozesses. Andererseits kann kompositionsrelevante Information zentral abgelegt und zugreifbar gemacht werden.

- T_1 Integrierte nahtlose Werkzeugkette
- T_2 Verwendung existierender MDA-Standards
- T_3 Geleiteter Erstellungsprozess
- T_4 Verwendung einer zentralen Modellablage

Nutzungs- und Laufzeitumgebung

Die vorgestellten Arbeiten nutzen Laufzeitumgebungen zur Modellinterpretation, zur Kontextüberwachung sowie zur UI-Migration und -Komposition. Diese Aufgaben können prinzipiell auch von einer multimodalen Dialogplattform wahrgenommen werden. Deren Komponenten zur multimodalen Fusion und Fission ermöglichen zudem eine synergistische Multimodalität. Eine serverseitige Laufzeitumgebung erlaubt außerdem die Verwendung leichtgewichtiger mobiler Klienten. In diesem Fall fungiert die Laufzeitumgebung als Middleware zur Interaktion mit Kollaborationsprozessen. Ein Dialogsystem in der Rolle einer solchen Middleware koordiniert prozessbeteiligte Agenten und ermöglicht Dialoge mit gemischter Initiative. Technologisch schafft der Middleware-Ansatz eine cloud-fähige Nutzungsumgebung als Kombination von Laufzeitumgebung und mobilen Klienten, die durch den Einsatz von standardisierten Schnittstellen und adäquaten Kommunikationsprotokollen einfach und austauschbar gekoppelt werden können.

- R_1 Nutzung einer Laufzeitumgebung
- R_2 Unterstützung synergistischer Multimodalität
- R_3 Middleware zur Interaktion mit Kollaborationsprozessen
- R_4 Berücksichtigung leichtgewichtiger mobiler Klienten
- R_5 Laufzeitumgebung koordiniert prozessbeteiligte Agenten
- R_6 Cloud-fähig
- R_7 Verwendung etablierter Kommunikationsstandards

5.5.2 Vergleich der verwandten Arbeiten

Der Vergleich der verwandten Arbeiten in Bezug zu den zuvor definierten Anforderungen auf Basis der Forschungsfragen ist in Tabelle 5.2 aufgeführt. Wird ein Kriterium erfüllt, so wird dies mit einem ✓gekennzeichnet. Eine teilweise Erfüllung wird durch eine Klammerung (✓) ausgedrückt. Wird ein Kriterium nicht erfüllt, so wird dies mit ei-

nem – gekennzeichnet. Eigene Beiträge zum ISE-Rahmenwerk aus (Porta 2010) werden mit einem * ausgezeichnet. Im Vorgriff auf die nachfolgenden Kapitel ist die vorliegende Arbeit bereits im Vergleich enthalten. In Kapitel 11.2 (S. 256) wird nochmals auf diese Tabelle referenziert und entsprechende Begründungen in Form von Kurzzusammenfassungen der Beiträge dieser Arbeit geliefert.

Tabelle 5.2: Vergleich der verwandten Arbeiten

	M1	M2	M3	M4	M5	C1	C2	C3	C4	P1	P2
UsiXML / MultiXML	✓	✓	–	–	✓	–	–	–	–	✓	–
MARIA / TERESA	✓	–	–	–	✓	(✓)	–	–	–	✓	–
EMODE	✓	–	–	–	–	–	–	–	–	✓	✓
Useware Engineering	✓	–	–	–	–	–	–	–	–	✓	–
URC / ASaP	–	–	–	–	✓	(✓)	–	–	–	✓	✓
DomainEditor	–	–	✓	–	–	–	–	–	–	✓	–
ServFace Builder	–	–	–	–	✓	–	–	–	–	✓	–
CRUISe	–	–	–	–	✓	–	(✓)	–	–	✓	–
mashArt	–	–	–	–	✓	–	–	–	–	✓	–
PoSR	–	–	–	–	✓	✓	✓	–	✓	✓	–
UI^{2}Ont	✓	–	–	–	–	–	✓	(✓)	–	–	–
USDL	–	–	–	–	–	–	–	–	(✓)	✓	✓
ISE	✓*	✓*	✓*	✓*	✓*	–	–	–	(✓)	✓	✓
KomBInoS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

	T1	T2	T3	T4	R1	R2	R3	R4	R5	R6	R7
UsiXML / MultiXML	–	–	–	–	–	–	–	–	–	–	–
MARIA / TERESA	✓	–	–	–	✓	–	(✓)	–	–	–	–
EMODE	✓	✓	–	✓	✓	–	–	–	–	–	–
Useware Engineering	–	–	–	–	(✓)	–	–	–	–	–	–
URC / ASaP	✓	✓	(✓)	✓	✓	(✓)	–	✓	–	✓	✓
DomainEditor	✓	–	(✓)	–	✓	(✓)	–	–	–	–	–
ServFace Builder	✓	–	–	✓	–	–	–	✓	–	–	–
CRUISe	✓	–	–	✓	✓	–	–	–	–	–	–
mashArt	✓	–	–	✓	✓	–	✓	–	–	✓	–
PoSR	✓	–	–	✓	–	–	–	–	–	–	–
UI^{2}Ont	–	–	–	✓	✓	–	–	–	–	–	–
USDL	✓	✓	–	✓	–	–	–	–	–	–	–
ISE	✓	✓	✓	✓	✓*	✓*	✓*	✓*	✓*	✓*	✓*
KomBInoS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Teil II

Ein ganzheitlicher Ansatz zur modellgetriebenen Entwicklung und Komposition von multimodalen Dialogschnittstellen für Smart Services

KAPITEL 6

Die KomBInoS-Metamodell-Architektur

Dieses Kapitel diskutiert die Metamodell-Architektur von KomBInoS zur Modellierung von multimodalen Dialogschnittstellen für Smart Services. Die konzipierte Architektur ermöglicht eine integrierte modellgetriebene Entwicklungs- und Kompositionsmethode zur Erstellung solcher Benutzerschnittstellen. Es werden die Forschungsfragen (1) und (2) adressiert und in Bezug auf die notwendige Metamodellierung beantwortet.

6.1 Einleitung

Eine integrierte Methode zur modellgetriebenen Entwicklung und Komposition von multimodalen Dialogschnittstellen für Smart Services bedarf einer abgestimmten Metamodell-Architektur, welche in der Lage ist, (1) alle relevanten Facetten einer solchen UI abzubilden, (2) ein modellgetriebenes Vorgehen zur Entwicklung dieser UIs zu ermöglichen und (3) gleichzeitig die Kompositionsfähigkeit von Diensten auf die UI selbst auszudehnen. Eine solche Metamodell-Architektur wird in diesem Kapitel konzipiert. Zur nahtlosen Anwendung der MDA (Kapitel 2.5, S. 23) wurden die Metamodelle mit Hilfe von Ecore, dem Essential MOF-konformen Meta-Metamodell des Eclipse Modeling Framework (EMF), umgesetzt. Die Metamodelle wurden teils von Grund auf neu entwickelt, teilweise wurden aber auch existierende Metamodelle wiederverwendet. Die Wiederverwendung existierender Metamodelle erfolgt abgestuft. Metamodelle, die einen Standard abbilden werden unverändert verwendet, um deren Konformität zum jeweiligen Standard nicht zu beeinflussen. Abhängigkeiten zu diesen Metamodellen werden ausschließlich durch Aggregation geschaffen. Dies betrifft etwa Metamodelle zur Repräsentation von Kollaborationsprozessen, die den Ausgangspunkt für eine KomBInoS-UI darstellen. Der Einsatz von Ecore erlaubt eine direkte Verwendung von Metamodellen der SiAM-Dialogplattform. Diese werden durch Vererbung funktional ergänzt. Dadurch erhöht sich der Grad der Abhängigkeit aber auch die Integrationstiefe, was der Rolle

der SiAM-Dialogplattform als Zielplattform des modellgetriebenen Ansatzes im Rahmen der in Kapitel 9 (S. 211) vorgestellte KomBInoS-Laufzeitumgebung zuträglich ist. Außerdem kann die spätere Nutzung bereits existierender Werkzeuge für die Modellerstellung und -transformation sowie die Implementierung einer eigenen angepassten Werkbank innerhalb der Eclipse-Plattform sowie einer Laufzeitumgebung sinnvoll gewährleistet werden.

6.2 Anwendungsmodell

Das Anwendungsmodell dient als Container, welcher alle nachfolgend erzeugten Artefakte an zentraler Stelle miteinander verknüpft. Abbildung 6.1 zeigt den Zusammenhang der beteiligten Teilmodelle. So ermöglicht eine KomBInoS-Anwendung (Application) den Zugriff auf alle Teilmodelle der Benutzerschnittstelle. Im Einzelnen sind dies das Prozessmodell (ProcessModel), das Ressourcenmodell (ResourceModel), das Domänenmodell (EPackage), das Kompositionsmodell (CompositionModel), das Aufgabenmodell (TaskModel), das Dialogaktmodell (DialogueActModel), das Modell für die grafische Benutzerschnittstelle (GuiModel), Modelle für Spracheingabe (SpeechInModel) und Sprachausgabe (SpeechOutModel) sowie weitere paralinguistische Ein- und Ausgabemodelle, etwa zur Gesteninterpretation (GestureModel) oder zur Steuerung eines virtuellen Charakters (CharacterModel). Die angewendete Farbkodierung wird zur leichteren Identifizierung der Teilmodellzugehörigkeit einzelner Konzepte in späteren Abbildungen wiederverwendet. Das Prozessmodell abstrahiert von konkreten Prozessmodellierungsstandards, die abgeleiteten Klassen CmmnModel und BpmnModel dienen lediglich als Hülle, die auf das jeweilige Wurzelelement der entsprechenden Kollaborationsprozesse (Definitions) verweisen. Das Ressourcenmodell beinhaltet die dialogrelevante Rollenmodellierung (Role). Diese werden im initialen Transformationsschritt aus dem Kollaborationsprozess extrahiert. ActivityWrapper dienen zum Kapseln von Aktivitäten eines Kollaborationsprozesses. Darin enthalten sind für die jeweilige Aktivität relevante Prozessvariablen (Variable) sowie eine Abbildung der Variablen auf Konzepte des dialogspezifischen Domänenmodells (DomainMapping).

6.3 Prozessmodelle

In Kapitel 3.4.3.2 (S. 41) wurden Modellierungssprachen für Kollaborationsprozesse vorgestellt. Aufgrund des breiten Einsatzgebiets und der Verfügbarkeit von standardisierten Metamodellen werden hier die BPMN 2.0 und die CMMN 1.1 berücksichtigt. Weitere standardisierte Prozess-Metamodelle können modular hinzugefügt werden. Wie in Abbildung 6.1 beispielhaft gezeigt, ist die Integration eines in Zukunft öffentlich verfügbaren Referenz-Metamodells für (erweiterte) EPKs (EpcModel) bereits vorbereitet.

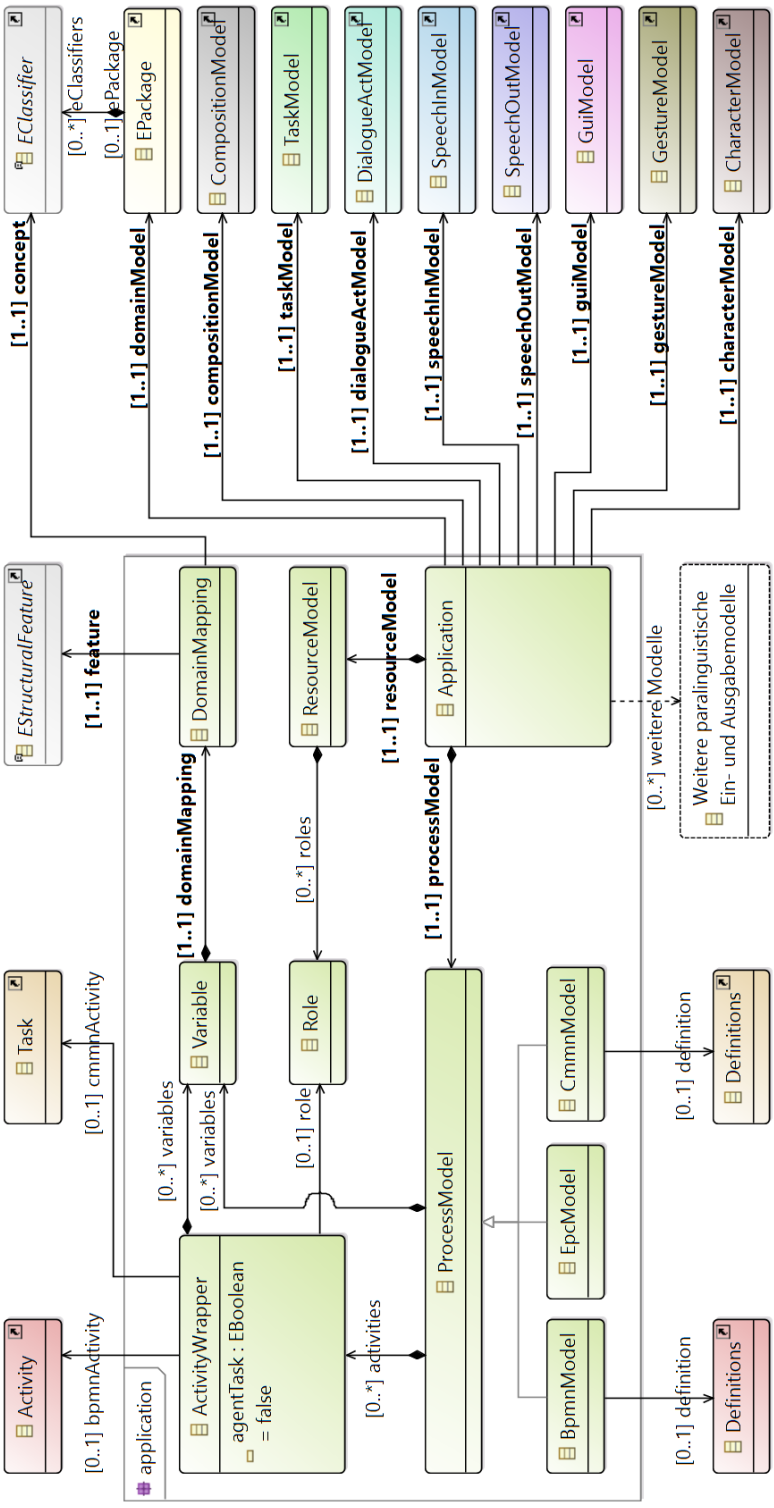


Abbildung 6.1: Das KomBInoS-Anwendungsmodell

6.3.1 BPMN 2.0

Für die BPMN 2.0 existiert ein öffentlich verfügbares Referenz-Metamodell auf Ecore-Basis[1], welches auch hier eingebunden wird. Abbildung 6.2 zeigt einen zur besseren Lesbarkeit auf das Wesentliche reduzierten Ausschnitt des Modells.

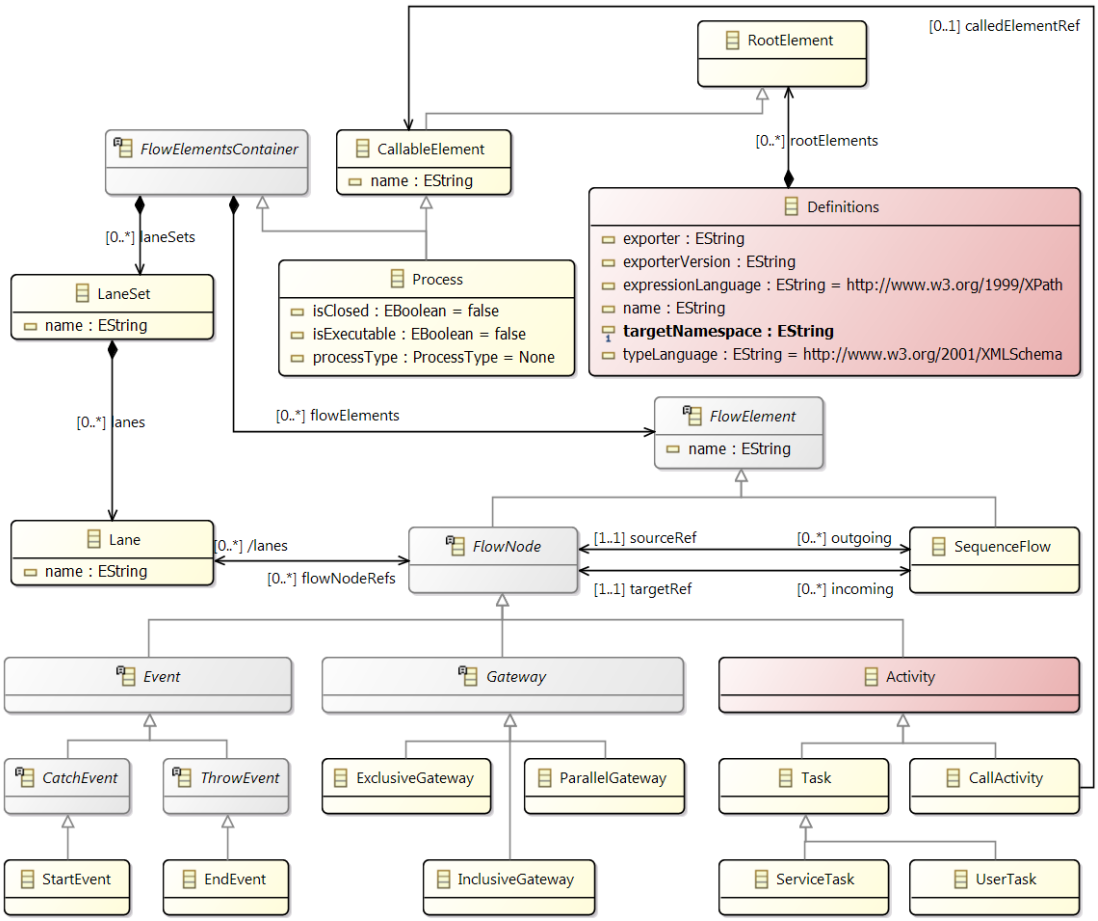


Abbildung 6.2: Ausschnitt aus dem in KomBInoS verwendeten BPMN 2.0 Metamodell

Ein BPMN-Prozess (Process) enthält eine Menge von Aktivitäten (Activity), die über Sequenzflüsse (SequenceFlow) und Entscheidungspunkte (Gateway) logisch und temporal miteinander verknüpft sind. Aktivitäten beschreiben Aufgaben (Task), die entweder von einem (nicht notwendigerweise menschlichen) Benutzer, also einem Agenten, durchzuführen sind (UserTask) oder automatisiert ablaufen (z. B. ServiceTask). Eine Aktivität kann einer Bahn (Lane) zugeordnet sein, welche etwa einer betrieblichen Organisationseinheit, also einer Rolle, entspricht. Ein Prozess wird durch mindestens ein Startereignis (StartEvent) gestartet. Ein Endeereignis (EndEvent) markiert das Ende eines Prozesses.

[1] siehe http://wiki.eclipse.org/MDT-BPMN2 (letzter Zugriff: 04.11.2017)

Wie Abbildung 6.3 zeigt, kann eine Aktivität darüber hinaus eine Datenschnittstelle (InputOutputSpecification) spezifizieren. Diese enthält mindestens eine Eingabemenge (InputSet) von eingehenden Daten (DataInput) und eine Ausgabemenge (OutputSet) von ausgehenden Daten (DataOutput). Der Datenfluss zwischen Aktivitäten wird über Datenassoziationen (DataInputAssociations, DataOutputAssociations) spezifiziert, die zwei Datenobjekte in Beziehung setzen.

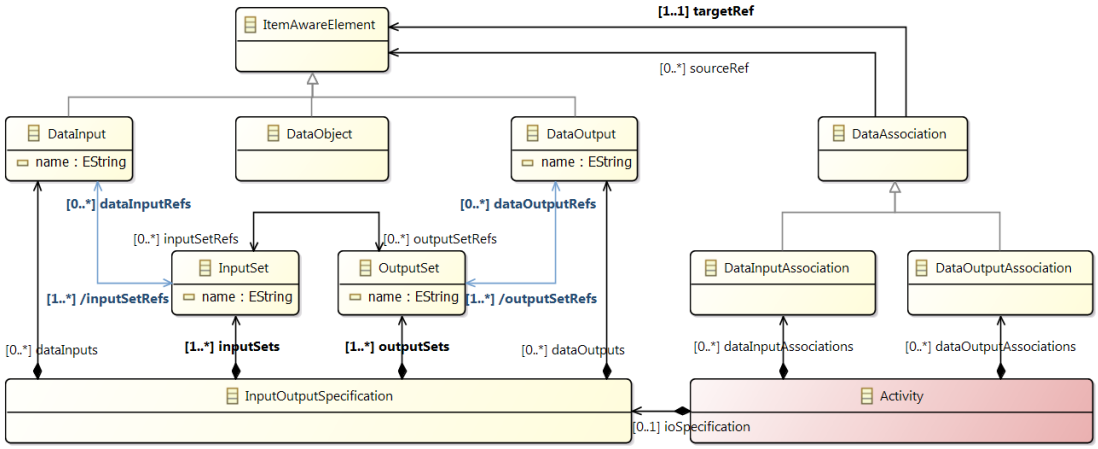


Abbildung 6.3: Datenmodellierung und -fluss in einem BPMN 2.0 Metamodell

Neben diesen gezeigten Konzepten der BPMN existieren weitere Konzepte zur genaueren Spezifikation von Aktivitäten, Ereignissen und Entscheidungspunkten, die im Rahmen dieser Arbeit nicht von Bedeutung sind. Darum wird, wie im bereits vorgestellten PoSR-System (Kapitel 5.3.3, S. 97), die BPMN in dieser Arbeit auf die gezeigte Teilmenge eingeschränkt. Dies erleichtert bei nahezu gleicher Ausdrucksmächtigkeit die spätere Spezifikation einer Transformation ins Aufgabenmodell.

6.3.2 CMMN 1.1

Ebenfalls interessant ist die Betrachtung der CMMN für die Modellierung von weniger strukturierten Ad-hoc-Kollaborationsprozessen. Entsprechend modellierte CMMN-Prozesse sind wie BPMN direkt in einer entsprechenden Prozessausführungsmaschine ausführbar. Im Rahmen dieser Dissertation wurde ein darüber hinaus wiederverwendbares CMMN-Metamodell auf Ecore-Basis anhand des normativen XML-Schemas[2] teilautomatisch mit Hilfe der EMF-eigenen Schema-Importfunktion und einer anschließenden manuellen Glättung von Konzeptnamen erstellt. Somit hat dieses Metamodell, welches auszugsweise in Abbildung 6.4 dargestellt ist, Referenzcharakter.

Ein Fall (Case) umfasst die vollständige Fallbeschreibung. Darin ist u. a. eine Rollenmodellierung (Role) enthalten. Eine Fallbeschreibung unterteilt sich in mehrere Pha-

[2]siehe http://www.omg.org/spec/CMMN/20151109/CMMN11.xsd (letzter Zugriff: 04.11.2017)

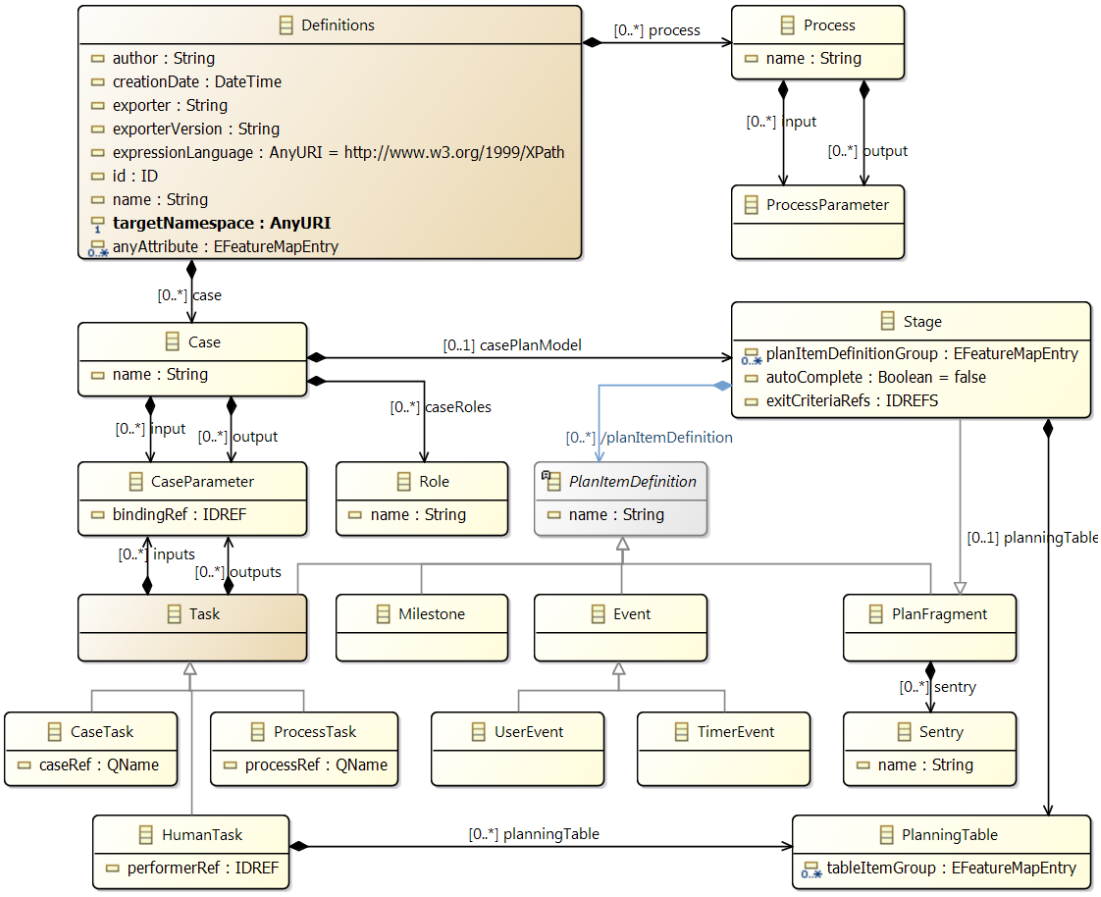


Abbildung 6.4: Ausschnitt aus dem CMMN 1.1 Metamodell von KomBInoS

sen (Stage), die eine Menge von Aufgaben (Task) umfassen und durch einen Meilenstein (MileStone) abgeschlossen werden. Eine Aufgabe startet entweder eine neue Fallbearbeitung (CaseTask) oder Prozessinstanz (ProcessTask) oder muss von einem (trotz des Namens nicht notwendigerweise menschlichen) Agenten durchgeführt werden (HumanTask). Entscheidungspunkte (Sentry) stellen in CMMN eine Kombination von Ereignis (Event) und Bedingung dar, welche den Verlauf der Fallbehandlung beeinflussen können. Über die Verknüpfung solcher Entscheidungspunkte lassen sich Abhängigkeiten zwischen Aufgaben und Phasen abbilden.

CMMN und BPMN verbinden einige Ähnlichkeiten, z. B. Konzepte für die Modellierung von Rollen, Aufgaben und Daten. Außerdem verfügen beide Standards über eine operationelle Semantik, die entsprechende Modelle ausführbar werden lassen. Jedoch können aufgrund der unterschiedlichen Anwendungsgebiete der beiden Standards auch fundamentale Unterschiede in den Metamodellen identifiziert werden. Allen voran die Diskrepanz zwischen Kontrollflussmodellierung in der BPMN und Abhängigkeitsmodel-

lierung in der CMMN. Die Modellierung des Aufgabenmodells, das im Rahmen eines modellgetriebenen Vorgehens Konzepte beider Standards unterstützt, muss Gemeinsamkeiten und Unterschiede für den im Vordergrund stehenden Kollaborations- und Interaktionszweck in geeignetem Maße berücksichtigen.

6.3.3 KomBInoS-spezifische Erweiterungen

Bei der Erstellung von ausführbaren Geschäfts- bzw. Kollaborationsprozessen können vereinbarte Modellierungsrichtlinien zur Etablierung einer guten Lesbarkeit angewendet werden, die den Kontrollfluss leicht verständlich machen sollen. Die Varianz bei der Modellierung des Datenflusses innerhalb eines ausführbaren Prozesses ist vielfach höher. Auf konzeptioneller Ebene geschieht dies mit Hilfe von gegebenenfalls untereinander verknüpften Daten- bzw. Parameter-Objekten, die mit Aufgaben assoziiert sind. Technisch werden diese Datenobjekte als Prozessvariablen mit eingeschränkter Sichtbarkeit realisiert. Um dies leicht und nachvollziehbar zu bewerkstelligen, haben manche Anbieter einer Prozessausführungsmaschine eigene Erweiterungen des jeweiligen Prozessmodellierungsstandards mit Hilfe der vorgesehenen Erweiterungsmechanismen auf Basis von XML-Schema definiert, die von der angebotenen Prozessausführungsmaschine unterstützt werden. Jedoch können in der Praxis Prozessvariablen auch völlig intransparent und unabhängig von der Prozessdefinition in einer Prozessausführungsmaschine erzeugt und verwaltet werden, z. B. durch HTML-Formulare auf Basis von Java Server Pages. Ebenso kann eine Abbildung von einfach typisierten Prozessvariablen auf komplexere Domänenmodelle in der implementierten Geschäftslogik erfolgen.

Die KomBInoS-Erweiterung für Prozessmodelle wird in Abbildung 6.5 samt beispielhafter Anwendung vorgestellt. Diese schafft die Möglichkeit, den durch Prozessvariablen gestalteten Datenfluss innerhalb eines BMPN- oder CMMN-basierten Kollaborationsprozesses in Form einer anbieterspezifischen Erweiterung explizit und somit transparent zu erfassen. Es können alle Elemente eines Prozesses annotiert werden. Relevant sind insbesondere der Prozess selbst, seine Aktivitäten sowie Start- und Endeereignisse. Die Annotation ist optional, aber auch unabhängig von der Methodenanwendung dem tieferen Prozessverständnis zuträglich. Auf die Ausführung eines Prozesses nimmt sie keinen Einfluss. Konkret spezifiziert eine Variablenannotation (VariableAnnotation) neben der Angabe des Variablennamens auch den atomaren Typ der Variable (*boolean*, *int*, *string*, etc.) sowie die Flussrichtung der Variablen. Eingehende Variablen (*in*) werden gelesen, ausgehende Variablen (*out*) geschrieben, auf ein- und ausgehende Variablen (*inout*) wird in einer Aufgabe zuerst lesend und schließlich schreibend zugegriffen. Die XML-Serialisierung zeigt die Annotation einer Beipielaufgabe mit zwei Variablen *a* vom Typ *string* sowie *b* vom Typ *int*. Während die Variable *a* gelesen wird, beschreibt die Variable *b* etwa das Ergebnis der Aufgabendurchführung. Mit Hilfe der geschilderten Erweiterung wird es insbesondere möglich, das Ein- und Ausgabeverhalten von Aufgaben explizit zu erfassen, um auf dieser Grundlage zu wissen, welche Information zur Verfügung steht und welche zu liefern ist.

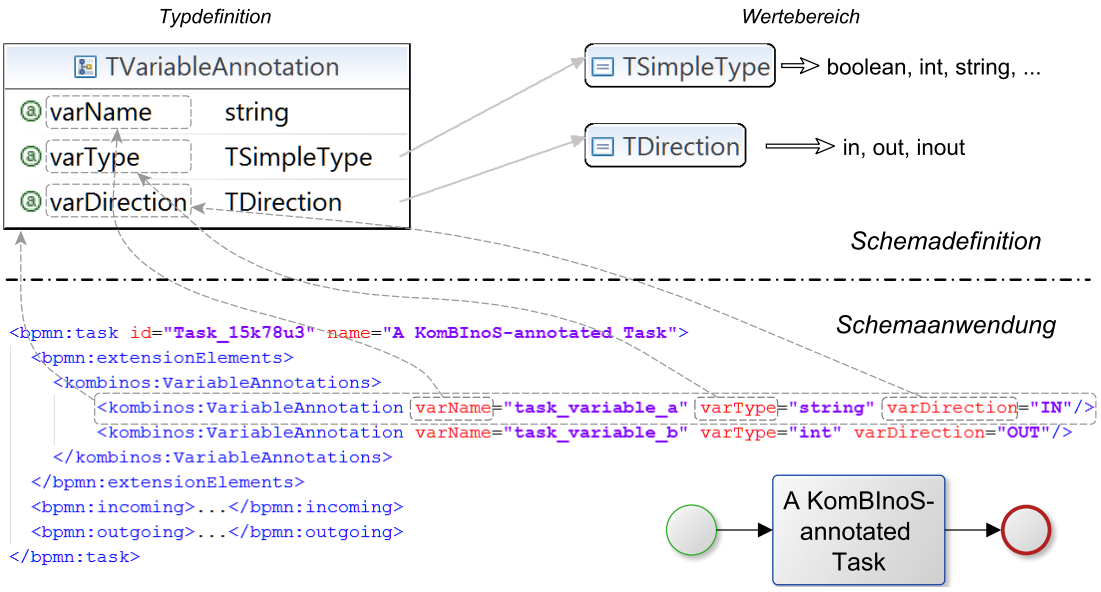


Abbildung 6.5: KomBInoS-Erweiterung von BPMN und CMMN um expliziten Daten- bzw. Variablenfluss

6.4 Domänenmodell

Die Erstellung eines Domänenmodells für eine KomBInoS-UI erfolgt direkt auf Basis des Meta-Metamodells Ecore. Dieses ist in Abbildung 6.6 auf das Wesentliche reduziert dargestellt. Beliebige Domänenkonzepte können mit Hilfe von Klassen (EClass) repräsentiert und in einem gemeinsamen Paket (EPackage) gebündelt werden. Domänenkonzepte verfügen über Attribute (EAttribute) und Referenzen (EReference) auf andere Domänenkonzepte.

Unterschiedliche Dienste verwenden individuelle Domänenmodelle. Auch aus Sicht einer wiederverwendbaren multimodalen Dialogplattform sowie eines darauf aufbauenden modellgetriebenen Entwicklungs- und Kompositionsansatzes für KomBInoS-UIs gibt es außerdem die Notwendigkeit eines gemeinsamen abstrakten Domänenmodells, welches potentielle Gegenstände eines Dialogs einheitlich abbildet. Dazu gehören prinzipiell alle Entitäten, auf die im Dialog durch ein Diskusphänomen (Kapitel 4.3.3, S. 62) referenziert oder die durch einen individuellen Namen bezeichnet werden können. SiAM-dp gibt in diesem Zusammenhang aufgrund der sehr breit gestreuten Einsatzmöglichkeiten vor, dass alle Domänenobjekte vom abstrakten Konzept Entity bzw. NamedEntity abgeleitet sein sollen. Das KomBInoS-Domänenmodell bildet nun zusätzliche kollaborationsrelevante Konzepte in Anlehnung an etablierte SemanticWeb-Standards ab, wie z. B. Friend-of-a-Friend[3] zur Repräsentation von Agenten und deren Beziehungen un-

[3] siehe http://xmlns.com/foaf/spec/ (letzter Zugriff: 04.11.2017)

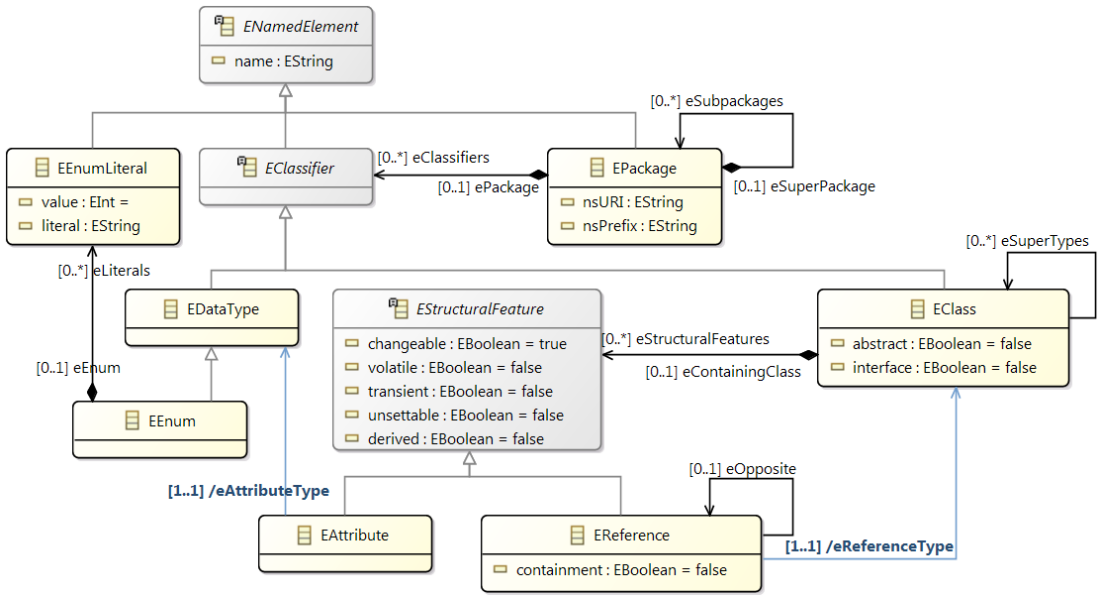


Abbildung 6.6: Ausschnitt aus dem Meta-Metamodell Ecore

tereinander oder DublinCore[4] zur Beschreibung von digitalen Inhalten im Internet. Weitere können hinzugefügt werden. Abbildung 6.7 zeigt die initialen Konzepte des KomBInoS-Domänenmodels, welche als Ausgangspunkt für davon abgeleitete Konzepte individueller Domänenmodelle konkreter KomBInoS-Anwendungen dienen.

6.5 Aufgabenmodell

Metamodelle für die Aufgabenmodellierung wurden in der Literatur vielfach beschrieben und existieren für die unterschiedlichsten Zwecke. Balbo u. a. (2004) vergleichen verschiedene Notationen anhand relevanter Kriterien wie Zweck, Abdeckung, Adaptivität und Erweiterbarkeit. Sie berücksichtigen aber auch eher weiche Kriterien wie die Nutzbarkeit hinsichtlich Kommunikation und Modellierung.

6.5.1 Anforderungen

Der Zweck des im Folgenden vorgestellten Aufgabenmodells definiert sich einerseits über den Einsatz des Modells im Rahmen der modellgetriebenen Entwicklung von multimodalen Dialogschnittstellen für Smart Services. Andererseits müssen auch Anforderungen bzgl. der Laufzeit, also der Nutzungsphase der resultierenden UIs (Kapitel 3.3, S. 33), sowie rein technische Anforderungen berücksichtigt werden. Insgesamt ergibt

[4] siehe http://dublincore.org/documents/dcmi-terms/ (letzter Zugriff: 04.11.2017)

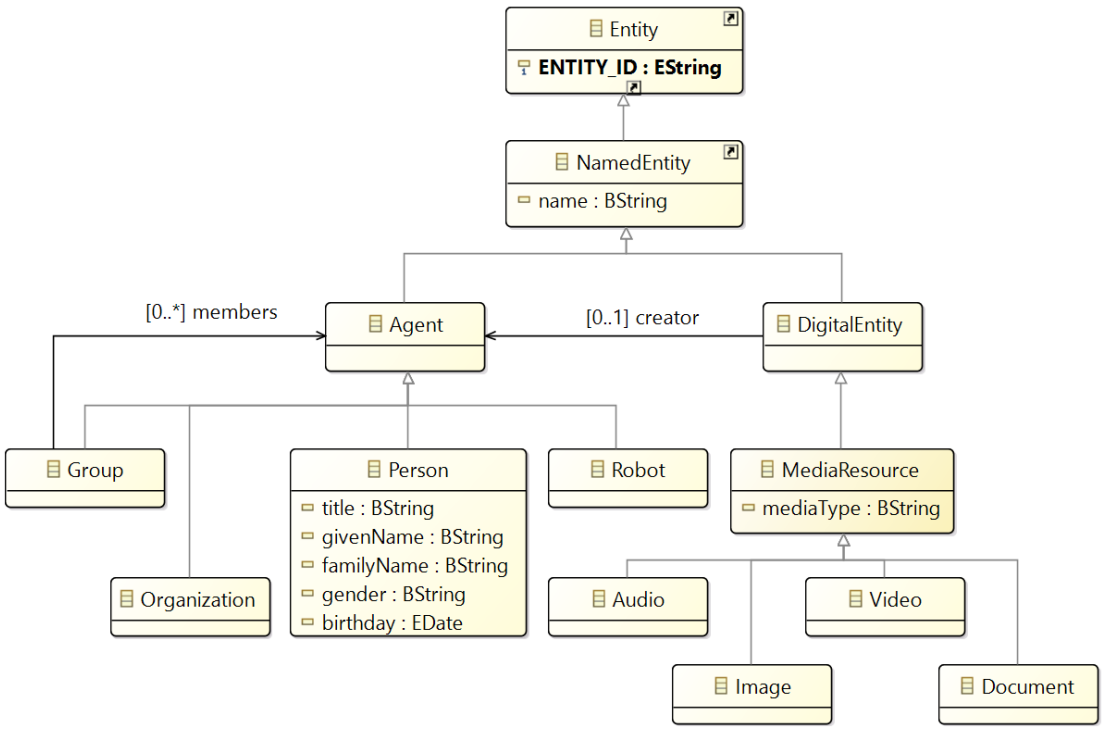


Abbildung 6.7: Ausschnitt aus dem KomBInoS-Domänenmodell

sich eine sieben Punkte umfassende Liste. Die Punkte (1) bis (3) sind hierbei konzeptueller Natur, die Punkte (4) und (5) adressieren technische Aspekte und die Punkte (6) und (7) betreffen die Nutzungsphase.

(1) **Strukturierte Aufgabenmodellierung.** Das Aufgabenmodell soll eine strukturierte Aufgabenmodellierung ermöglichen, die die verwendeten Konzepte der unterstützten Prozessmodellierungssprachen in ausreichendem Maße berücksichtigt, damit ein Agent seine aktuell zugewiesene Aufgabe sowie deren Auswirkungen im Dialog auf einer Metaebene ergründen kann. Dadurch kann eine generische Assistenzfunktion auf Ebene des Aufgabenmodells realisiert werden, die Antworten bzgl. des Prozessfortschritts auf Fragen wie z. B. „*Was muss ich tun, um die Aufgabe abzuschießen?*“, „*Was kommt als Nächstes?*“ oder „*Wo stehe ich gerade im Prozess?*“ geben kann.

(2) **Feingranulare Repräsentation von Daten und Datenflüssen.** Kollaborationsprozesse operieren sowohl ereignis- als auch datengetrieben. Im Hinblick auf eine multimodale Dialoginteraktion mit solchen Prozessen zur Erledigung einzelner Aufgaben soll neben einer adäquaten Modellierung von Aufgaben ein besonderer Fokus auf die feingranulare Repräsentation von Daten und Datenflüssen gelegt werden.

(3) **Erweiterbarkeit.** Perspektivisch können, wie im vorangegangenen Kapitel 6.3

(S. 112) geschildert, weitere Prozessmodellierungssprachen berücksichtigt werden. Folglich muss eine gewisse Erweiterbarkeit bzgl. etwaiger neuer Konzepte gewährleistet sein.

(4) **Einfache Modelltransformationen.** Ein Aufgabenmodell wird in einem modellgetriebenen Vorgehen aus einem Kollaborationsprozess abgeleitet. Durch geeignete Maßnahmen soll es in technischer Hinsicht leicht möglich sein, eine entsprechende Modelltransformation zu implementieren. Außerdem werden auf Basis eines Aufgabenmodells weitere Teilmodelle generativ erzeugt. Für diesen Transformationsprozess muss ein Aufgabenmodell zur Entwicklungszeit die notwendige Information beinhalten und leicht abfragbar machen.

(5) **Geeignete Werkzeugunterstützung.** Die in dieser Arbeit vorgestellte Methodik sowohl zur Entwicklung als auch zur Komposition implementiert eine semiautomatische Methode, die teilweise manuelle Verfeinerungen nach einem automatischen Schritt erfordert. Beim Entwurf des Aufgabenmodells muss folglich die Eignung des Modells zur einfachen Realisierung einer adäquaten Werkzeugunterstützung beachtet werden.

(6) **Rollen- und fähigkeitsbasierte Zuweisung.** Darüber hinaus sollen Aufgaben zur Laufzeit an einen konkreten Agenten rollen- und fähigkeitsbasiert zugewiesen und abgeschlossen werden können.

(7) **Synchronisierung.** Im Gegensatz zu ConcurTaskTrees von Paternò u. a. (1997) oder dem planbasierten ANSI/CEA-2018 Standard zur Aufgabenmodellierung (CEA 2008) muss ein Aufgabenmodell in KomBInoS selbst nicht ausführbar sein. Dies wird bereits auf Ebene des Prozessmodells durch eine dafür vorgesehene Prozessausführungsmaschine bewerkstelligt. Stattdessen muss ein Aufgabenmodell aber zur Laufzeit mit dem tatsächlich ausgeführten Kollaborationsprozess synchronisiert werden können. Hierzu müssen einzelne Aufgaben des Aufgabenmodells mit ihren Entsprechungen im Kollaborationsprozess korreliert werden können.

6.5.2 Entwurf

Abbildung 6.8 zeigt das Aufgaben-Metamodell von KomBInoS. Ein Aufgabenmodell (TaskModel) beschreibt aufgrund der Möglichkeiten der berücksichtigten BPMN-Notation einen gerichteten zyklischen Graphen und besteht folglich abstrakt aus je einer Liste von Knoten (Node) und Kanten (Edge), die sich gegenseitig referenzieren. Strukturbeschreibende Knoten (FlowNodes) des Aufgabenmodells sind entweder Entscheidungspunkte (Gateways), welche zum Verzweigen und Zusammenführen von Kontrollflüssen (TaskFlow) dienen, externe Ereignisse (Trigger) oder Aufgaben (Task). Letztere werden wiederum nach atomaren (AtomicTask) und zusammengesetzten Aufgaben (CompositeTask) unterschieden. Das Task-Attribut *agentTask* gibt an, ob ein physischer Agent im Sinne von Abbildung 6.7 die Aufgabe bearbeiten muss. Hier kommen also prinzipiell menschliche Benutzer, kollaborative Roboter oder (gemischte) Teams in Frage.

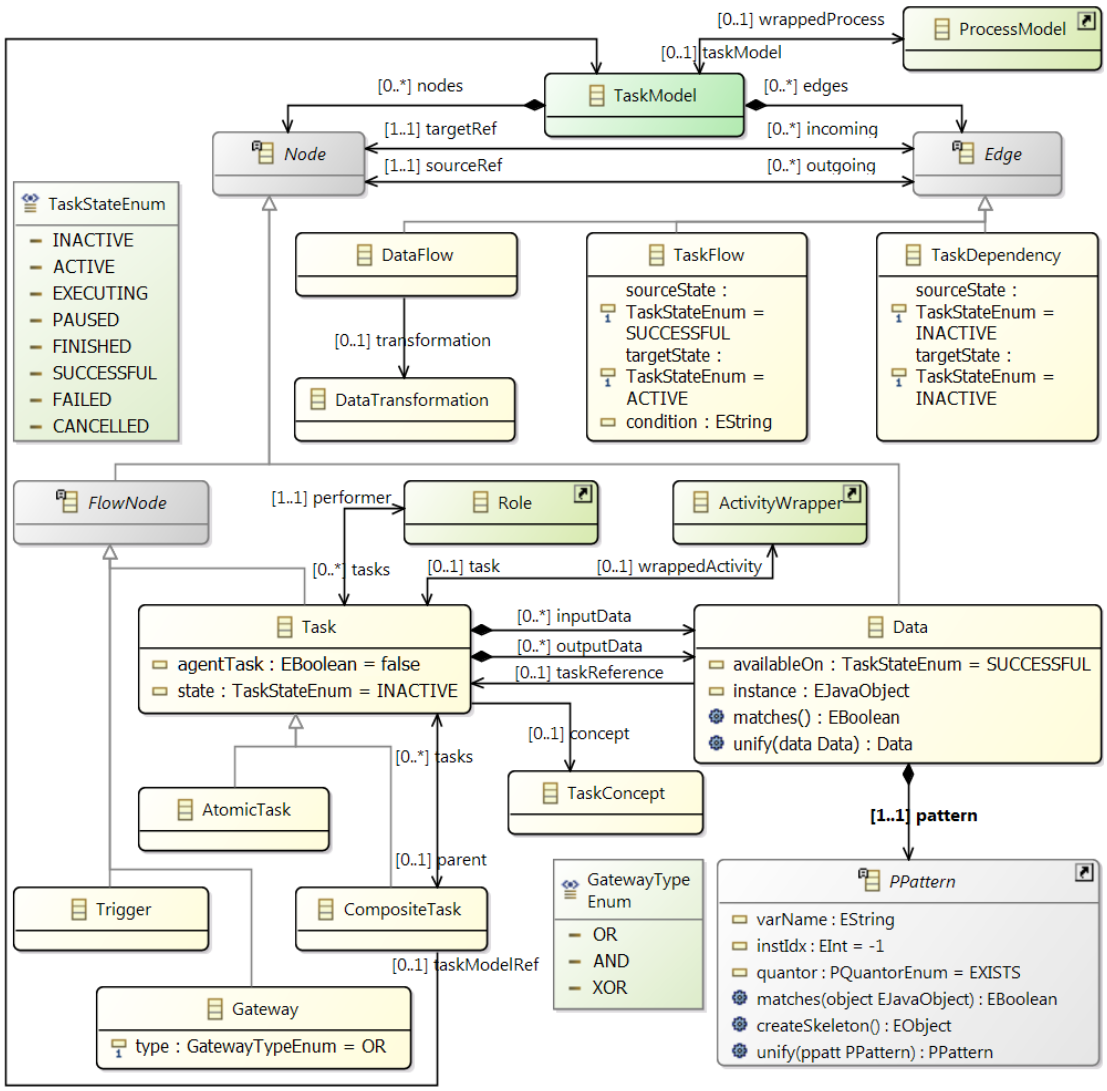


Abbildung 6.8: Das KomBInoS-Aufgabenmodell

Die nähere Bestimmung erfolgt über die mit der Aufgabe verknüpften Rolle. Eine Aufgabe verfügt außerdem über einen zur Laufzeit relevanten Zustand (TaskStateEnum). Über die Verknüpfung zu einem Aufgabenkonzept (TaskConcept) kann einer Aufgabe eine tiefere semantische Bedeutung jenseits ihres Namens oder einer textuellen Beschreibung beigemessen werden. Eine grundlegende Charakterisierung von (Agenten-) Aufgaben ist in Abbildung 6.9 auszugsweise dargestellt. Hierüber kann auch ein Fähigkeitenabgleich im Rahmen einer Aufgabenzuweisung stattfinden. Mit Hilfe des TaskConcept-Attributs *uri* kann zudem bei Bedarf in eine komplexere ontologische Modellierung verlinkt werden.

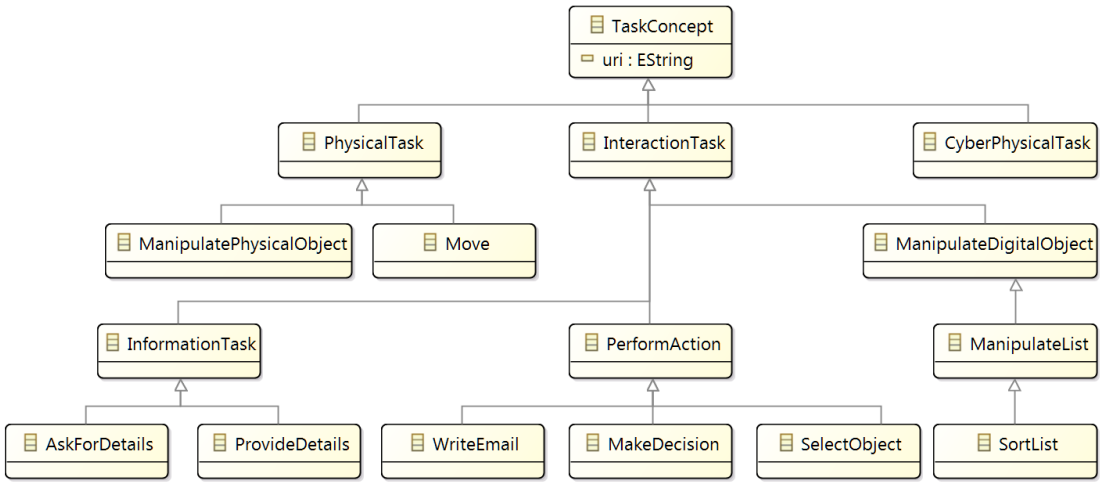


Abbildung 6.9: Hierarchische Charakterisierung von Aufgaben in KomBInoS

Schließlich kann eine Aufgabe durch Datenelemente (Data) mit einer feingranularen Datenspezifikation versehen werden. Eingangsdaten (*inputData*) stehen bei Beginn der Aufgabendurchführung zur Nutzung bereit. Auf dieser Basis können etwa von einem Agenten Entscheidungen getroffen oder Webdienste aufgerufen werden. Ausgangsdaten (*outputData*) stehen erst bei Beendigung der Aufgabe zur Verfügung, z. B. wenn ein Agent die benötigte Information angegeben oder ein Webdienst ein Ergebnis geliefert hat. Der genaue Zeitpunkt der Datenverfügbarkeit wird über das Data-Attribut *availableOn* spezifiziert. Datenelemente können auch direkt auf Ebene des Aufgabenmodells angegeben werden. Dies kann etwa dazu dienen, Sensorwerte oder allgemein Kontextinformation abzubilden. Die exakte Struktur eines Datenelements wird mit Hilfe eines Musters (PPattern) definiert. Das dazu verwendete Metamodell der SiAM-Plattform wird in Anhang C (S. 273) samt Beispiel im Detail erläutert. Es erlaubt über die Strukturangabe hinaus auch die Spezifikation von tieferen inhaltsbezogenen Einschränkungen. Durch eine von SiAM-dp angebotene Programmierschnittstelle kann ein Muster mit einer (vermeintlichen) Instanz, aber auch mit einem anderen Muster verglichen werden.

Es gibt drei verschiedene Arten von Kanten. Es lassen sich einerseits Kontrollflüsse

(TaskFlow) zwischen Aufgaben modellieren, andererseits können auch Abhängigkeiten (TaskDependency) zwischen Aufgaben dargestellt werden. Darüber hinaus können Datenflüsse (DataFlow) zwischen Aufgaben modelliert werden, wodurch implizit auch eine Form der Abhängigkeit zwischen Aufgaben entsteht.

Der Entwurf eines Aufgabenmodells einerseits zur Verwendung in Modelltransformationen und andererseits für eine werkzeugunterstützte manuelle Verfeinerungen muss diese prinzipiell gegensätzlichen Anwendungsgebiete gegeneinander abwägen. Ein restriktives, hierarchisches Metamodell erlaubt wenig Spielraum für Fehlmodellierungen und eine leichte Erstellung eines adäquaten Editors zum Preis komplexerer Modelltransformationen. Im Rahmen dieser Arbeit wurde sich bewusst für eine flache Modellierung des Aufgabenmodells entschieden. Dadurch können Modelltransformationen einfach gestaltet werden. Im Gegenzug müssen die gewährten Modellierungsfreiheiten, welche Fehlmodellierungen zulassen, durch formale Einschränkungen beschnitten werden. Dies wurde mit Hilfe von *Non-API EClassifier Invarianten* (Steinberg u. a. 2009, S. 549ff) der OCL (Kapitel 2.5, S. 25) umgesetzt. Abbildung 6.10 zeigt eine entsprechende Annotation der Klasse TaskFlow, die den Wertebereich der *sourceRef*- und *targetRef*-Referenzen auf Instanzen vom Typ FlowNode sowie davon abgeleitete Typen einschränkt.

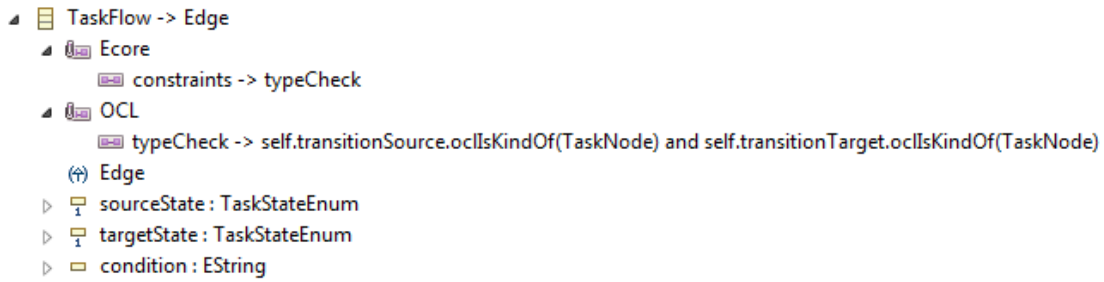


Abbildung 6.10: OCL-Invariante zum Einschränken des Wertebereichts der Referenzen *sourceRef* und *targetRef* in der Klasse TaskFlow

Durch diese Einschränkungen kann ein Aufgabenmodell auf statische Validität und Plausibilität hin überprüft werden. In einem EMF-basierten Editor dienen diese Einschränkungen als Invarianten, die es einem Entwickler verwehren, dagegen zu verstoßen.

6.5.3 Diskussion

Anhand des in Kapitel 6.5.2 (S. 121) erläuterten Entwurfs des Aufgabenmodells lässt sich nun untersuchten, ob und durch welche Maßnahmen die an die Modellierung gestellten Anforderungen erfüllt werden.

(1) Die konkreten Konzepte des Aufgabenmodells wurden durch Analyse der initial unterstützten Prozessmodellierungsstandards unter Beachtung der Expressivität erhaltenden Einschränkungen extrahiert. Durch eine explizite Modellierung von

Verknüpfungen wie Kontroll- und Datenflüssen sowie Abhängigkeiten zwischen Aufgaben sowie eine feingranulare Modellierung von Daten kann die entsprechende Information zur Realisierung einer generischen Assistenzfunktionalität direkt über das Modell abgefragt werden.

(2) Eine feingranulare Modellierung von Daten wird wie beschrieben durch den Einsatz des Pattern-Metamodells der SiAM-Dialogplattform (Anhang C, S. 273) erreicht. Damit lassen sich zur Entwurfszeit Muster erwarteter Datenobjekte auf Basis eines zugrundeliegenden Domänenmodells beschreiben. Ein Datenobjekt spezifiziert zur Entwurfszeit zudem den Zeitpunkt des Vorhandenseins der Daten zur Laufzeit.

(3) Die Erweiterbarkeit des Aufgabenmodells ist durch seinen abstrakten Unterbau in Form von Nodes, FlowNodes und Edges gewährleistet und kann durch entsprechende Spezialisierung ohne Auswirkung auf existierende Konzepte leicht umgesetzt werden. Beispielsweise könnte so ein neues Konzept DataDependency analog zu TaskDependency von Edge abgeleitet werden.

(4) Eine konzeptionelle Nähe des Aufgabenmodells zu unterschiedlichen Prozessmodellierungsstandards wird durch die Punkte (1) bis (3) erreicht. Dies alleine ist aber noch kein Garant für eine einfache Umsetzung einer entsprechenden Modelltransformation. Dies wird erst durch eine möglichst *flache Modellierung* erreicht, die eine tiefe Verschachtelung von Transformationsregeln zur Abfrage und Erzeugung von Elementen von vornherein vermeidet. Konkret bedeutet dies, dass alle modellierten Aufgaben über eine UML-Kompositionsreferenz (im EMF-Jargon eine Containment-Referenz) im instanziierten TaskModel ohne Rücksicht auf etwaige Abhängigkeiten oder Aufgabenhierarchien aufgesammelt werden. Außerdem werden Kanten, also Verknüpfungen aller Art, expliziert als Konzepte modelliert. Dabei notwendige Referenzierungen werden in Form von einfachen UML-Assoziationen erstellt. Insgesamt wird so auch die Erstellung von Modelltransformationen ausgehend von einem Aufgabenmodell in nachgelagerte Modelle erleichtert.

(5) Ein von EMF anhand des Metamodells automatisch erzeugter Baumeditor für Aufgabenmodelle ist aufgrund der schlechten Darstellung von zeitlichen Abfolgen und Zusammenhängen eher ungeeignet. Die gerade in Punkt (4) angeführte flache Modellierung erschwert die Nutzung des Baumeditors zudem, weil hierin Aufgabenhierarchien nicht ersichtlich werden. Das Eclipse-Projekt Sirius[5] zur deklarativen Erstellung grafischer Editoren für EMF-basierte Metamodelle bietet durch eine Abbildungen semantischer Konzepte und Relationen auf grafische Darstellungen auch für ein flaches Metamodell eine günstige Ausgangsbasis zur einfachen Entwicklung umfangreicher grafischer Editoren, wie in Kapitel 8.6 (S. 196) für das Aufgabenmodell gezeigt.

(6) Durch die Ableitung des Task-Konzepts von IdentifiableEntity, kann ein eindeutiger

[5] siehe https://www.eclipse.org/sirius/ (letzter Zugriff: 04.11.2017)

Bezug von Aufgaben im Aufgabenmodell zu korrespondierenden Elementen in Kollaborationsprozessen über die Eigenschaften *id* und *name* hergestellt werden. Transformationen sorgen für eine Übertragung der entsprechenden Information ins Aufgabenmodell. Es ist jedoch Aufgabe der Laufzeitumgebung (Kapitel 9.4, S. 216), für eine entsprechende Synchronisierung zu sorgen.

(7) Die Zuweisung einer Rolle an eine Aufgabe zur Entwurfszeit ermöglicht die Instanziierung und rollenbasierte Zuweisung einer Aufgabe an einen konkreten Agenten zur Laufzeit. Die rollenbasierte Zuweisung kann zudem benötigte Fähigkeiten und relevante Kontextfaktoren bei der Auswahl des Agenten berücksichtigen.

6.6 Dialogaktmodell als abstraktes UI-Modell

SiAM-dp stellt bereits eine Basismodellierung von Dialogakten auf Grundlage des ISO-Dialogakt-Standards (ISO 2012) einschließlich einer Modellierung von kommunikativen Funktionen zur Verfügung. Dies geschieht in den Metamodellen *IO* und *Communicative Functions* (Abbildung 4.6). Die darin enthaltenen Konzepte werden hier wiederverwendet und entsprechend den Anforderungen bzgl. Nachvollziehbarkeit und Bedeutungserhaltung von Transformationen ergänzt. Die Beschreibung von Mustern erfolgt wieder auf Basis des Pattern-Metamodells (Anhang C, S. 273).

Das Dialogaktmodell in KomBInoS stellt in der Metamodellarchitektur das abstrakte UI-Modell dar. Abbildung 6.11 zeigt das entsprechende Metamodell. Ein Dialogaktmodell (DialogActModel) dient als Container-Element und enthält folglich alle Dialogakte, die zum Durchführen und Bearbeiten von Aufgaben eines Kollaborationsprozesses notwendig sind. Ein Dialogakt (DialogueAct) erweitert einen kommunikativen Akt (CommunicativeAct) neben einem deskriptiven Namen zur Anzeige in der Entwicklungsumgebung um Referenzen auf (1) ein Aufgabenmodell (*taskModelRef*), (2) eine Aufgabe innerhalb eines Aufgabenmodells (*taskRef*) sowie (3) einen dedizierten Bestandteil eines komplexeren Datenobjekts (*slotRef*). Ein Dialogakt kann entweder ein Eingabeakt (InputAct) oder ein Ausgabeakt (OutputAct) sein. Eine Spezialisierung von gleichnamigen Konzepten des SiAM-IO-Metamodells stellt sicher, dass entsprechende Instanzen aufgrund einer vorhandenen kommunikativen Funktion und eines semantischen Inhalts in der SiAM-Laufzeitumgebung verarbeitet werden können. Der semantische Inhalt (SemanticContent) von SiAM-dp wird um eine Referenz auf ein Muster zur Beschreibung von definiten Inhalten zur Entwurfszeit (*contentPattern*) ergänzt. Dem entgegen ermöglicht SiAM-dp die Beschreibung von Referenzen auf semantische Inhalte (ReferenceModel), die zur Laufzeit noch im Kontext (z. B. Zeit, Ort, Dialog, crossmodal) aufgelöst werden müssen. Dies geschieht ebenfalls durch die Spezifikation eines Musters. Bei der aufgabenorientierten Interaktion, insbesondere mit datenverarbeitenden Prozessen, stehen zusätzlich Domänen- und Datenobjekte im (je nach Aufgabe wechselnden) Fokus. Dementsprechend müssen semantische Inhalte auch im Kontext einer Aufgabe, also der der Aufgabe zugrundeliegenden oder erwarteten Information, interpretiert werden können. Mit Hilfe einer Referenz auf eine Wissensbasis (KnowledgeBaseReference)

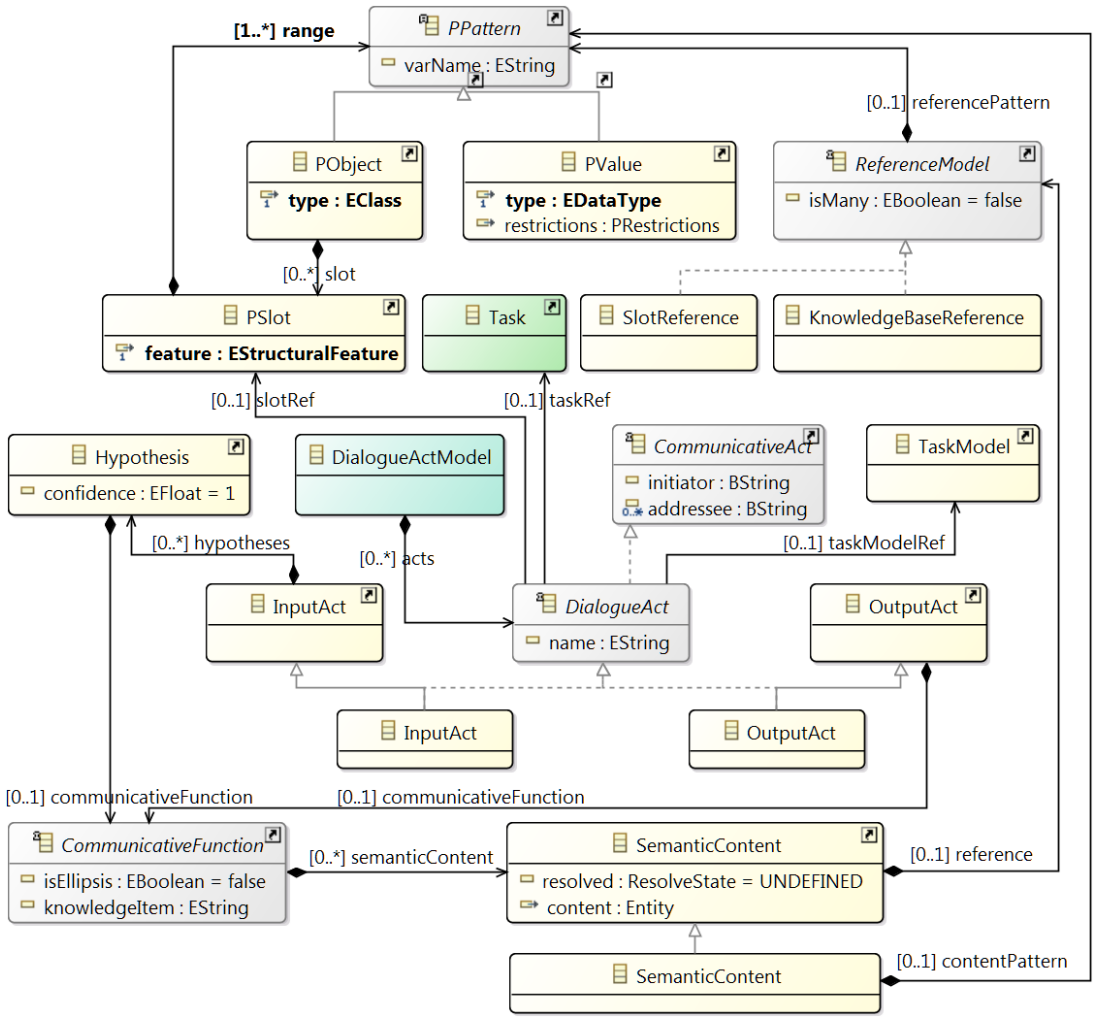


Abbildung 6.11: Das KomBInoS-Dialogaktmodell

kann definite aber unvollständige Information im Rahmen eines informationsliefernden Dialogakts (Kapitel 4.2.3, S. 52) mit zusätzlicher Information aus einer Wissensbasis angereichert werden. Umgekehrt spezifiziert eine Referenz auf einen Slot (Kapitel 4.3.4, S. 64) einer dialog- oder domänenspezifischen Datenstruktur (SlotReference), welche Information im Rahmen eines informationssuchenden Dialogakts aus dem Aufgabenkontext zu extrahieren ist.

Zur Interaktion im Rahmen von Kollaborationsprozessen sind jedoch weitere kommunikative Funktionen notwendig, die es einem Agenten ermöglichen, steuernd in die Aufgabenbearbeitung oder -zuweisung einzugreifen. Diese sind in Abbildung 6.12 dargestellt. Die Einfärbung der Konzepte in orange bedeutet, dass diese zusätzlich von der allgemeingültigen kommunikativen Funktion Request abgeleitet sind. Analog leiten

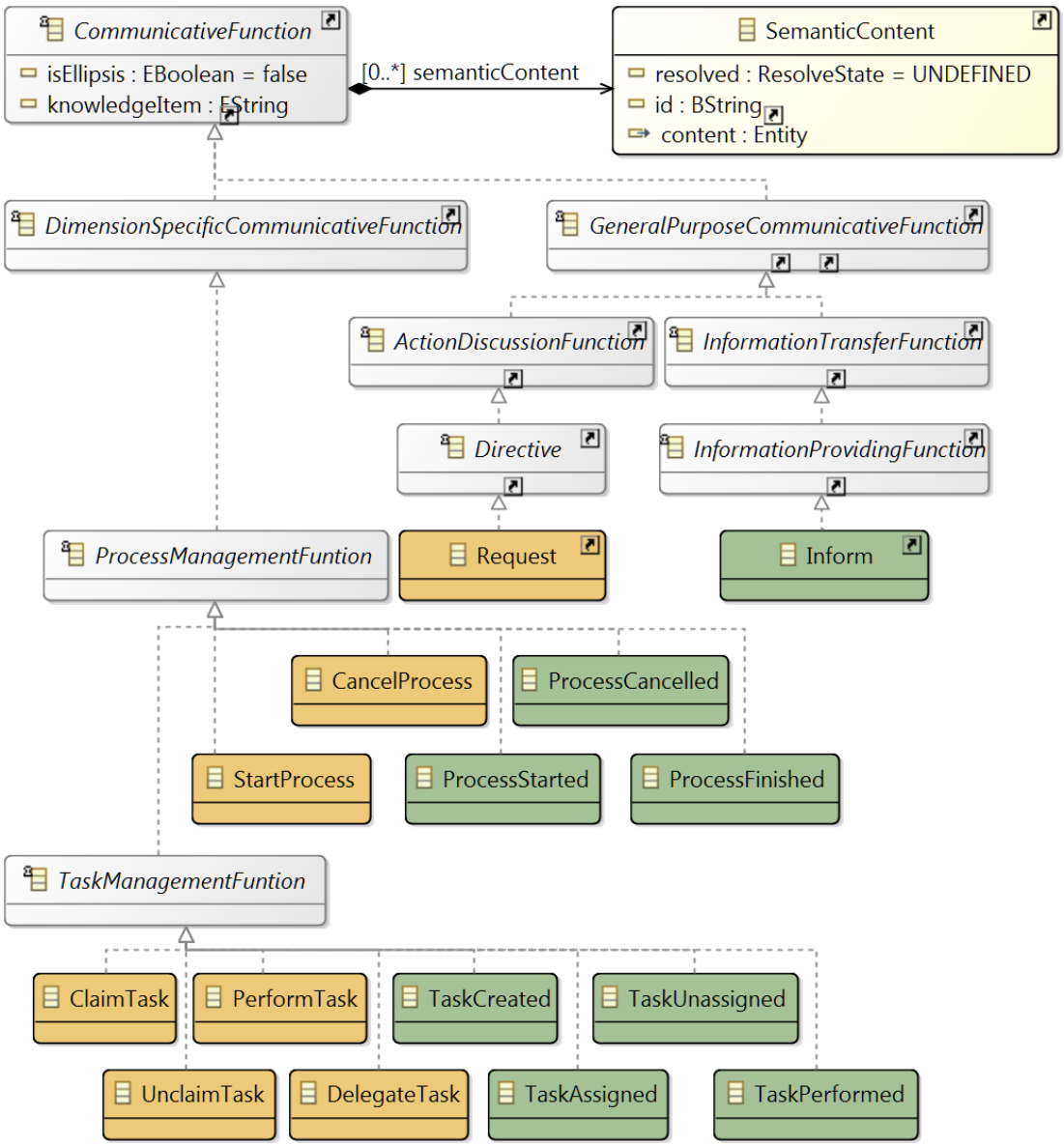


Abbildung 6.12: Kommunikative Funktionen zur Prozesssteuerung und Aufgabenbearbeitung in KomBInoS

sich Konzepte in grün von Inform ab. Auf eine Darstellung der Vererbungsbeziehung in UML-Notation wurde zugunsten einer besseren Lesbarkeit verzichtet. Die Erweiterung umfasst konkrete Prozessmanagementfunktionen (ProcessManagementFunction) zum Starten einer neuen Prozessinstanz (StartProcess) sowie zum Abbruch einer laufenden (CancelProcess). Außerdem existieren kommunikative Funktionen zur Benachrichtigung über relevante Statusänderungen einer Prozessinstanz (ProcessStarted, ProcessCancelled und ProcessFinished). Um welchen Prozess es sich dabei handelt, wird im semantischen Inhalt näher bestimmt. Neben Prozessmanagementfunktionen wurden spezialisierte kommunikative Funktionen zur Aufgabenverwaltung (TaskManagementFunction) abgebildet. Dazu zählen das Beanspruchen einer Aufgabenbearbeitung (ClaimTask), das Zurückgeben einer entsprechenden Verantwortlichkeit (UnclaimTask) sowie das Delegieren von Aufgaben an andere Agenten (DelegateTask). Unter Umständen, etwa bei manuellen Tätigkeiten, kann es nötig sein, dass ein Agent eine an ihn gestellte Aufgabe explizit als durchgeführt kennzeichnet (TaskPerformed). Mit dem Dialogsystem als Initiator eines entsprechenden Dialogakts dient die kommunikative Funktion TaskPerformed ebenso wie TaskCreated, TaskAssigned und TaskCompleted zur Kommunikation von Statusänderungen auf Aufgabenebene.

Aus Gründen der Einfachheit werden kommunikative Funktionen zum Prozessmanagement als eigenständige Dimension (DimensionSpecificCommunicativeFunction) betrachtet, auch wenn dies nach Bunt u. a. (2010, S. 4) keine eigenständige Facette im Sinne des Dialogaktstandards ist. Bezogen auf die Modellierung stellt die Mehrfachvererbung der konkreten kommunikativen Funktionen eine Strukturierungshilfe dar, welches eine zusätzliche Perspektive auf diese Funktionen bzw. Konzepte ermöglicht.

6.7 Konkrete UI-Modelle

Mit einer modalitätsspezifischen Konkretisierung von UI-Modellen rückt die Verarbeitung dieser Modelle zur Laufzeit, etwa durch eine Codegenerierung oder Interpretation immer stärker in den Vordergrund. So beinhaltet die SiAM-Dialogplattform Laufzeitkomponenten, die aus semantisch annotierten Spracheingabemodellen unmittelbar W3C-standardisierte Grammatiken gemäß der Speech Recognition Grammar Specification (SRGS) (Hunt und McGlashan 2004) erzeugen oder aus regelbasierten Vorlagen zur Sprachgenerierung Syntheseaufträge in der Speech Synthesis Markup Language (SSML) (Burnett u. a. 2004) erstellen. Auf technischer Ebene gilt es, diese Komponenten in der erweiterten KomBInoS-Laufzeitumgebung auf Basis der SiAM-Dialogplattform weiterzuverwenden. Gleichzeitig muss dafür Sorge getragen werden, dass die Nachvollziehbarkeit von Zusammenhängen und Abhängigkeiten sowie die Rückverfolgung von Artefakten während und nach dem Generierungs- und Kompositionsprozess entlang der Modellarchitektur auf Modellebene gewährleistet sind – ein Aspekt, der in SiAM-dp bislang nicht im Fokus stand. Diese Anforderungen lassen sich für das Spracheingabe- und das Sprachausgabe-Modell mit gezielten Erweiterungen der entsprechenden SiAM-Basismodelle umsetzen, ohne diese selbst anpassen zu müssen. Das GUI-Modell wurde

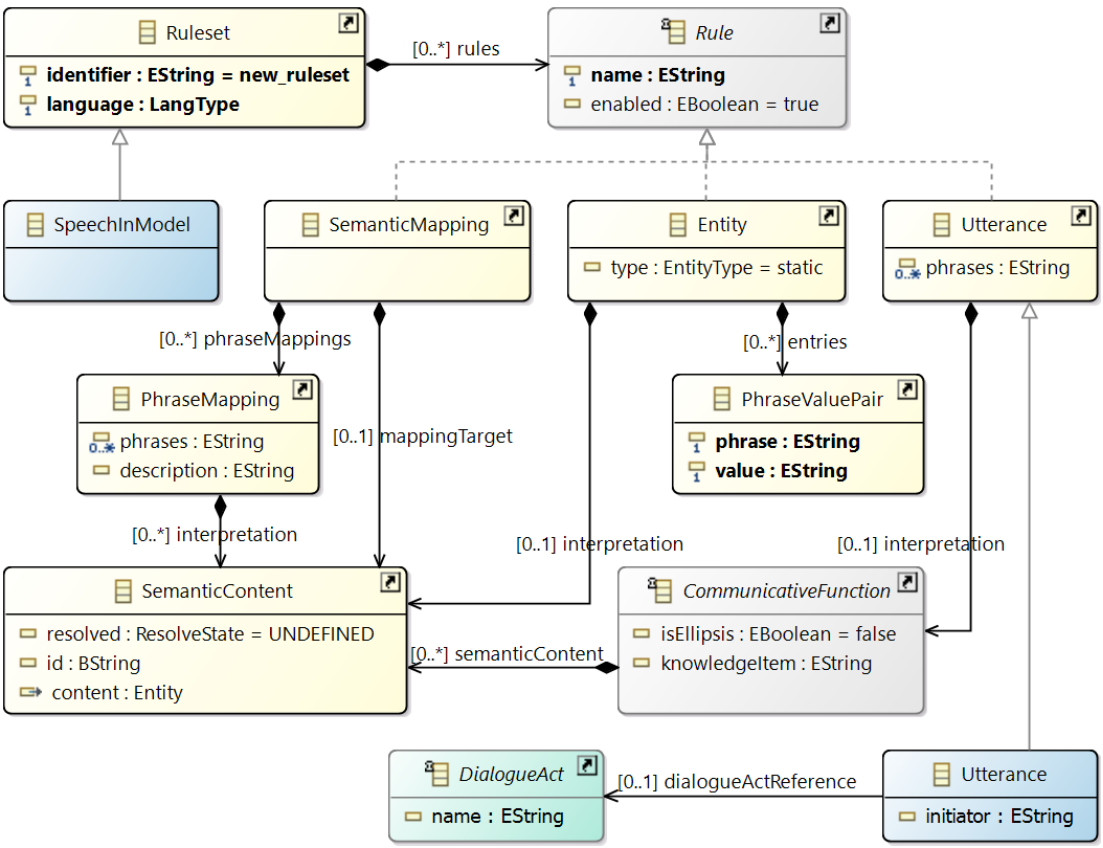


Abbildung 6.13: Das KomBInoS-Spracheingabemodell

in KomBInoS unter Berücksichtigung von Konzepten in der SiAM-Entsprechung von Grund auf neu entwickelt. Außerdem wurden Modelle zur Abbildung und Ausnutzung paralinguistischer Kommunikationskanäle in Form eines Gesteneingabe-Modells sowie eines Modells zur Animationssteuerung eines virtuellen Charakters mit Hilfe von SiAM-Basismodellen umgesetzt.

6.7.1 Spracheingabe-Modell

Das Spracheingabemodell in KomBInoS ist in Abbildung 6.13 dargestellt. Das ursprüngliche SiAM-Basismodell zur Spezifikation von Nutzeräußerungen fasst Regeln (Rule) in einer Regelmenge (Ruleset) zusammen. Es gibt drei Arten von Regeln. Vollumfängliche Äußerungen eines Agenten (Utterance) spezifizieren eine Menge von zeichenkettenbasierten Satzmustern in erweiterter Backus-Naur-Form. Diese Phrasen können textuelle Referenzen auf untergeordnete Regeln besitzen, indem der Name einer solchen Regel in Großbuchstaben in Verbindung mit einem vorangestellten Dollar-Zeichen ($NAME) verwendet wird. Des Weiteren spezifiziert eine Äußerung durch eine kommunikativen

Funktion samt semantischem Inhalt, wie sie zu interpretieren ist. Dieser Inhalt kann durch Unterregeln unifikationsbasiert sukzessive aufgebaut werden. Anhang B (S. 271) erläutert Unifikation und Overlay auf Ecore-basierten Metamodellen. Tätigt ein Agent zur Laufzeit eine entsprechende Äußerung und wird diese vom Adressaten in Form eines Dialogsystems mit Hilfe eines Spracherkenners erkannt, so wird im Interpretationsschritt ein Eingabeakt (InputAct, Abbildung 6.11) zur weiteren Verarbeitung erstellt, welcher in einer Hypothese die kommunikative Funktion mit errechnetem semantischen Inhalt enthält. Nachgelagerte semantische Abbildungsregeln (SemanticMapping) bieten einen Mechanismus zur besseren Strukturierung des Spracheingabemodells und zur Aggregation von semantischen Inhalten gleichen Typs (SemanticContent), welche durch unterschiedliche Phrasen erzeugt werden (PhraseMappings). Dieser Mechanismus erlaubt auch die Hinterlegung von Metadaten jenseits des semantischen Inhalts, etwa hinsichtlich des Grades der Höflichkeit der (Teil-) Äußerung. Nachgelagerte Entitätenregeln (Entity) dienen zum Aufzählen von Entitäten gleichen Typs über einen Namen, z. B. Städte oder Personen. Zur Gewährleistung der Rückverfolgbarkeit generierter Spracheingaberegeln im Rahmen des modellgetriebenen Entwicklungsansatzes erfolgte eine Verfeinerung der SiAM-Äußerung (Utterance in gelb). Die abgeleitete KomBInoS-Äußerung (Utterance in blau) enthält eine zusätzliche Referenz auf den Dialogakt, welcher durch die Äußerung dargestellt wird. Erst durch Ausnutzung dieser bislang nicht verfügbaren Zusatzinformation, die während des Generierungsprozesses automatisch abfällt, wird es möglich, rollen- und aufgabenspezifische Grammatiken zur Laufzeit bereitzustellen.

6.7.2 Sprachausgabe-Modell

Die Modellierung von Äußerungen des Systems erfolgt ebenfalls regelbasiert. Das entsprechende Sprachausgabemodell, welches in Abbildung 6.14 dargestellt ist, stützt sich dabei auf SiAM-Konzepte zur Modellierung von Abbildungsregeln (MappingRule), deren Bedingung durch ein Muster (PPattern) eines Ausgabeaktes spezifiziert wird. Teile bzw. Unterstrukturen eines Musters können durch die Vergabe eines Namens referenzierbar gemacht werden.

Aufgrund der Vererbungshierarchie enthält das Sprachausgabemodell (SpeechOutModel) eine Menge von Schablonen zur Sprachsynthese (SynthesisTemplate). Eine Syntheseschablone ist eine Ausgabeabbildungsregel (OutputMappingRule) und erzeugt als solche eine Ausgaberepräsentation (OutputRepresentation) in Form einer Sprachsyntheseanweisung (SpeechSynthesis in gelb), welche aufgrund der KomBInoS-spezifischen Erweiterung (SpeechSynthesis in blau) ein komplexes SSML-Dokument (SSML) fassen kann. Das in Abbildung 6.14 ausschnittsweise gezeigte SSML-Metamodell wurde im Rahmen der Arbeit ebenfalls auf Basis des normativen XML-Schemas[6] in Ecore teilautomatisch erstellt. Ein SSML-Dokument spezifiziert laut Standard einen Sprachsynthesebefehl (Speak), welcher wiederum verschachtelte Anweisungen enthalten kann,

[6] siehe http://www.w3.org/TR/speech-synthesis/synthesis.xsd (letzter Zugriff: 04.11.2017)

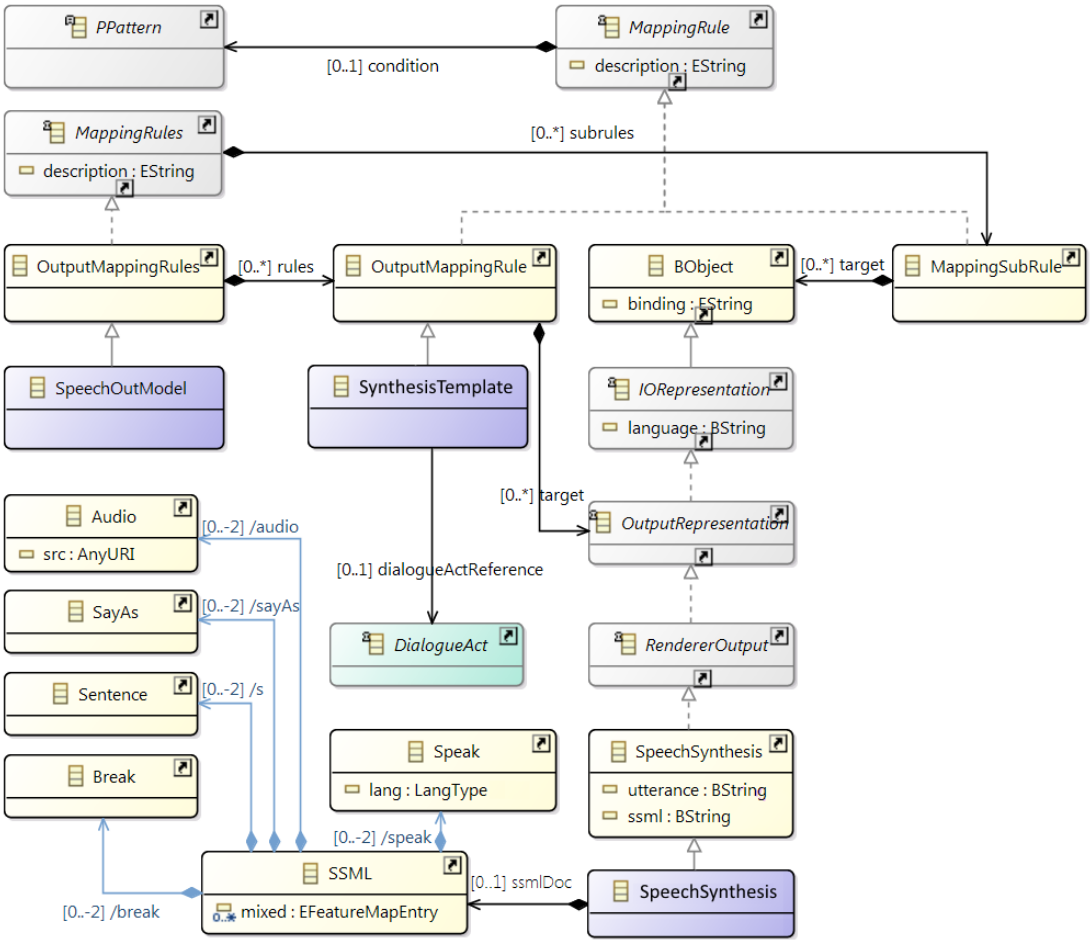


Abbildung 6.14: Das KomBInoS-Sprachausgabemodell

etwa zur Satzstrukturierung (Sentence) oder zur Aussprache (SayAs) einzelner Teile. Durch ebenfalls im Sprachausgabemodell enthaltene nachgelagerte Abbildungsregeln (MappingSubRule) kann das SSML-Dokument sukzessive aufgebaut werden. Dazu werden für über Namen referenzierbare Unterstrukturen des eingangs erkannten Musters weitere Regelanwendungen angestoßen. So wird die Wiederverwendung einzelner Fragmente bzw. Teilbäume eines SSML-Dokuments durch die Wiederverwendung der entsprechenden Regelmenge ermöglicht. Eine Syntheseschablone in KomBInoS stellt zur Rückverfolgbarkeit ebenfalls eine Verknüpfung zum Ausgabeakt her, der für die Erzeugung der Schablone ursächlich war.

6.7.3 GUI-Modell

Eine grafische Benutzerschnittstelle dient nach Garrett (2010) (Kapitel 5.2.1, S. 72) gleichermaßen als Informations- und Bedienschnittstelle. Ein entsprechend umfassendes Modell muss daher sowohl Konzepte zur grafischen Präsentation von Information als auch Konzepte zur Modellierung von Interaktion beinhalten. Dies führt oft dazu, dass eine entsprechende Interaktionslogik direkt in die Präsentation als Code eingebettet ist, etwa zur Ereignisbehandlung. In der Folge wird die Testbarkeit und Wartbarkeit der GUI erschwert. Das Architekturmuster Model-View-Controller (MVC) schafft hier durch eine klare Trennung von Verantwortlichkeiten eine Verbesserung. Das Prinzip ist in Abbildung 6.15 (links) dargestellt.

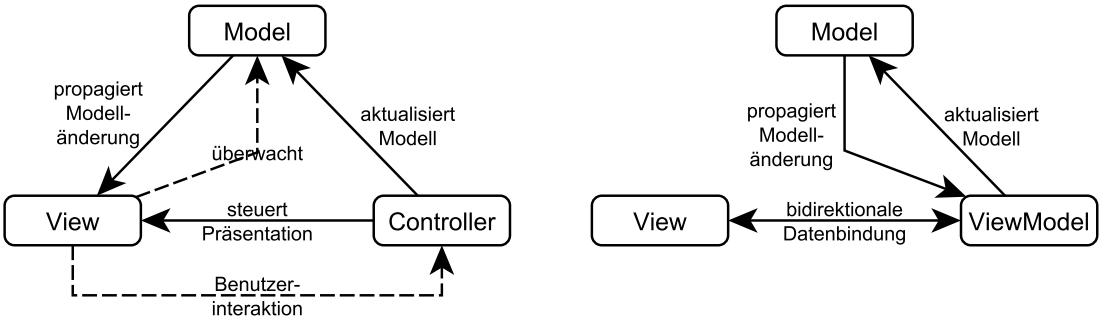


Abbildung 6.15: Vergleich der Architekturmuster MVC (links) und MVVM (rechts)

Die Präsentation (View) leite jede Form von Benutzerinteraktion zum assoziierten Controller weiter, welcher daraufhin die Präsentation steuern und das Datenmodell aktualisieren kann. Das Modell wiederum notifiziert die Präsentation über Änderungen, welche ihrerseits die aktualisierten Daten vom Modell abfragt und anzeigt. Hierbei ist die Präsentation sowohl mit dem Controller als auch dem Modell fest verbunden.

Als neuere Variante von MVC erreicht das Architekturmuster Model-View-ViewModel (MVVM) in Abbildung 6.15 (rechts) durch einen Mechanismus zur bidirektionalen Datenbindung nun einerseits eine Abkopplung der Präsentation vom Modell sowie eine lose Kopplung zwischen Präsentation und Controller. In diesem Kontext wird der Controller dann auch als Präsentationsmodell (ViewModel) bezeichnet, da er alle für die Präsentation notwendigen Daten und Methoden an zentraler Stelle kapselt. Auf der einen Seite kann die Präsentation somit rein deklarativ ohne eingebetteten Code beschrieben werden. Zudem ist sie leicht austauschbar. Auf der anderen Seite lässt sich das Präsentationsmodell aufgrund der vollständig enthaltenen Präsentationslogik sehr gut ohne Präsentation automatisiert testen. Damit kann auch die Entwicklung einer GUI auf unterschiedlich spezialisierte Design- und Softwareentwicklungsteams aufgeteilt werden. Kritische Komponente zum reibungslosen Zusammenspiel zwischen Präsentation und Präsentationsmodell bleibt hierbei auf technischer Ebene ein verlässliches Rahmenwerk zur bidirektionalen Datenbindung.

Die vorangegangenen Überlegungen führen im Ergebnis zur Entscheidung, ein eigenes GUI-Modell für KomBInoS zu entwickeln, welches nicht nur grafische Präsentationsaspekte und Ereignisbehandlung berücksichtigt, sondern auch eine Aufteilung einer GUI nach dem MVVM-Prinzip vornimmt. Dies kann die Wiederverwendbarkeit einzelner Modellfragmente im Rahmen einer Komposition erhöhen, u. a. weil das Präsentationsmodell auf logischer Ebene als abstraktes GUI-Modell angesehen werden kann. Trotzdem sind konkrete GUI-Konzepte wie Button oder Label aus dem SiAM-GUI-Modell auch im KomBInoS-GUI-Modell vertreten, sodass beide Modelle auf Blattebene kongruent sind. Auch wenn das KomBInoS-GUI-Modell einem Architekturmuster folgt, so wird darin keine konkrete Zielarchitektur (z. B. die Windows Presentation Foundation oder das JavaScript-basierte Rahmenwerk KnockoutJS) vorgegeben. Dies geschieht erst im Rahmen der anschließenden M2C-Transformation. Abbildung 6.16 zeigt das KomBInoS-GUI-Modell.

Für jede Rolle im Ressourcenmodell ist eine eigenständige grafische Benutzerschnittstelle (GuiApplication) vorgesehen. Diese werden im GUI-Modell (GuiModel) aufgesammelt. Eine rollenspezifische GUI enthält eine Menge von Präsentationen (View) und Präsentationsmodellen (ViewModel). Ein Präsentationsmodell kann eine rollenspezifische Aufgabe (Task) referenzieren und setzt sich aus spezifischen Elementen (ViewModelElement) zusammen. Eigenschafen (ViewModelProperty) können auf einen Slot (PSlot) eines Datenelements (Data) referenzieren. Handelt es sich dabei um eine von einem Agenten preiszugebende Information, so kann über die *onChange*-Referenz ein Dialogakt (DialogueAct) angegeben werden, der die Information bei Änderung des entsprechenden Wertes in der GUI zum Dialogsystem übermittelt. Methoden innerhalb eines Präsentationsmodells (ViewModelMethod dienen als Einstiegspunkt zur Ausführung von beliebig komplexer Präsentationssteuerungslogik (z. B. Navigationsanweisungen oder das Filtern einer angezeigten Liste) oder Dienstaufrufen an ein Dienste-Back-End. Konkret kann hierüber die Ausführung von Prozessmanagementfunktionen (Abbildung 6.12) aus der GUI heraus abgebildet werden. Dazu kann ein entsprechender Dialogakt mit Hilfe der *onInvoke*-Referenz spezifiziert werden.

Als ein Element der grafischen Benutzerschnittstelle (GuiElement) kann eine Präsentation ein Präsentationsmodell gemäß dem MVVM-Architekturmuster referenzieren. Dessen Kindelemente (z. B. Label, Image, etc.) können wiederum über Datenbindungen (DataBinding) mit Elementen dieses Präsentationsmodells verknüpft werden. Diese Einschränkung ist über einen entsprechenden OCL-Ausdruck analog zu Abbildung 6.10 sichergestellt. Welche Eigenschaft eines grafischen Elements an ein Element eines Präsentationsmodells gebunden wird, lässt sich durch eine Aufzählung (DataBindingTypeEnum) näher bestimmen. Während die ersten drei Literale (*text*, *visible*, *style*) das Aussehen und den anzuzeigenden Text kontrollieren, dienen die restlichen Literale (*click* bis *selectedOption*) zum Steuern von Formularen und zur Dateneingabe. Nicht dargestellt sind Literale zur bedingten Manipulation der Hierarchie und zur Replikation von grafischen Elementen. Durch die explizite Datenbindung ist auf semantischer Ebene bekannt, welches grafische Element welche Information enthält und gegebenenfalls manipulieren

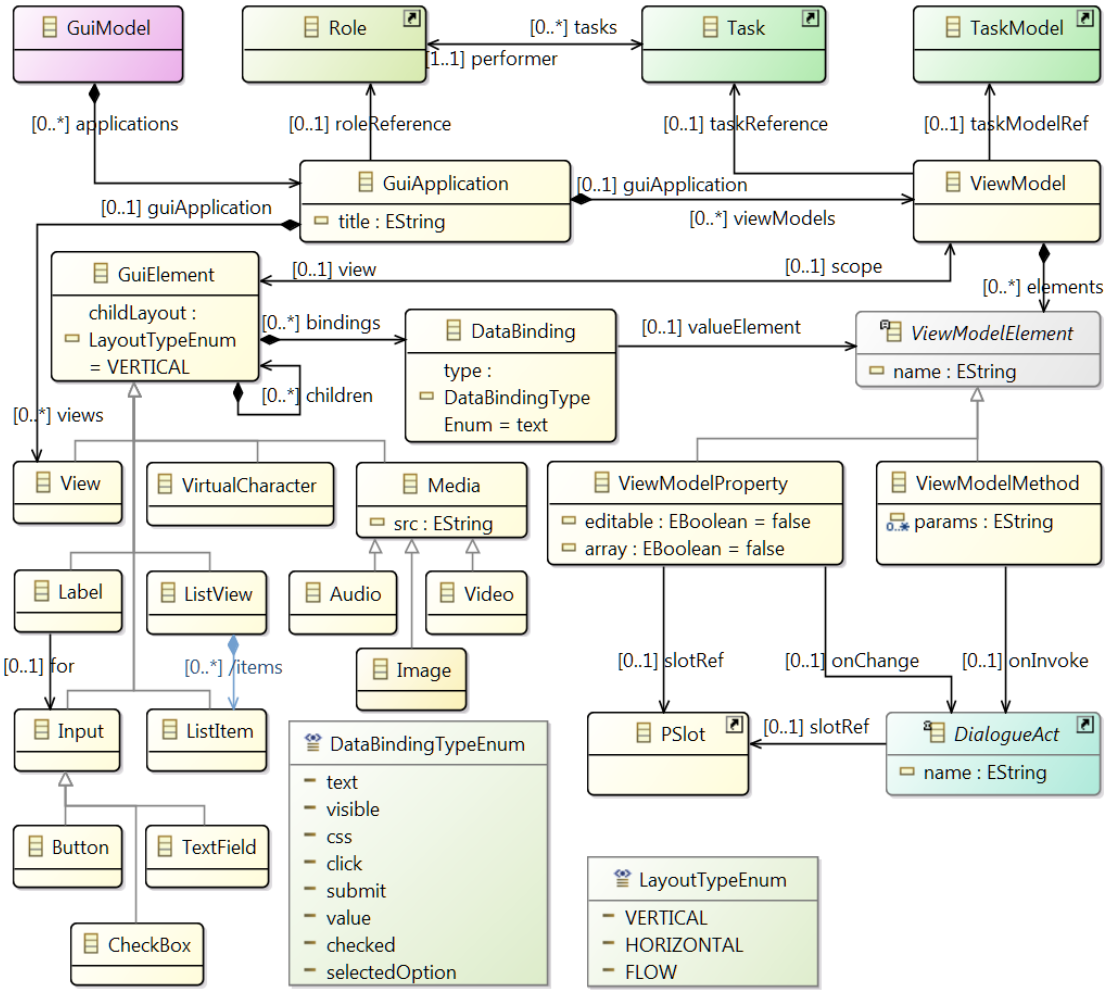


Abbildung 6.16: Das KomBInoS-GUI-Modell

kann.

Ebenfalls kann in einem Baum von grafischen Elementen eine Hierarchie von unterschiedlichen Präsentationsmodellen aufgebaut werden, indem unterhalb einer Präsentation weitere Präsentationsmodelle über die *scope*-Referenz mit grafischen Elementen verknüpft werden. So muss eine Präsentation nun nicht genau einer Aufgabe des Aufgabenmodells zugeordnet werden. Die Vererbungshierarchie von GUI-Elementen ist auf der linken Seite von Abbildung 6.16 ausschnittsweise dargestellt und orientiert sich an Standard-GUI-Modellen wie XAML, HTML und dem SiAM-GUI-Modell.

Es existieren Konzepte zur multimedialen Präsentation und Abfrage von Information, die in der finalen GUI auf Basis des aktuellen (Geräte-) Kontextes durch unterschiedliche Präsentations- und Interaktionsmetaphern optimal dargestellt werden können. Beispielsweise wird eine Liste von Einträgen durch eine Listenansicht (ListView)

repräsentiert, welche zur Laufzeit eine Menge von Listeneinträgen (ListItem) enthält. Die Definition eines Listeneintrags zur Entwurfszeit erfolgt über eine Schablone, welche durch eine foreach-Datenbindung an eine listenartige Eigenschaft des korrespondierenden Präsentationsmodells zur Laufzeit entsprechend oft mit entsprechenden Inhalten repliziert wird. Die Darstellung der Listenansicht zur Laufzeit kann etwa in Form einer Tabelle oder eines Karussells erfolgen. Unter Angabe eines Namens kann ein virtueller Charakter (VirtualCharacter) in die grafische Präsentation eingebunden werden. Dessen Steuerung aus der Dialogplattform heraus wird in Form des im nächsten Abschnitt diskutierten Modells zur Animationssteuerung für virtuelle Charaktere spezifiziert.

6.7.4 Weitere paralinguistische Ein- und Ausgabemodelle

Die im Folgenden vorgestellten Modelle machen sich analog zum Sprachausgabemodell (Kapitel 6.7.2, S. 131) den generischen SiAM-Mechanismus zur regelbasierten Ausgabesteuerung aber auch zur Eingabeinterpretation zunutze. Darüber hinaus können auf die gleiche Weise weitere paralinguistische Ein- und Ausgabemodelle in KomBInoS zur Unterstützung einer massiven Multimodalität im Rahmen eines modellgetriebenen Ansatzes hinzugefügt werden, etwa Modelle zur Verarbeitung von Mimik oder Augenbewegungen oder zur Steuerung anderer (grafischer) Präsentationsformen, z. B. zur Navigation in einem virtuellen dreidimensionalen Raum (Kapitel 10.6, S. 243).

Abbildung 6.17 zeigt das KomBInoS-Modell zur Animationssteuerung einen virtuellen Charakters (CharacterModel). Dieses umfasst eine Menge von Animationsregeln (CharacterAnimationRule). Eine solche Regel spezifiziert einen Ausgabeakt des Dialogsystems in Form eines Musters als Bedingung sowie eine bei Erfüllung der Bedingung durchzuführenden Animation des virtuellen Charakters. Die Animation muss der Definition des virtuellen Charakters im GUI-Modell bekannt sein.

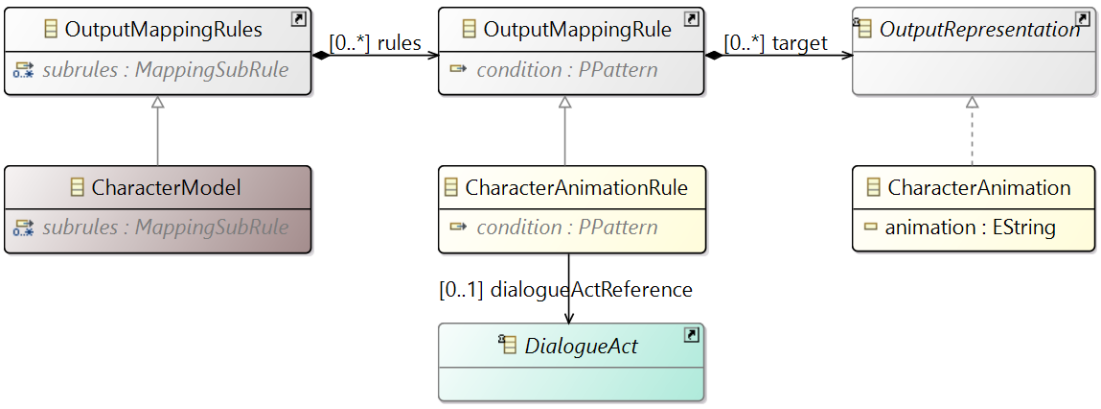


Abbildung 6.17: KomBInoS-Modell zur Animationssteuerungsmodell einen virtuellen Charakters

Das Gesteninterpretationsmodell (GestureModel) in KomBInoS, welches in Abbildung

6.18 gezeigt wird, umfasst eine Menge von Gesteninterpretationen (GestureInterpretation). Eine solche Interpretation beschreibt zur Entwicklungszeit einen Eingabeakt eines Gestenerkenners durch ein Muster und bildet dieses auf eine Gestenhypothese (GestureHypothesis) ab, welche die mit der Geste verknüpfte kommunikative Funktion transportiert.

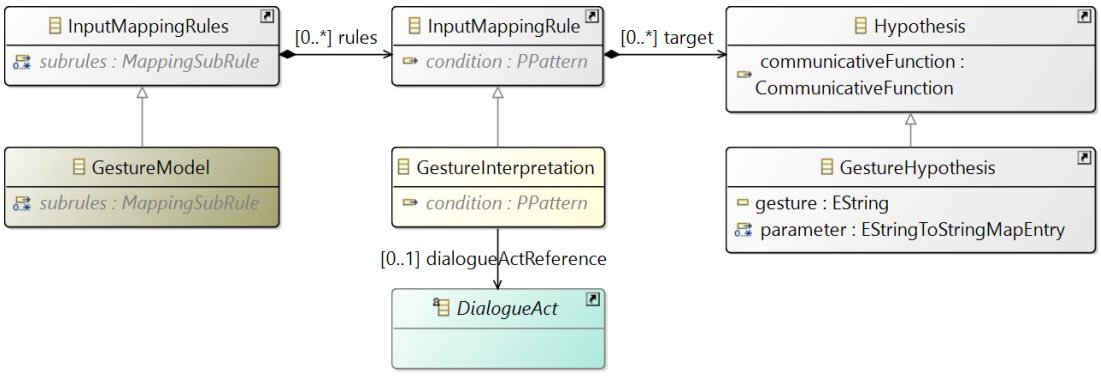


Abbildung 6.18: Das Gesteninterpretationsmodell in KomBInoS

6.8 Kompositionsmodell

Nachdem nun alle Metamodelle zur Beschreibung einer KomBInoS-UI vorgestellt wurden, zielt dieses Unterkapitel darauf ab, ein Metamodell zu spezifizieren, welches Modellfragmente unterschiedlicher KomBInoS-UIs in Beziehung setzt, um damit eine Komposition von Benutzerschnittstellen zu beschreiben. Die Notwendigkeit zur Komposition solcher Benutzerschnittstellen hat ihren Ursprung in einer vorangegangenen Komposition, genauer gesagt einer Orchestrierung, der zugrundeliegenden Dienste im Rahmen eines Kollaborationsprozesses. Eine genauere Betrachtung liefert Anforderungen zum Entwurf des Metamodells.

6.8.1 Anforderungen

Die Komplexität und der Grad einer Wiederverwendung von Modellfragmenten im Rahmen einer UI-Komposition unterscheidet sich in Abhängigkeit von der Tiefe der zuvor durchgeführten Dienstorchestrierung. Abbildung 6.19 teilt diese in drei Komplexitätsstufen ein. Hierbei ist zu bedenken, dass sich ein Dienst aufgrund der allgemeinen Kompositionsfähigkeit intern selbst aus einer Kompositionskette oder Orchestrierung zusammensetzen kann.

(1) **Ein existierender Dienst bzw. dessen Kollaborationsprozess wird im Rahmen einer neuen Orchestrierung vollständig wiederverwendet.** Dies kann technisch z. B. durch eine BPMN-CallActivity, einen CMMN-CaseTask oder

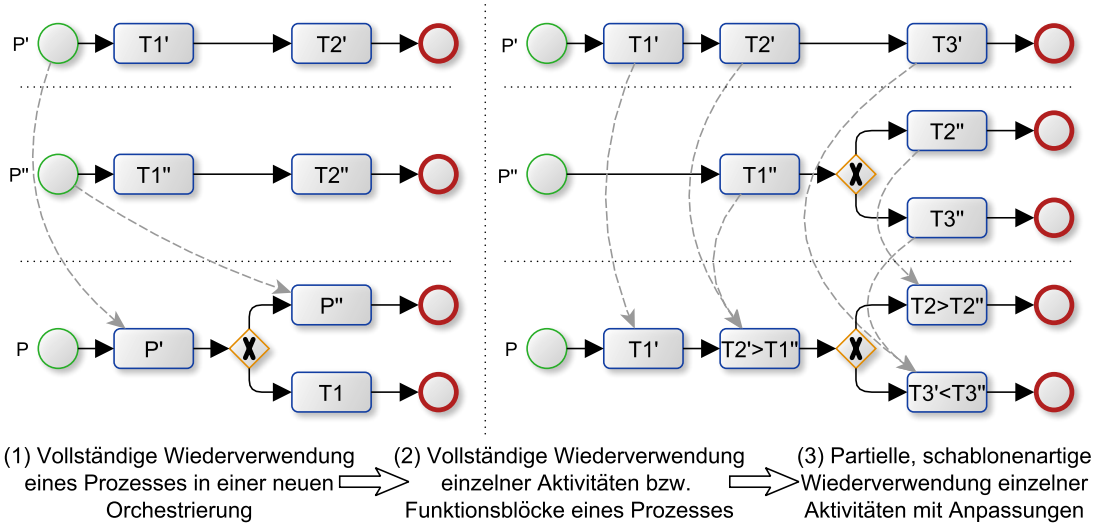


Abbildung 6.19: Kompositionskomplexität prozessorientierter Dienste am Beispiel von niedrig (links) bis hoch (rechts)

CMMN-ProcessTask durch Angabe einer Prozess-Id indirekt realisiert werden. Die Orchestrierung kann außerdem neue Aktivitäten enthalten. In der Praxis können orchestrierte Dienste unterschiedliche aber gegebenenfalls ähnliche Domänenmodelle zur internen Datenmodellierung verwenden. Für nachgelagerte Modelle wie das Aufgabenmodell können Abbildungsvorschriften zur Datentransformation (DataTransformation) im Datenfluss (DataFlow) des Aufgabenmodells referenziert werden. Mit Hilfe von Schema- bzw. Ontologie-Matching-Ansätzen (Euzenat und Shvaiko 2013) können diese Abbildungsvorschriften aufgrund des gemeinsam genutzten abstrakten KomBInoS-Domänenmodells weitgehend, etwa mit Werkzeugen wie COMA (Do 2012), automatisiert erstellt werden.

(2) **Einzelne Aktivitäten eines orchestrierten Dienstes werden vollständig in einem neuen Dienst orchestriert.** Voraussetzung hierfür ist der Zugriff auf den internen Aufbau des Dienstes in Form des implementierten Kollaborationsprozesses. Eine ausführbare technische Umsetzung kann mit den betrachteten Prozessmodellierungsstandards zwar nur durch eine Kopie der Aktivität innerhalb der Orchestrierung erfolgen. Durch einen expliziten Hinweis im Kompositionsmodell ist es jedoch während der UI-Erstellung möglich, die zugehörigen UI-Artefakte zu identifizieren und im aktuellen Kontext wiederzuverwenden. Alternativ könnte eine Annotation direkt im Kollaborationsprozess mit Hilfe der durch den verwendeten Standard bereitgestellten Sprachmittel erfolgen. Die Erfassung kompositionsbezogener Metadaten in einem zentralen Modell ist aber aufgrund der besseren Wartbarkeit und Übersicht zu bevorzugen.

(3) **Zusammenhängende Teile oder einzelne Aktivitäten eines existierenden**

Prozesses werden in einer Orchestrierung nach einer Adaption wiederverwendet. In Verbindung mit dem vorangegangenen Punkt kann sich so auf Basis eines gemeinsamen abstrakten Domänenmodells eine Vorlagenbibliothek von oft angewandten Kollaborationsmustern und assoziierten Benutzerschnittstellen etablieren, die speziell zur einfachen Orchestrierung und weniger zur direkten Verwendung vorgesehen sind. Ein solches Bündel aus abstraktem Referenzprozess und Benutzerschnittstelle kann als Schablone aufgefasst werden, welches einen weiteren Prozess samt Benutzerschnittstelle strukturell und inhaltlich um ein bestimmtes Muster ergänzt.

Abbildung 6.19, links, zeigt beispielhaft in BPMN-Notation, wie die Dienste bzw. Prozesse P' und P" als einzelne Aktivitäten vollständig im neuen Prozess P gekapselt werden. Auf der rechten Seite wird das konzeptuelle Vorgehen der Punkte (2) und (3) gezeigt. P' soll mit Hilfe von P" zu P erweitert werden. Dazu wird die Aktivität T1' in P unverändert wiederverwendet. Anhand einer Kompatibilitätsprüfung der Domänenmodelle von P' und P", etwa auf Basis eines Unifikationsansatzes, wird festgestellt, dass T2' T1" vollständig subsumiert. Bei T3' und T3" ist es genau anders herum. T2" stellt zusammen mit dem Gateway die eigentliche Erweiterung von P' dar. T2" hat also keine Entsprechung in P', muss aber gegebenenfalls zur Verwendung in P noch angepasst werden.

Abbildung 6.20 zeigt das geschilderte Vorgehen am konkreten Beispiel.

Stark wachsende Unternehmen müssen dafür sorgen, dass ihre Prozesse ebenso mitwachsen. Mit Hilfe einer Vorlagenbibliothek, wie oben geschildert, kann ein erfolgreiches Unternehmen etwa seinen Nachbestellprozess von Waren auch in Bezug auf die Benutzerschnittstelle schnell um ein 4-Augen-Prinzip erweitern, um den Anforderungen seines starken Wachstums und den damit verbundenen notwendigen organisatorischen Veränderungen Rechnung zu tragen. Dieses Beispiel wird im Software-Campus-Demonstrator in Kapitel 10.9 (S. 249) wieder aufgegriffen.

6.8.2 Entwurf

Das Kompositionsmodell ist in Abbildung 6.21 dargestellt. Es fügt sich nahtlos in die in Abbildung 6.1 gezeigte Basismodellierung einer multimodalen Dialogschnittstelle für einen Smart Service ein. Die darin benutzte Farbkodierung findet zur leichteren Zuordnung dargestellter Konzepte auch hier Anwendung.

Ein Kompositionsmodell (CompositionModel) verknüpft Elemente bestehender Kollaborationsprozesse sowie zugehöriger multimodaler Dialogschnittstellen miteinander. Dies geschieht durch einzelne Kompositionen (Composition). Spezialisierungen von ProcessComposition realisieren die expliziten Orchestrierungen von Diensten und Prozessen gemäß der in den Anforderungen geschilderten Komplexitätsstufen einer Dienstorchestrierung. Die Konzepte BpmnProcessComposition, CmmnProcessComposition und CmmnCaseComposition ermöglichen die Wiederverwendung vollständiger Prozesse in einer neuen Orchestrierung. CMMN unterschiedet hierbei außerdem zwischen stark

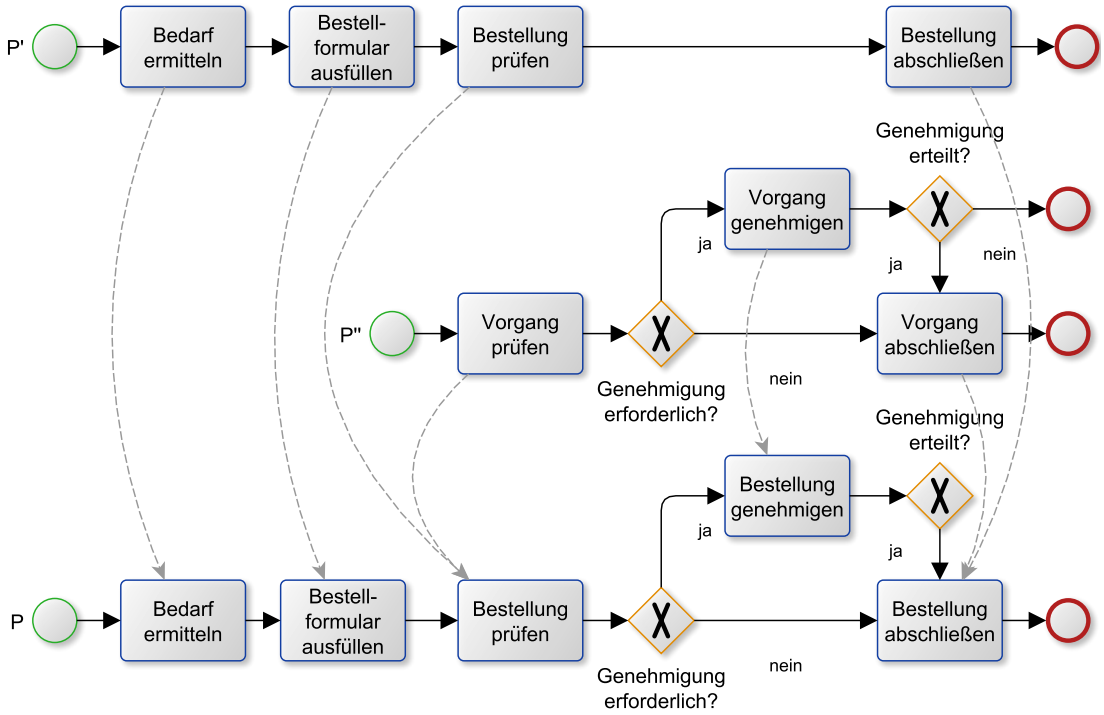


Abbildung 6.20: Beispielhafte Erweiterung eines Bestellprozesses (P') um ein 4-Augen-Prinzip (P")

strukturierten Prozessen und schwach strukturierten Fällen. BpmnTaskComposition und CmmnTaskComposition verknüpfen ein oder mehrere Aufgaben existierender Prozesse mit einer neuen Aufgabe des orchestrierenden Prozesses. Über den Kompositionstyp (CompositionType) wird gesteuert, ob ein vollständiger Prozess wiederverwendet (PROCESS-INSERTED) wurde oder aber eine einzelne Aufgabe unverändert (ACTIVITY-INSERTED) oder abgewandelt (ACTIVITY-ADAPTED) übernommen wurde. Werden mehrere Aufgaben auf eine Aufgabe abgebildet, so gelingt dies nur durch eine Verschmelzung (ACTIVITIES-MERGED). In Abhängigkeit von Kompositionen auf Prozessebene werden durch Aufgabenkompositionen (TaskCompositions) auch Aufgabenmodelle gänzlich oder in Teilen wiederverwendet. Ein vollständig wiederzuverwendendes Aufgabenmodell kann zusätzlich auf eine zusammengesetzte Aufgabe (CompositeTask) abgebildet werden. Die Komposition auf Ebene der MVVM-basierten GUI (GuiComposition) unterscheidet zwischen einer Komposition der Präsentationsmodelle (ViewModelCompo- sition) und einzelner Bestandteile einer grafischen Präsentation (GuiElementComposition). Wie in Kapitel 7.2 (S. 143) geschildert, benötigt die KomBInoS-Methode keine formale Abbildung einer Komposition von Dialogakt- und Sprachmodellen.

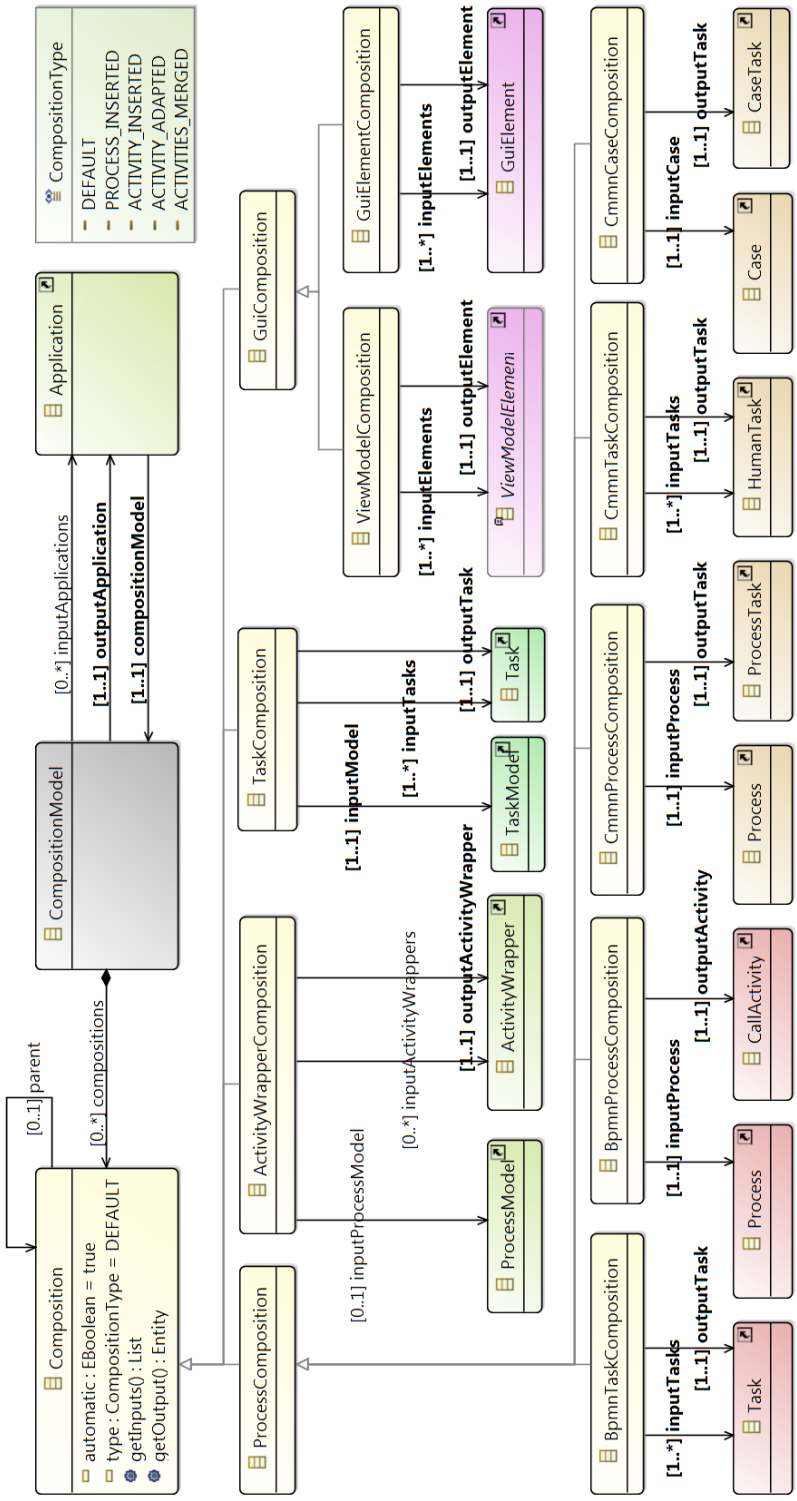


Abbildung 6.21: Das KomBInoS-Kompositionsmodell

6.8.3 Diskussion

Die ursächliche Erweiterung und Orchestrierung von Diensten steht nicht im Fokus der Arbeit. Dies kann manuell erfolgen oder mit Hilfe von semantischen Technologien automatisch geplant werden (Klusch 2008). Vielmehr wird betrachtet, wie die bereits vorhandenen UI-Modelle der orchestrierten Dienste auf Basis eines Kompositionsmodells konsistent komponiert werden können. Das Kompositionsmodell enthält hierzu zu Beginn der Anwendung der im nächsten Kapitel beschriebenen KomBInoS-Methode lediglich Information über die zusammengeführten Aktivitäten eines Kollaborationsprozesses. Durch die Anwendung der KomBInoS-Methode wird das Kompositionsmodell sukzessive aufgebaut. Somit stellt das Kompositionsmodell einen nachvollziehbaren Verlauf der durchgeführten UI-Kompositionen auf Basis einer initialen Beschreibung der Orchestrierung auf Dienstebene bzw. auf Ebene des internen Kollaborationsprozesses dar.

6.9 Fazit

In diesem Kapitel wurde die Metamodell-Architektur von KomBInoS zur konsistenten und umfassenden Modellierung multimodaler Dialogschnittstellen für Smart Services konzipiert und mit Hilfe der Metamodellierungssprache Ecore umgesetzt. Die Metamodell-Architektur integriert außerdem homogen ein Kompositionsmodell, welches gemäß der geschilderten Komplexitätsstufen die Wiederverwendung von UI-Artefakten auf Basis einer zuvor durchgeführten Dienstorchestrierung ermöglicht. Quantitative Kennzahlen zur Metamodell-Architektur befinden sich in Anhang A (S. 269). Die aufeinander abgestimmten Metamodelle beantworten somit die modellierungsbezogenen Teilaspekte der in Kapitel 1.2.3 (S. 4) formulierten Forschungsfrage (1) „*Wie kann man multimodale Dialogschnittstellen für Smart Services schnell und mit geringem Programmieraufwand entwickeln? Wie sehen eine notwendige Metamodell-Architektur und entsprechende Modelltransformationen aus, die funktional vollständige und über Modalitätsgrenzen hinweg konsistente Benutzerschnittstellen erschaffen?*“ und (2) „*Wie kann man existierende multimodale Dialogschnittstellen von Smart Services wiederverwenden? Bis zu welchem Grad ist dies sinnvoll möglich?*

Die Modellierungsaspekte in den Fragestellungen sind eng verknüpft mit methodischen Fragestellungen, wie und in welcher Weise die Metamodelle anzuwenden und aufeinander abzubilden sind, um dem Anspruch einer integrierten modellgetriebenen Entwicklung und Komposition gerecht zu werden. Dies wird Gegenstand des nächsten Kapitels sein.

KAPITEL 7

Die KomBInoS-Methode

Dieses Kapitel stellt die integrierte Methode zur modellgetriebenen Entwicklung und Komposition von KomBInoS-UIs als Antwort auf die eingangs formulierte Forschungsfrage (3) vor. In diesem Zuge wird außerdem die Forschungsfrage (1) unter dem Gesichtspunkt der Gestaltung von notwendigen Modelltransformationen im Rahmen der UI-Entwicklung beantwortet. Eine detaillierte Betrachtung der funktionalen und temporalen Abhängigkeiten zwischen UIs im Rahmen einer Anforderungsanalyse beantwortet die Forschungsfrage (2) bzgl. der Grenzen einer sinnhaften Komposition von UIs.

7.1 Einleitung

Die integrierte KomBInoS-Methode zur modellgetriebenen Entwicklung und Komposition soll es ermöglichen, multimodale Dialogschnittstellen für Smart Services mit geringem Aufwand zu erstellen. Die operative Grundlage von für KomBInoS relevante Smart Services stellen formal repräsentierte Kollaborationsprozesse gemäß Definition 6 (S. 40) dar. Eine eingehende Anforderungenanalyse in Kapitel 7.2 (S. 143) untersucht zunächst aus Sicht der Benutzerschnittstelle relevante Anwendungsszenarien zur Anwendbarkeit der Methode, bevor anschließend in Kapitel 7.3 (S. 147) die einzelnen Arbeitsschritte der Methode im Detail erläutert werden. Hierbei werden die im vorherigen Kapitel beschriebenen Metamodelle durch eine Menge von aufeinander aufbauenden Modelltransformationen in Beziehung gesetzt.

7.2 Anforderungen

Bevor die KomBInoS-Methode im Detail erläutert wird, müssen zunächst die funktionalen Anforderung und Voraussetzungen zur Anwendbarkeit der Methode diskutiert werden. Ausgangspunkt für die Anwendung der vorgestellten Methode ist die Existenz

eines formal repräsentierten Kollaborationsprozesses. Wie in Kapitel 6.3 (S. 112) erläutert, unterstützt die Methode ohne Beschränkung der Allgemeinheit die Prozessmodellierungssprachen BPMN in der Version 2.0 und CMMN in der Version 1.1. Weitere können leicht eingebunden werden. Jeder ablauffähige, informationsverarbeitende Prozess verfügt bereits über eine Domänenmodellierung. Diese kann in unterschiedlichen Formen vorliegen, etwa als XML-Schema, in der Interface Definition Language (OMG 2014b) oder lediglich als implementierte Klassen in einer Programmiersprache. In jedem Fall ist es möglich und notwendig, die (gegebenenfalls implizit) vorhandene Domänenmodellierung gleichwertig als Ecore-Modell darzustellen und diese so der Methode zur Verfügung zu stellen. Ein optionales Kompositionsmodell beschreibt initial außerdem eine etwaige Dienstorchestrierung.

Jeder in Betrieb genommene Dienst ist mit einer eindeutigen Id und einer Versionsnummer versehen. Eine Änderung eines Dienstes führt bei einer wiederholten Inbetriebnahme zu einer Inkrementierung der Versionsnummer, sodass der neue Dienst technisch neben seiner alten Version koexistieren kann, um z. B. vertragliche Vereinbarungen im Produktiveinsatz oder API-Kompatibilität einzuhalten. Eine UI wird daher immer für eine bestimmte Version eines Dienstes entwickelt und kann unabhängig davon natürlich weiterentwickelt werden. Dies führt unter Berücksichtigung der Komponierbarkeit von Diensten und UIs zu den in Abbildung 7.1 aufgezeigten rekursiven Abhängigkeiten, die im Folgenden erläutert werden. Die Anzahl der zu einem neuen Dienst orchestrierten Basisdienste ist prinzipiell unbeschränkt.

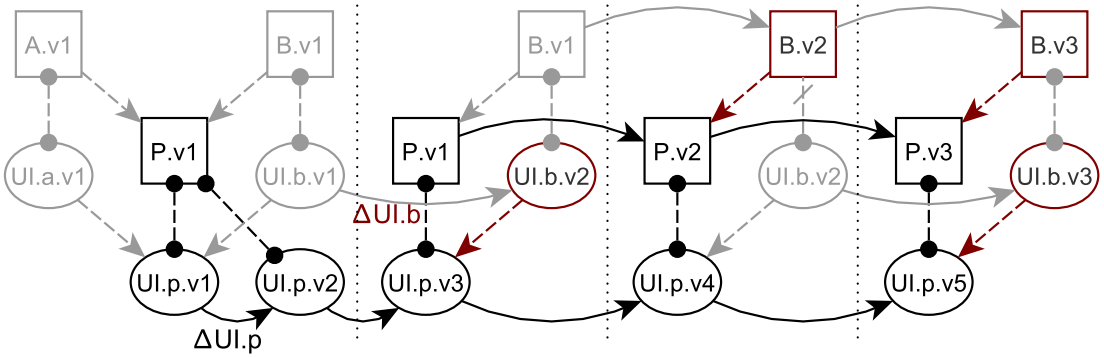


Abbildung 7.1: Versionsabhängigkeiten von (komponierten) Diensten und dazugehörigen UIs

Bei der modellgetriebenen Entwicklung und Komposition von multimodalen Dialogschnittstellen für Smart Services lassen sich nun verschiedene Ausgangssituationen bzw. Motivationen zur Methodenanwendung unterscheiden, aus denen sich funktionale Anforderungen an die Methode ableiten lassen. Zunächst wird der Kompositionsaspekt, welcher in Abbildung 7.1 im Hintergrund grau und rot dargestellt ist, bei den Erläuterungen außer Acht gelassen.

(1) **Für einen neuen Dienst bzw. dessen Kollaborationsprozess P soll eine Benutzerschnittstelle UI_P von Grund auf neu entwickelt werden ($P.v1 \rightarrow UI_P.v1$).** Hierzu ist ein klar strukturiertes Vorgehen notwendig, um die Komplexität, die durch die Multimodalität auch im Entwicklungsprozess entsteht, beherrschbar zu machen. Zur Reduzierung des potentiellen Aufwandes sollen optional bei Bedarf und Eignung auch Modellartefakte existierender UIs entlang der Metamodellarchitektur wiederverwendet werden können. Diese UIs müssen nicht unbedingt zu Gliedern in der Kompositionskette des neuen Dienstes gehören. Das Ergebnis stellt eine konsistente und funktional vollständige Benutzerschnittstelle für den Dienst oder Prozess dar.

(2) **Etwa zur Verbesserung des Nutzererlebnisses sollen nachträglich Anpassungen an konkreten UI-Modellen vorgenommen werden ($UI_P.v1 \rightarrow UI_P.v2$).** Die Anforderungen hierbei sind ähnlich zu den im vorangegangenen Punkt. Schließlich sorgt eine wiederholt durchgeführte Validierung auf Modellebene für die notwendige Konsistenz und Vollständigkeit der Benutzerschnittstelle in Bezug auf den Dienst bzw. Prozess. Da konkrete Modelle - sofern sie nicht interpretiert werden - in finalen Code überführt werden, müssen die entsprechenden M2C-Transformationen so gestaltet sein, dass möglichst wenige Ergänzungen am finalen Code notwendig sind. Dies steigert die Codequalität und ist über verschiedene UIs hinweg einer einheitlichen Bedienung und einem wiedererkennbaren Aussehen im Sinne eines Corporate Design zuträglich. Außerdem müssen diese Ergänzungen so gekapselt werden können, dass sie von einer wiederholten Codegenerierung nicht tangiert werden. Andernfalls können manuelle Code-Verschmelzungen erforderlich werden.

(3) **Aufgrund einer Anpassung des Dienstes ($P.v1 \rightarrow P.v2$) müssen etwaige Änderungen in die assoziierte Benutzerschnittstelle propagiert werden ($UI_P.v3 \rightarrow UI_P.v4$).** Sousa u. a. (2008, S. 558) und Sousa u. a. (2010) skizzieren in diesem Zusammenhang einen entsprechenden Algorithmus zur konsistenten Anpassung und Vervollständigung eines GUI-Modells aufgrund einer geänderten Prozessdefinition. Zusätzlich zur Anpassung konkreter UI-Modelle im Punkt (2) muss es also im Prinzip möglich sein, an jedem beliebigen Punkt wiederholt in die Anwendung der Methode einzusteigen. Aufgrund der Metamodellarchitektur muss eine Anpassung in einem Modell konsistent zu den darüber liegenden, abstrakteren Modellebenen sein und darf die Vollständigkeit der UI nicht verletzen. Außerdem muss die Anpassung konsistent durch die nachgelagerten, konkreteren Modelle propagiert werden. Dabei ist zu berücksichtigen, dass nicht betroffene Modellteile unberührt bleiben, um gegebenenfalls zuvor durchgeführte manuelle Anpassungen zu erhalten. Aufgrund der unterschiedlichen Abstraktionsebenen der Metamodelle und weil es sich bei den Modelltransformationen im Rahmen eines Forward-Engineerings im Allgemeinen um injektive Abbildungen handelt, ist ein Reverse- oder Roundtrip-Engineering methodisch und technisch nur eingeschränkt möglich. Die Bedeutungserhaltung manueller Anpassungen bei wiederholt ange-

wandtem Forward-Engineering von einem beliebigen Einstiegspunkt ist technisch ebenfalls herausfordernd. Derzeit bietet EMF hierfür auch keine universelle Lösung an. Eine solche könnte perspektivisch durch Einsatz vorhandener Werkzeuge wie EMFCompare[1] und mit Hilfe der Verfolgbarkeit von QVT-Transformationen (Willink und Matragkas 2015) realisiert werden.

Die Kompositionsfähigkeit von Diensten steigert nun die Komplexität einer modellgetriebenen Methode zur Entwicklung und Komposition von UIs nochmals, weil sie aufgrund kausaler Zusammenhänge Einfluss auf die zuvor dargelegten Punkte nimmt. Somit ergeben sich folgende zusätzliche Anwendungsfälle und prinzipielle Anforderungen an die Methode.

(4) **Ein neuer Dienst P wurde als Komposition von bestehenden Diensten A und B realisiert. Die nun zu entwickelnde Benutzerschnittstelle des Dienstes UI_P soll die UI-Modelle seiner Bestandteile - sofern vorhanden - wiederverwenden ($UI_A.v1 \circ UI_B.v1 \rightarrow UI_P.v1$).** Hierzu ist ein initiales Kompositionsmodell erforderlich, welches die durchgeführte Dienstorchestrierung reflektiert. Dieses Modell muss der Methode in jedem Transformationsschritt als relevante Kontextinformation zur Identifizierung und Wiederverwendung relevanter existierender UI-Artefakte gemäß der Komplexitätsstufen zur Verfügung stehen.

(5) **Die UI des komponierten Dienstes soll angepasst werden ($UI_P.v1 \rightarrow UI_P.v2$).** Dieser Punkt entspricht dem vorangegangenen Punkt (2).

(6) **Die UI eines Bestandteils des komponierten Dienstes wurde angepasst ($UI_B.v1 \rightarrow UI_B.v2$). Die Anpassungen sollen in die komponierte UI übernommen werden ($UI_B.v2 \circ UI_P.v2 \rightarrow UI_P.v3$).** Hierzu muss ersichtlich sein, welche Modellfragmente betroffen sind. Dies kann aufgrund von nachträglichen manuellen Anpassungen an der $UI_P.v2$ in Punkt (5), welche als Grundlage für $UI_P.v3$ dient, problematisch sein. Zuerst müssen auf Modellebene die Deltas von $UI_B.v2$ zu $UI_B.v1$ (ΔUI_B) sowie $UI_P.v2$ zu $UI_P.v1$ (ΔUI_P) gebildet werden, um die unabhängigen Änderungen an den UIs seit dem Zeitpunkt, in dem die Abhängigkeit geschaffen oder synchronisiert wurde, nachvollziehen zu können. Dann identifiziert man anhand von ΔUI_B sowie des Kompositionsmodells von UI_P die Menge A_{UI_P} der Elemente in $UI_P.v1$, die von der Änderung in $UI_B.v2$ betroffen sind. Die Elemente der Schnittmenge $A_{UI_P} \cap \Delta UI_P$ wurden sowohl in $UI_B.v2$ als auch in Schritt (5) manuell angepasst und bedürfen gegebenenfalls einer manuellen Revision. Elemente der Differenzmenge $A_{UI_P}/\Delta UI_P$ können automatisiert ausgetauscht oder gemäß ΔUI_B angepasst werden.

(7) **Ein Bestandteil des komponierten Dienstes wurde angepasst ($B.v1 \rightarrow B.v2$). Die Anpassung wurde in den komponierten Dienst übernommen ($P.v1 \rightarrow P.v2$). Nun soll die vorhandene komponierte UI auf den neuen Dienst aktualisiert werden ($UI_P.v3 \rightarrow UI_P.v4$).** Da $UI_B.v2$ nicht mehr zum

[1] siehe http://www.eclipse.org/emf/compare/ (letzter Zugriff: 04.11.2017)

ursprünglich angepassten Dienst B.v2 kompatibel ist und $UI_B.v3$ noch nicht existiert, ist eine wiederholte UI-Komposition nach Punkt (6) nicht möglich. Somit kann dieser Fall auf Punkt (3) abgebildet werden.

(8) **Ein Bestandteil des komponierten Dienstes wurde aktualisiert ($B.v2 \rightarrow B.v3$). Diese Anpassung wurde bereits in den komponierten Dienst ($P.v2 \rightarrow P.v3$) propagiert. Außerdem wurde die UI des Dienstes B angepasst ($UI_B.v2 \rightarrow UI_B.v3$). Nun soll ebenfalls die UI des Dienstes P aktualisiert werden ($UI_B.v3 \circ UI_P.v4 \rightarrow UI_P.v5$).** Dieser Fall kann als Sonderfall des Punktes (6) verstanden werden, wobei die Menge ΔUI_P der zwischenzeitlich manuell angepassten Elemente der UI_P die leere Menge $\emptyset$ darstellt.

Nach der soeben erfolgten Analyse der Anwendungsszenarien und der daraus abgeleiteten Anforderungen wird nun im nächsten Unterkapitel die daraus resultierende integrierte Methode zur modellgetriebenen Entwicklung und Komposition im Detail vorgestellt.

7.3 Arbeitsschritte

Abbildung 7.2 zeigt die konzeptuelle Darstellung der Methode. Der Prozess beschreibt eine alternierende Folge von automatischen und manuellen Arbeitsschritten, da im Allgemeinen das Zielmodell durch eine automatisch durchgeführte Modelltransformation nicht in dem Umfang aufgebaut werden kann, der erforderlich ist, um die nächste Modelltransformation sinnvoll anzustoßen. So folgt auf jeden automatischen in der Regel ein manueller Arbeitsschritt, der die Informationslücke schließt.

Die durchzuführenden Arbeitsschritte sind durch Pfeile gekennzeichnet und nummeriert. Manuelle Arbeitsschritte werden zudem als Pfeile mit gleichem Ursprung und Ziel dargestellt. Das Delta zwischen einem partiellen und dem darauf aufbauenden vollständigen Zielmodell sollte so weit als möglich minimal sein. Gleichzeitig sollten dem Entwickler die Informationsanforderungen an ein vollständiges Zielmodell bekannt gemacht werden, die es zu erfüllen gilt. Daraus leiten sich unmittelbar die manuellen Modellierungstätigkeiten im Sinne einer Informationsanreicherung ab. Die Schaffung dieser notwendigen Transparenz wird als Teil der Werkzeugunterstützung angesehen, während die Definition der Anforderungen als Vorbedingung für eine Modelltransformation Teil der Methodenkonzeption sind. Im Folgenden werden nun die einzelnen Arbeitsschritte zur Instanzbildung sowie die Modellschnittstellen unter Verweis auf die Nummerierung in Abbildung 7.2 im Detail erläutert. Dies geschieht einmal informativ in Textform aber auch formal durch Abbildungsregeln.

Hierbei werden einige Hilfsmethoden und Operatoren verwendet, welche nun vorab definiert werden. Der im Folgenden verwendete Typ des Ecore-Meta-Metamodells EClass bezeichnet ein beliebiges Konzept eines Ecore-basierten Metamodells. Der Typ EObject bezeichnet eine beliebige Instanz eines solchen Konzepts. Die Methode

$$\mathbf{el} : \mathcal{P}(EObject) \longrightarrow EObject$$

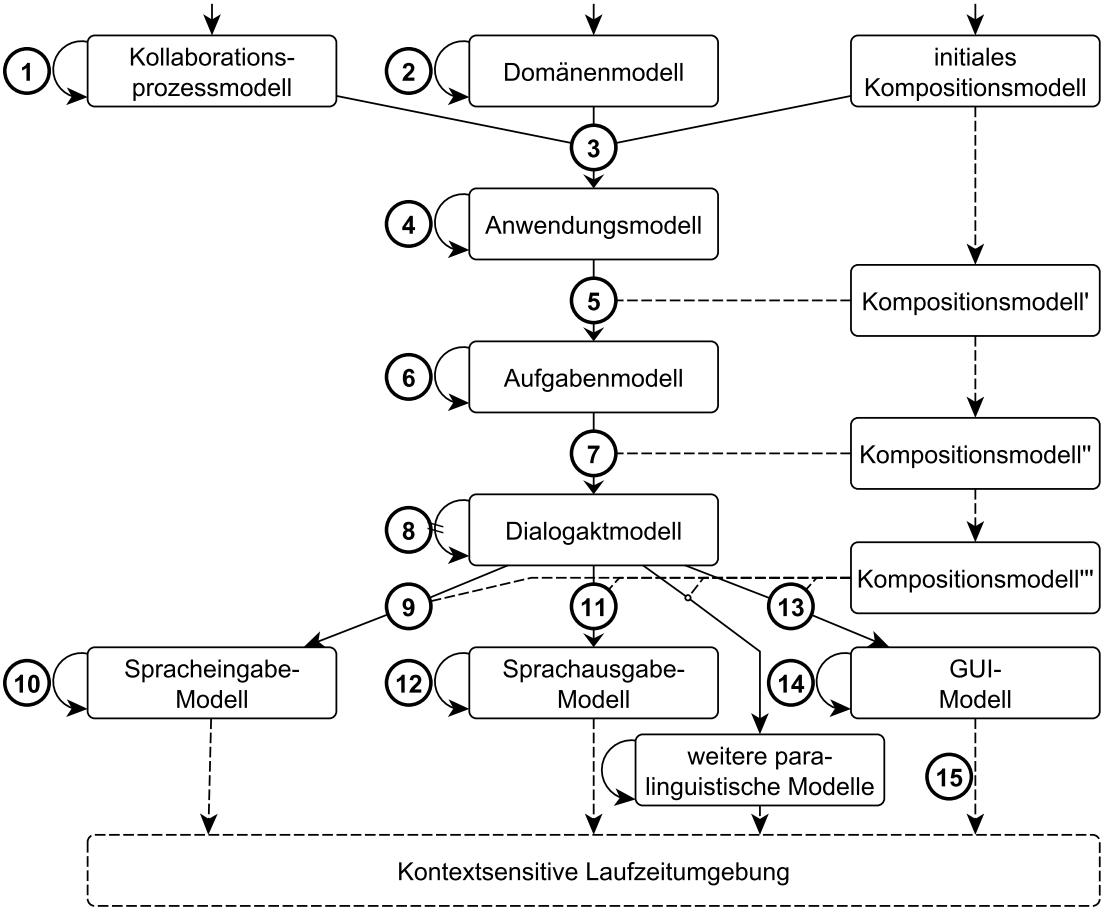


Abbildung 7.2: Die integrierte KomBInoS-Methode zur modellgetriebenen Entwicklung und Komposition von multimodalen Dialogschnittstellen für Smart Services

liefert ein beliebiges Element einer Menge zurück, der Operator + konkateniert zwei Zeichenketten. Die Methoden (7.1) bis (7.3) dienen zur Informationsabfrage auf einem Kompositionsmodell. Im Einzelnen liefert die Methode

$$\begin{aligned} \textbf{getComposition} : EObject \times CompositionModel &\longrightarrow Composition \\ (obj, cm) &\longmapsto comp \\ \text{mit } comp \in cm.compositions \wedge obj = comp.getOutput() & \end{aligned} \tag{7.1}$$

zu einem komponierten Objekt *obj* die zugehörige Composition-Instanz *comp* zurück, welche *obj* als Ausgabe spezifiziert. In dem Zusammenhang liefert die Methode

$$\begin{aligned} \textbf{isComposed} : EObject \times CompositionModel &\longrightarrow Boolean \\ \text{mit } isComposed(obj, cm) = getComposition(bj, cm) &\neq null \end{aligned} \tag{7.2}$$

true zurück, falls das Objekt *obj* im Rahmen einer Komposition entstand. Die Methode

$$\begin{aligned} \textbf{getInputs} : EObject \times CompositionModel &\longrightarrow \mathcal{P}(EObject) \\ (obj, cm) &\longmapsto inputs \\ \text{mit } inputs = getComposition(obj, cm).&getInputs() \end{aligned} \tag{7.3}$$

sammelt zu einem komponierten Objekt *obj* die zugehörigen Bestandteile seiner Komposition ein und gibt diese in Form einer Menge zurück. Die Signaturen der Methoden vernachlässigen zur universellen Anwendbarkeit in der folgenden Konzeption die konkreten Typen der Ein- und Ausgabeparameter. Diese lassen sich jedoch bei der späteren Implementierung im Kontext eindeutig bestimmen. Zur leichteren Lesbarkeit wird der Parameter *cm* im Folgenden ausgelassen, da die Methoden direkt auf dem Kompositionsmodell operieren. Die Implementierung der Methoden erfolgte im Rahmen einer QVT-Query unter Verwendung der Anfragesprache OCL. Listing 7.1 zeigt als Beispiel die Implementierung der Methode getComposition (7.1).

Listing 7.1: Implementierung der Methode getComposition als QVT-Query

```
1 query CompositionModel::getComposition(obj : EObject) : Composition
2 {
3     return self.allSubobjectsOfType(Composition)
4                     ->selectOne(comp | comp.getOutput() = obj);
5 }
```

Des Weiteren werden Hilfsmethoden zur Typreflexion deklariert, welche in äquivalenter Form auch für EMF bzw. die OCL existieren und darum nicht gesondert definiert werden. Die Methode

$$\textbf{typeOf} : EObject \times EClass \longrightarrow Boolean$$

prüft auf Typgleichheit, die Methode

$$\textbf{kindOf} : EObject \times EClass \longrightarrow Boolean$$

prüft, ob ein Objekt vom angegebenen oder einem davon abgeleiteten Konzept ist. Für $typeOf(obj, AType)$ soll $AType(obj)$ eine äquivalente Kurzschreibweise darstellen. Die Methode

$$\textbf{clone} : EObject \longrightarrow EObject$$

erzeugt eine Kopie eines Objekts.

7.3.1 Erstellung des Kollaborationsprozesses

Der Kollaborationsprozess wird der Methode als Startpunkt im Allgemeinen unveränderlich in Bezug auf die darin implementierte Logik zur Verfügung gestellt.

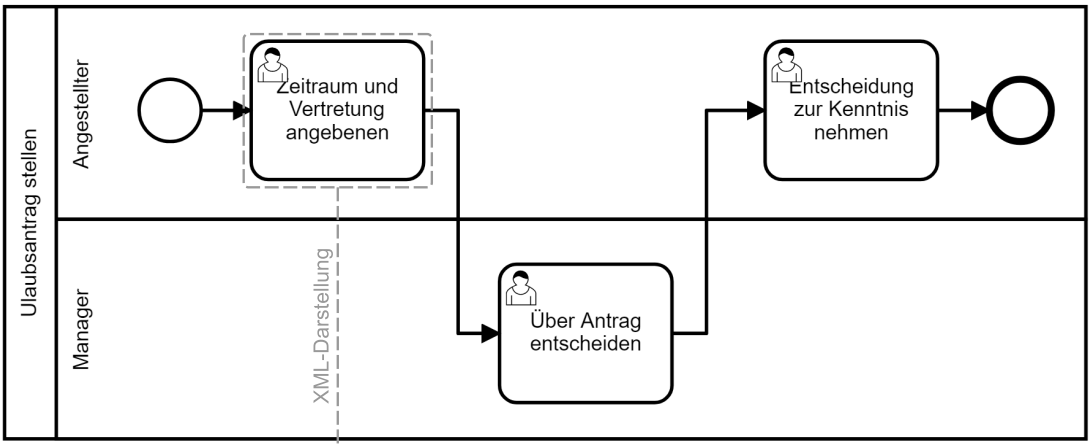
(1) Optionale Annotation des Kollaborationsprozesses

Dennoch kann mit Hilfe der KomBInoS-spezifischen Prozesserweiterung (Kapitel 6.3.3, S. 117) das Ein- und Ausgabeverhalten von Prozessaktivitäten modelliert und damit der durch Prozessvariablen gestaltete Datenfluss innerhalb eines BMPN- oder CMMN-basierten Kollaborationsprozesses explizit gemacht werden. Dadurch reduziert sich in nachfolgenden Phasen der Methode der manuelle Modellierungsaufwand, etwa bei der Abbildung des Domänenmodells auf Prozessvariablen während der Erzeugung des Anwendungsmodells in Kapitel 7.3.3 (S. 153).

Hierzu können entsprechende Variablenannotation händisch erstellt werden. Dies betrifft in der Regel neue, gemäß Kompositionsmodell nicht komponierte Aktivitäten. Für komponierte Aktivitäten kann diese Information zu einem späteren Zeitpunkt durch Anwendung der Transformation **annotation2variable** (7.6) aus den Aufgabenmodellen der Kompositionsbestandteile wiederverwendet werden.

Beispiel: Kollaborationsprozess

Abbildung 7.3 illustriert die Annotation eines Kollaborationsprozesses an einem einfachen Beispiel, welches im weiteren Verlauf dieses Kapitels immer wieder aufgegriffen und verfeinert wird. Der gezeigte Kollaboratiosprozess modelliert die Beantragung von Urlaub. Der XML-Ausschnitt zeigt eine Annotation der Aufgabe *Zeitraum und Vertretung angeben* mit entsprechenden Prozessvariablen.




```
<bpmn:userTask id="..." name="Zeitraum und Vertretung angebenen">
  <bpmn:incoming>...</bpmn:incoming>
  <bpmn:outgoing>...</bpmn:outgoing>
  <bpmn:extensionElements>
    <kombinos:VariableAnnotations>
        <kombinos:VariableAnnotation varName="start_date"   varType="string" varDirection="OUT"/>
        <kombinos:VariableAnnotation varName="end_date"     varType="string" varDirection="OUT"/>
        <kombinos:VariableAnnotation varName="substitution" varType="string" varDirection="OUT"/>
    </kombinos:VariableAnnotations>
  </bpmn:extensionElements>
</bpmn:userTask>
```

Abbildung 7.3: Beispiel - Kollaborationsprozess mit Variablenannotation

7.3.2 Erstellung des Domänenmodells

Jeder Prozess operiert auf einem komplexen Domänenmodell. Wie eingangs in Kapitel 7.2 (S. 143) gefordert, setzt die Anwendung der Methode die Existenz eines Ecore-basierten Domänenmodells voraus. Dieses liegt entweder bereits in der geforderten Notation vor oder kann durch eine Modelltransformation automatisiert erzeugt werden. Transformationen von z. B. XML-Schema nach Ecore sind gängige Praxis und wurden auch zur Erzeugung des CMMN-Metamodells (Kapitel 6.3.2, S. 115) im Rahmen der Arbeit angewendet. Alternativ ist eine händische Modellierung der Domäne anhand des Kollaborationsprozesses und der darin verwendeten Prozessvariablen erforderlich. Hierbei können Variablenannotationen (Kapitel 6.3.3, S. 117) als Ausgangsbasis zur dialogspezifischen Domänenmodellierung dienen. Gleichzeitig muss immer der Zugriff auf relevante Domänenkonzepte von Benutzerschnittstellen in der Kompositionskette sichergestellt sein, um darauf operierende UI-Artefakte effektiv wiederverwenden zu können. Im Ergebnis steht der Methode ein Domänenmodell zur weiteren Verfeinerung zur Verfügung.

(2) Adaption des Domänenmodells

Die Durchführung folgender Maßnahmen hat zum Ziel, das Domänenmodell auf die Anforderungen der multimodalen dialogischen Interaktion zu adaptieren. Selbstverständlich können diese Punkte bereits während der initial händischen Modellierung Berücksichtigung finden.

(1) **Verknüpfung relevanter Domänenkonzepte mit dialogspezifischen Basiskonzepten der abstrakten KomBInoS-Domänenmodellierung.** Diese Verknüpfung erfolgt durch Vererbung. Dabei ist es ohne Weiteres möglich, ein Domänenkonzept von mehreren Basiskonzepten erben zu lassen. SiAM-dp verlangt lediglich eine Ableitung dialogrelevanter Konzepte von Entity (Neßelrath 2016, S. 117). Dies gilt auch weiterhin in KomBInoS. Darüber hinaus ist es notwendig, die Ableitung so spezifisch wie möglich vorzunehmen, um gegebenenfalls existierende Logik zur Interaktions- und Dialogverarbeitung wiederzuverwenden.

(2) **Konsolidierung der verknüpften Domänenkonzepte.** Die Vererbung einzelner Domänenkonzepte von existierenden Konzepten kann eine unerwünschte Redundanz bedeutungs- oder gar namensgleicher Attribute nach sich ziehen, die es aufzulösen gilt. Eine Anpassung vererbter oder aus anderen Domänenmodellen eingebundener Konzepte ist nicht möglich, um die Kompositionsfähigkeit nicht zu gefährden.

(3) **Tiefe schaffen.** Zanten (1998) beschreibt einen hierarchischen Ansatz der Dialogakterzeugung, um in einem kooperativen Question-Answering-Szenario unspezifische Fragen sowie Über- und Unterbeantwortung adaptiv abzubilden. Mit einer entsprechend hierarchischen Domänenmodellierung kann dieser Ansatz auch hier effektiv angewendet werden. Um für einen solchen adaptiven Dialog mit gemischter Initiative geeignet zu sein, müssen flach strukturierte Domänenobjekte, insbe-

sondere auf Basis von Prozessvariablen händisch modellierte, in eine thematisch gegliederte hierarchische Struktur überführt werden. Dies wird erreicht, indem logisch zusammengehörige Variablen und Attribute gegebenenfalls in neue (Teil-) Konzepte ausgelagert werden. Hierbei sind die Maßnahmen (1) und (2) wiederholt anzuwenden. Anschließend werden die Teilkonzepte im ursprünglichen Domänenkonzept per Containment-Referenz verknüpft. Die (Teil-) Konzeptgrenzen stellen somit den thematischen Kontext einer späteren dialogischen Interaktion her.

Im Falle einer zu komponierenden Benutzerschnittstelle verfügen die Bestandteile der Komposition bereits rekursiv über Domänenmodelle, die nach den obigen Prinzipien ausgearbeitet wurden. Um die Funktionsweise der zugehörigen UI-Artefakte in der komponierten UI sicherzustellen, müssen relevante Konzepte dieser Domänenmodelle im komponierten Domänenmodell Eingang finden. Konzeptuell ist dies durch Vererbung und Aggregation, wie oben beschrieben, möglich, da eine (möglicherweise komplexere) Abbildung des dialogspezifischen Domänenmodells auf Prozessvariablen zur Laufzeit erst im nächsten Schritt erfolgt. Ein so strukturiertes Domänenmodell liefert schließlich im Rahmen eines modellgetriebenen Vorgehens die Grundlage für einen natürlicheren Dialog, als dies beispielsweise im DomainEditor (Kapitel 5.2.3, S. 83) mit flach strukturierten Datenstrukturen der Fall ist.

Beispiel: Domänenmodell

Abbildung 7.4 veranschaulicht die Erstellung des dialogspezifischen Domänenmodells an einem Beispiel.

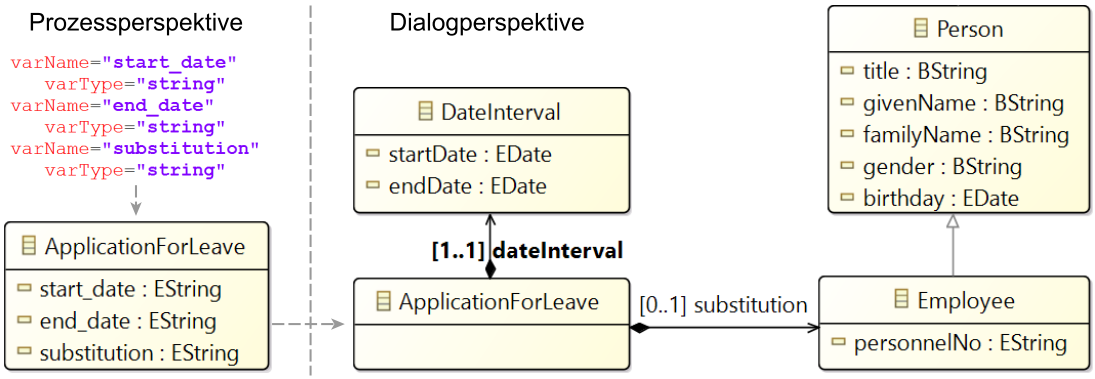


Abbildung 7.4: Beispiel - Umformung eines Domänenobjekts aus Prozessperspektive in die Dialogperspektive

Das Domänenobjekt Urlaubsantrag (ApplicationForLeave) wird aus der Prozess- in die Dialogperspektive überführt. Dies geschieht u. a. durch Umwandlung des textuelle Attributs *substitution* in eine Referenz vom Typ Employee. Employee ist wiederum eine Verfeinerung des KomBInoS-Domänenkonzepts Person (Kapitel 6.4, S. 118). Auf die gleiche Weise wird mit den Attributen zur Erfassung des Urlaubszeitraums verfahren.

Diese werden durch das Dialogkonzept DateInterval substituiert. So wird ein Rahmen geschaffen, der etwa beim Schablonenfüllen (Kapitel 4.3.4, S. 64) unter Verwendung bereits existierender Artefakte zur Konversation über Personen und Zeitintervalle ausgenutzt werden kann.

7.3.3 Erstellung des Anwendungsmodells

Die Aufgabe eines Anwendungsmodells ist es, die beteiligten Teilmodelle der gesamten Benutzerschnittstelle schnell an einem einzigen Ort zugreifbar zu machen und vom konkreten Kollaborationsprozess des Dienstes zu abstrahieren.

(3) Automatische Generierung des Anwendungsmodells
Liegen der Kollaborationsprozess und das dazugehörige Domänenmodell in der geeigneten Form vor, so kann daraus durch eine Modelltransformation automatisch ein initiales Anwendungsmodell erstellt werden.

Zur besseren Nachvollziehbarkeit der folgenden Transformationen ist es ratsam, sich die beteiligten Metamodelle aus Kapitel 6 (S. 111) nochmals ins Gedächtnis zu rufen. Ebenso sind einige Mengendefinitionen notwendig, auf die in einer Fallunterscheidung zurückgegriffen wird. Sei

$$\mathcal{ACT} := \{act \in process \mid kindOf(act, Activity)\}$$

die Menge aller Aktivitäten des Kollaborationsprozesses *process*. Sei weiter

$$\mathcal{ACT}_{\mathcal{C}} := \{act \in \mathcal{ACT} \mid isComposed(act)\}$$

die Menge aller Aktivitäten, die aus einer Prozesskomposition hervorgegangen sind. Dann enthält die Menge

$$\mathcal{ACT}_{\mathcal{N}} := \mathcal{ACT} / \mathcal{ACT}_{\mathcal{C}}$$

alle neuen Prozessaktivitäten. Außerdem unterscheidet man anhand des Kompositionstyps folgende Mengen:

$$\mathcal{ACT}_{\mathcal{CP}} := \{act \in \mathcal{ACT}_{\mathcal{C}} \mid getComposition(act).type = PROCESS - INSERTED\}$$

$$\mathcal{ACT}_{\mathcal{CA}} := \mathcal{ACT}_{\mathcal{C}} / \mathcal{ACT}_{\mathcal{CP}}$$

$$\mathcal{ACT}_{\mathcal{CAI}} := \{act \in \mathcal{ACT}_{\mathcal{CA}} \mid getComposition(act).type = ACTIVITY - INSERTED\}$$

$$\mathcal{ACT}_{\mathcal{CAA}} := \{act \in \mathcal{ACT}_{\mathcal{CA}} \mid getComposition(act).type = ACTIVITY - ADAPTED\}$$

$$\mathcal{ACT}_{\mathcal{CAM}} := \{act \in \mathcal{ACT}_{\mathcal{CA}} \mid getComposition(act).type = ACTIVITY - MERGED\}$$

Das Anwendungsmodell *app* (Application) ergibt sich schließlich gemäß der Transformation **bpmn2app** (7.4) anhand der Eingabeparameter BPMN-Kollaborationsprozess *process* (Definitions), Domänenmodell *dm* (EPackages) und Kompositionsmodell *comp*

(CompositionModel) wie folgt. Das Kompositionsmodell kann leer sein, falls der Prozess nicht aus einer Komposition hervorging, also vollständig neu erstellt wurde.

Die Darstellung einer Ergebnisstruktur orientiert sich in der vorliegenden Arbeit an der Notation für getypte Merkmalstrukturen. Mengenartige Merkmale werden darin durch Anwendung einer nachgelagerten Transformation aufgebaut, oft in Kombination mit prädikatenlogischen Ausdrücken erster Ordnung, welche die Rolle eines Filters vor der eigentlichen Transformation übernehmen.

$$\begin{aligned} \textbf{bpmn2app} : Definitions \times EPackage \times CompositionModel &\longrightarrow Application \\ (process, dm, cm) &\longmapsto app \end{aligned} \tag{7.4}$$

mit

$$app = \left[\begin{array}{ll} \textsc{Application} & \\ \text{id} & process.id \\ \text{name} & process.name \\ \text{domainModel} & dm \\ \text{compModel} & cm \\ \text{resourceModel} & \left[\begin{array}{ll} \textsc{ResourceModel} & \\ \text{roles} & \left\{\begin{array}{l} \forall\{l \in process \mid typeOf(l,Lane)\}: \\ \quad lane2role(l) \end{array}\right\} \end{array}\right] \\ \text{processModel} & \left[\begin{array}{ll} \textsc{BpmnProcessModel} & \\ \text{id} & process.id \\ \text{definition} & process \\ \text{variables} & \left\{\begin{array}{l} \forall\{va \in process \mid VariableAnnotation(va)\}: \\ \quad annotation2variable(va) \end{array}\right\} \\ \text{activities} & \left\{\begin{array}{l} \forall\{act \in process \mid kindOf(act,\ Activity)\}: \\ \quad activity2wrapper(act,\ cm) \end{array}\right\} \end{array}\right] \end{array}\right]$$

Im Anwendungsmodell *app* werden Verknüpfungen mit den Eingangsmodellen erstellt. Außerdem werden die beteiligten Rollen sowie die Aktivitäten im Prozess analysiert und ins Anwendungsmodell übertragen. Als Modellierungssprache des Kollaborationsprozesses wird hierbei BPMN vorausgesetzt. Die Abbildung von CMMN ins Anwendungsmodell wurde im Rahmen der betreuten Masterarbeit von Jafarian (2017) realisiert und ist analog aufgebaut, weshalb auf ein entsprechendes Listing verzichtet wird.

Im Detail greift die Generierung der Ergebnisstruktur auf drei nachgelagerte Transformationen zurück.

(1) Die Transformation **lane2role** (7.5) erzeugt für alle Instanzen einer Lane eine Rolle im Anwendungsmodell.

(2) Die Transformation **annotation2variable** (7.6) überführt generische XML-basierte Variablenannotationen in Ecore-basierte Variable-Instanzen. Hierbei werden der Typ der Variablen von XML-Schema nach Ecore sowie die Flussrichtung der Variablen von einer Zeichenkettenrepräsentation in eine Enumeration transformiert. Außerdem werden bereits leere DomainMapping-Instanzen angelegt, deren Attribute im nachfolgenden manuellen Arbeitsschritt zu spezifizieren sind. Dazu zählen die Angabe (1) eines dialogspezifischen Domänenkonzepts, (2) eines Merkmals des zuvor spezifizierten Domänenkonzepts sowie (3) optionale Logik in Form von Skripten zur Datenmediation zwischen Kollaborationsprozess und dialogspezifischem Domänenmodell.

(3) Die Transformation **activity2wrapper** (7.7) erzeugt für alle von Activity erbenden Instanzen im Prozess, z. B. UserTask, eine ActivityWrapper-Instanz, welche wiederum Variablen enthalten kann.

$$\begin{aligned} \textbf{lane2role} : Lane &\longrightarrow Role \\ lane &\longmapsto role \end{aligned} \tag{7.5}$$

mit

$$role = \begin{bmatrix} \textsc{Role} & \\ \text{id} & lane.id \\ \text{name} & lane.name \end{bmatrix}$$

$$\begin{aligned} \textbf{annotation2variable} : VariableAnnotation &\longrightarrow Variable \\ annotation &\longmapsto variable \end{aligned} \tag{7.6}$$

mit

$$variable = \begin{bmatrix} \textsc{Variable} & \\ \text{name} & annotation.name \\ \text{type} & mapToEcoreType(annotation.type) \\ \text{direction} & DirectionEnum.getByName(annotation.direction) \\ \text{domainMapping} & \begin{bmatrix} \textsc{DomainMapping} & \\ \text{concept} & \textbf{\textit{(}}\rightarrow \textbf{\textit{manuell)}} \\ \text{feature} & \textbf{\textit{(}}\rightarrow \textbf{\textit{manuell)}} \\ \text{exprIn} & \textbf{\textit{(}}\rightarrow \textbf{\textit{manuell, optional)}} \\ \text{exprOut} & \textbf{\textit{(}}\rightarrow \textbf{\textit{manuell, optional)}} \end{bmatrix} \end{bmatrix}$$

$$\begin{aligned} \textbf{activity2wrapper} : Activity \times CompositionModel &\longrightarrow ActivityWrapper \\ (act, cm) &\longmapsto aw \end{aligned} \tag{7.7}$$

mit

$$aw = \begin{bmatrix} \textsc{ActivityWrapper} & \\ \text{id} & act.id \\ \text{name} & act.name \\ \text{bpmnActivity} & act \\ \text{agentTask} & UserTask(act) \vee ManualTask(act) \\ \text{role} & getRole(act.lanes.el().id) \\ \text{variables} & getVars(act,\ cm) \end{bmatrix}$$

In der Transformation **activity2wrapper** (7.7) gibt die Methode **getRole** anhand der eingegebenen Id die entsprechende Rolle des Resourcenmodells zurück. Die Abbildung **getVars** (7.8) sorgt bei der Erzeugung eines ActivityWrappers anhand einer Fallunterscheidung für eine etwaige Wiederverwendung von Variable-Instanzen samt DomainMappings, sofern anhand des Kompositionsmodells ersichtlich ist, dass die zugrundeliegende Aktivität des Prozessmodells komponiert wurde (Fall $\boxed{B}$). Ansonsten werden diese Instanzen partiell, d.h. mit leeren DomainMappings, durch eine weitere Anwendung der Transformation (7.6) erzeugt (Fall $\boxed{A}$). Die Fallunterscheidung erfolgt auf Basis der zuvor definierten Mengen $\mathcal{ACT}_{\mathcal{N}}$ und $\mathcal{ACT}_{\mathcal{C}}$ und ist auf die in Kapitel 6.8.1 (S. 137) geschilderten Anforderungen bzgl. der Komposition von prozessorientierten Diensten zurückzuführen.

$$\begin{aligned} \textbf{getVars} : Activity \times CompositionModel &\longrightarrow \mathcal{P}(Variable) \\ (act, cm) &\longmapsto vars \end{aligned} \tag{7.8}$$

mit

$$vars = \begin{cases} \boxed{A}\ \forall \text{va} \in \text{act.getVariableAnnotations}(): \\ \quad \text{annotation2variable(va)} & \textit{falls } act \in \mathcal{ACT}_{\mathcal{N}} \\ \boxed{B}\ \text{clone(getInputs(act).variables)} & \textit{falls } act \in \mathcal{ACT}_{\mathcal{C}} \end{cases}$$

Im Fall $\boxed{B}$ gilt zu beachten, dass der Teilausdruck *getInputs(act)* eine Menge an Elementen zurückliefert. Der erweiterte Teilausdruck *getInputs(act).variables* fragt folglich alle Variablen aller Elemente der Menge ab. Der ebenfalls im Folgenden verwendete Ausdruck *getInputs(act).el()* gibt - im Wissen, dass die Menge *getInputs(act)* nur ein einziges Element besitzt - dieses Element zurück.

Für einen auf Basis einer Aktivität $act \in \mathcal{ACT}_{\mathcal{C}}$ erzeugten ActivityWrapper aw wird im Kompositionsmodell als Seiteneffekt eine ActivityWrapperComposition-Instanz awc angelegt, mit

$$awc = \begin{bmatrix} \textsc{ActivityWrapperComposition} & \\ \text{parent} & getComposition(act) \\ \text{type} & getComposition(act).type \\ \text{inputProcessModel} & ipm \\ \text{outputActivityWrapper} & aw \end{bmatrix} \quad \text{für } act \in \mathcal{ACT}_{\mathcal{CP}}$$

bzw.

$$awc = \begin{bmatrix} \textsc{ActivityWrapperComposition} & \\ \text{parent} & getComposition(act) \\ \text{type} & getComposition(act).type \\ \text{inputActivityWrappers} & iaw \\ \text{outputActivityWrapper} & aw \end{bmatrix} \quad \text{für } act \in \mathcal{ACT}_{\mathcal{CA}}$$

mit

$$ipm \in cm.inputApplications.processModel \\ \wedge\ getInputs(act).el() \in ipm.definition$$

bzw.

$$iaw := \{aw \mid aw \in cm.inputApplications.processModel.activities \\ \wedge\ aw.bpmnActivity \in getInputs(act)\}$$

(4) Manuelle Ergänzung des Anwendungsmodells

Notwendige manuelle Tätigkeiten zur Vervollständigung des Anwendungsmodells umfassen die zusätzliche Erstellung von Variablen (Variable), falls der zugrundeliegende Kollaborationsprozess nicht vollständig annotiert wurde. Des Weiteren müssen fehlende Abbildungen ins dienstspezifische Domänenmodell (DomainMapping) neu erstellter Variablen angelegt bzw. vervollständigt werden. Der Fall $\boxed{B}$ in Formel (7.8) kann zur Bestimmung des Grades an zusätzlichen manuellen Tätigkeiten in Bezug auf wiederverwendete Domänenabbildungen differenzierter betrachtet werden. Falls die Aktivität $act \in \mathcal{ACT}_{\mathcal{CPI}} \cup \mathcal{ACT}_{\mathcal{CAI}}$, sind aufgrund der einfachen Wiederverwendung keine zusätzlichen Arbeiten nötig. Falls $act \in \mathcal{ACT}_{\mathcal{CAA}}$, ist möglicherweise das angegebene Konzept innerhalb einer Domänenabbildung zu spezialisieren. Es gilt obendrein, dass $\forall act \in \mathcal{ACT}_{\mathcal{CPI}} \cup \mathcal{ACT}_{\mathcal{CAI}} \cup \mathcal{ACT}_{\mathcal{CAA}} : |getInputs(act)| = 1$. Im Falle $act \in \mathcal{ACT}_{\mathcal{CAM}}$ können prinzipiell n Aktivitäten zu einer einzigen aggregiert worden sein. Hier muss also die Vereinigungsmenge der Variablen gegebenenfalls zuerst ausgedünnt werden. Eine Variable kann automatisiert aus der Menge entfernt werden, falls darin eine andere Variable existiert, die bei Namens-, Typ- und Richtungsgleichheit eine speziellere Domänenabbildung besitzt. Ein DomainMapping dm_1 ist spezieller als ein DomainMapping dm_2, falls gilt

$$dm_1.feature = dm_2.feature \wedge dm_2.concept.isSuperTypeOf(dm_1.concept)$$

Anschließend können Domänenabbildungen der verbliebenen Variablen bei Bedarf angepasst werden, um Daten des Kollaborationsprozesses auf Konzepte des dialogspezifischen Domänenmodells abzubilden.

Beispiel: Anwendungsmodell

Abbildung 7.5 zeigt das resultierende Anwendungsmodell auf Basis des in Abbildung 7.3 gezeigten Kollaborationsprozesses sowie des dialogspezifischen Domänenmodells aus

Abbildung 7.4. Für jede Prozessaktivität wurde eine ActivityWrapper-Instanz erzeugt, welche die durch Annotationen hinterlegten Prozessvariablen samt der Abbildung auf Domänenkonzepte enthält. So wird beispielsweise die Variable *start_date* auf die Eigenschaft *startDate* des Domänenkonzepts DateInterval abgebildet. Außerdem beinhaltet das Anwendungsmodell eine entsprechende Rollenmodellierung für Angestellte und Manager, die sich aus dem Kollaborationsprozess ergibt.

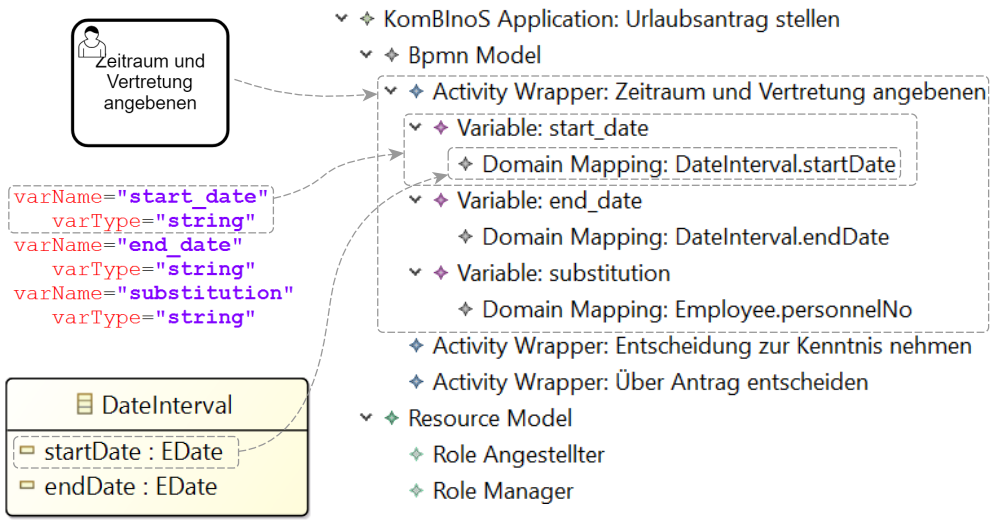


Abbildung 7.5: Beispiel - Anwendungsmodell

7.3.4 Erstellung des Aufgabenmodells

Das Aufgabenmodell dient dazu, Kollaborationsprozesse unabhängig von ihrer formalen Spezifikationssprache in einer aus Sicht der Benutzerschnittstelle einheitlichen und vollständigen Repräsentation darzustellen.

(5) Automatische Generierung des initialen Aufgabenmodells

Analog zu den im vorherigen Abschnitt definierten Mengen $\mathcal{ACT}$, $\mathcal{ACT}_{\mathcal{C}}$ und $\mathcal{ACT}_{\mathcal{N}}$ seien die Mengen $\mathcal{AW}$, $\mathcal{AW}_{\mathcal{C}}$ und $\mathcal{AW}_{\mathcal{N}}$ von ActivityWrappern definiert. Die Generierung des initialen Aufgabenmodells als Transformation **app2tm** (7.9) ergibt sich dann wie folgt:

$$\begin{aligned} \textbf{app2tm} : \mathit{Application} &\longrightarrow \mathit{TaskModel} \\ \mathit{app} &\longmapsto \mathit{tm} \end{aligned} \tag{7.9}$$

mit

$$tm = \begin{bmatrix} \textsc{TaskModel} & \\ \text{id} & app.id + \text{„\_taskmodel“} \\ \text{name} & app.name + \text{„ TaskModel“} \\ \text{wrappedProcess} & \text{app.processModel} \\ \text{nodes} & \left\{ \begin{array}{l} \boxed{A}\ \forall aw \in \mathcal{AW}_{\mathcal{N}}\text{: } aw2task(aw) \cup \\ \boxed{B}\ \forall aw \in \mathcal{AW}_{\mathcal{C}}\text{: } composeTask(aw) \cup \\ \boxed{C}\ \forall var \in app.processModel.variables\text{: } var2data(var) \end{array} \right\} \\ \text{edges} & \left\{ taskFlow,\ taskDependencies,\ dataFlow \right\} \end{bmatrix}$$

Für neue Aktivitäten (Fall $\boxed{A}$) erzeugt die Transformation **aw2task** (7.10) einen neuen AtomicTask, übernimmt relevante Information vom zugehörigen ActivityWrapper, spaltet eingehende und ausgehende Variablen in zwei Mengen auf und erzeugt mit Hilfe der nachgelagerten Transformation **var2data** (7.11) komplexe Datenelemente. Diese Transformation wird auch eingesetzt, um im Fall $\boxed{C}$ Variablen, welche direkt am Prozessmodell hängen, zu überführen. Komponierte Aktivitäten (Fall $\boxed{B}$) werden durch die Transformation **composeTask** (7.12) gesondert betrachtet.

$$\begin{aligned} \textbf{aw2task} : ActivityWrapper &\longrightarrow AtomicTask \\ aw &\longmapsto task \end{aligned} \tag{7.10}$$

mit

$$task = \begin{bmatrix} \textsc{AtomicTask} & \\ \text{id} & aw.id \\ \text{name} & aw.name \\ \text{wrapper} & aw \\ \text{performer} & aw.role \\ \text{agentTask} & aw.agentTask \\ \text{taskConcept} & \boldsymbol{(\rightarrow manuell)} \\ \text{inputData} & \left\{ \begin{array}{l} \forall \{var \in aw.variables \mid var.direction \neq OUT\}\text{:} \\ \quad \boldsymbol{var2data(var,\ self)} \end{array} \right\} \\ \text{outputData} & \left\{ \begin{array}{l} \forall \{var \in aw.variables \mid var.direction \neq IN\}\text{:} \\ \quad \boldsymbol{var2data(var,\ self)} \end{array} \right\} \end{bmatrix}$$

Die Transformation **var2data** (7.11) erzeugt auf Basis der referenzierten Domänenabbildung ein Muster des entsprechenden Objekts. Datenelemente (Data) können im Anschluss nochmals automatisiert aggregiert werden. Dabei muss der Zeitpunkt der Verfügbarkeit eines Datenelements in Bezug auf den Aufgaben-Lebenszyklus (*availableOn*), wie in Kapitel 6.5.2 (S. 121) geschildert, berücksichtigt werden.

Die verbleibenden Muster dienen zur Laufzeit zum Abgleich, ob die von einem Agenten im Dialog angegebene Information vollständig ist. Durch die Verfeinerung der Muster um Einschränkungen (*restrictions*) kann zudem die Konsistenz der Information im Rahmen einer Erwartung überprüft werden. Gleichzeitig kann die Einschränkung des Wertebereichs zum Filtern von Information benutzt werden.

$$\begin{aligned} \textbf{var2data} : Variable \times Task &\longrightarrow Data \\ (var, task) &\longmapsto data \end{aligned} \tag{7.11}$$

mit

$$data = \left[\begin{array}{ll} \textsc{Data} & \\ \text{name} & var.name \\ \text{taskRef} & task \\ \text{availableOn} & \begin{cases} \text{ACTIVE} & falls\ var.direction \neq OUT \\ \text{FINISHED} & falls\ var.direction \neq IN \end{cases} \\ \text{pattern} & \left[\begin{array}{ll} \textsc{PObject} & \\ \text{type} & var.domainMapping.concept \\ \text{slot} & \left[\begin{array}{ll} \textsc{PSlot} & \\ \text{feature} & var.domainMapping.feature \\ \text{range} & \left[\begin{array}{ll} \textsc{PValue} & \\ \text{varName} & var.name \\ \text{type} & var.domainMapping.feature.type \\ \text{restrictions} & \textbf{\textit{(→manuell)}} \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

Die Methode **composeTask** (7.12) dient lediglich zur Bündlung von Kompositionsaktivitäten und zu einer dedizierteren Fallunterscheidung.

$$\begin{aligned} &\textbf{composeTask} : ActivityWrapper \longrightarrow Task \\ &\text{mit } composeTask(aw) = \begin{cases} composeTaskModel(aw) & \text{für } aw \in \mathcal{AW}_{\mathcal{CP}} \\ composeTasks(aw) & \text{für } aw \in \mathcal{AW}_{\mathcal{CA}} \end{cases} \end{aligned} \tag{7.12}$$

Die hierin gekapselten Transformationen ähneln strukturell sehr stark der Transformation **aw2task** (7.10), weshalb im Folgenden lediglich die Unterschiede dazu aufgezeigt werden. So verlinkt die Transformation **composeTaskModel** (7.13) das Aufgabenmodell der in der Kompositionskette befindlichen KomBInoS-UI und klont die bereits aggregierten ein- und ausgehenden Datenobjekte dieses Aufgabenmodells. Das Klonen und damit das Erzeugen neues neuen Datenelements ist hier aufgrund der Modellierung der *inputData*- und *outputData*-Referenzen als Containment-Referenzen angezeigt. Ausgangspunkt für diese Form der Wiederverwendung ist die initiale Orchestrie-

rung eines vollständigen Prozesses, etwa durch eine BpmnCallActivity. Zur Laufzeit wird hierfür eine unabhängige Instanz dieses Prozesses innerhalb einer Prozessausführungsmaschine erzeugt, für die per Konstruktion bereits eine eigenständige KomBInoS-UI existiert. Die geklonten Datenobjekte modellieren somit die Einstiegspunkte in sowie die Rücksprungpunkte aus der unabhängigen KomBInoS-UI in die aktuelle.

$$\begin{aligned} \textbf{composeTaskModel} : ActivityWrapper &\longrightarrow CompositeTask \\ aw &\longmapsto task \end{aligned} \tag{7.13}$$

mit

$$task = \begin{bmatrix} \textsc{CompositeTask} & \\ \text{taskModelRef} & \textit{getInputs(aw).el().taskModel} \\ \text{inputData} & \textit{clone(getInputs(aw).el().taskModel.getInputData())} \\ \text{outputData} & \textit{clone(getInputs(aw).el().taskModel.getOutputData())} \end{bmatrix}$$

Die Transformation **composeTasks** (7.14) erzeugt einen CompositeTask als Container um die zuvor in irgendeiner Form wiederverwendeten Prozessaktivitäten. Hierin werden, analog zur Transformation **composeTaskModel** (7.13), die entsprechenden Aufgaben dieser Prozessaktivitäten verlinkt und die Menge an ein- und ausgehenden Datenobjekten dieser Aufgaben geklont. Dadurch werden auch etwaige Aggregationen der erwarteten Datenmuster gemäß Domänenmodell übernommen.

$$\begin{aligned} \textbf{composeTasks} : ActivityWrapper &\longrightarrow CompositeTask \\ aw &\longmapsto task \end{aligned} \tag{7.14}$$

mit

$$task = \begin{bmatrix} \textsc{CompositeTask} & \\ \text{tasks} & \textit{getInputs(aw).task} \\ \text{inputData} & \textit{clone(getInputs(aw).task.inputData))} \\ \text{outputData} & \textit{clone(getInputs(aw).task.outputData))} \end{bmatrix}$$

Die auf diese Weise durchgeführte Komposition von Aufgaben auf Basis einer ursächlichen Prozesskomposition hat wieder Nebeneffekte auf das Kompositionsmodell. Dieses wird um folgende Aufgabenkompositionen (TaskComposition) ergänzt.

$$\begin{bmatrix} \textsc{TaskComposition} & \\ \text{parent} & \textit{getComposition(aw)} \\ \text{type} & \textit{getComposition(aw).type} \\ \text{inputTaskModel} & \textit{getInputs(aw).el().taskModel} \\ \text{outputTask} & \textit{task} \end{bmatrix} \quad \text{für } aw \in \mathcal{AW}_{\mathcal{CP}}$$

$$\left[\begin{array}{ll} \textsc{TaskComposition} & \\ \text{parent} & \textit{getComposition(aw)} \\ \text{type} & \textit{getComposition(aw).type} \\ \text{inputTasks} & \textit{getInputs(aw).task} \\ \text{outputTask} & \textit{task} \end{array}\right] \quad \text{für } aw \in \mathcal{AW}_{\mathcal{CA}}$$

(6) Manuelle Ergänzung des Aufgabenmodells

Nicht immer ist eine 1-zu-1 Abbildung einer Prozessaktivität auf eine von einem Agenten durchzuführende Aufgabe adäquat, etwa weil die geforderte Handlung oder die geforderte Information sehr umfangreich oder komplex ist. Um dieses Missverhältnis auszugleichen, besteht - eine entsprechende Werkzeugunterstützung vorausgesetzt - für eine neu erzeugte Agentenaufgabe auf Basis eines ActivityWrappers aw, mit $aw \in \mathcal{AW}_{\mathcal{N}}$, immer die Möglichkeit, einen generierten komplexen AtomicTask in einen CompositeTask unter Beibehaltung sämtlicher Abhängigkeiten umzuwandeln. Anschließend kann der Entwickler unter Einsatz seines Expertenwissens die komplexe Aufgabe innerhalb des CompositeTasks über beliebige Stufen in mehrere einfachere Aufgaben hierarchisch dekomponieren. Diese zusätzlich geschaffene Strukturierung bleibt im Rahmen einer späteren Komposition erhalten. Für alle neuen Aufgaben ist weiterhin das TaskConcept gemäß der hierarchischen Charakterisierung von Aufgaben (Kapitel 6.5.2, S. 121) möglichst konkret zu spezifizieren. Es ist auch nicht auszuschließen, dass im Rahmen einer Agentenaufgabe Information vom Prozess bereitgestellt oder verlangt wird, die zwar aus Prozesssicht verständlich ist, jedoch aus Interaktionssicht kaum vermittelt werden kann. Bezogen auf das in Abbildung 7.4 gezeigte Domänenmodell kann ein Prozess etwa für Menschen wenig aussagekräftige Personalnummern verarbeiten. Ein menschlicher Agent möchte im Allgemeinen doch lieber den Namen des Mitarbeiters oder Kollegen lesen oder angeben.

Dies kann zur Entwurfszeit dadurch erreicht werden, dass der Entwickler einen oder mehrere zusätzliche Slots spezifiziert. Für benannte Entitäten (NamedEntity) kann dies durch Berücksichtigung des Slots *name* im Rahmen einer Transformation automatisiert erfolgen. Obwohl nun ein Agent - um bei obigem Beispiel zu bleiben - zur Laufzeit den Namen des Mitarbeiters mitgeteilt hat, ist zum erfolgreichen Abschluss der in Bearbeitung befindlichen Prozessaktivität dennoch zwingend die Personalnummer erforderlich. Diese Information kann in der Laufzeitumgebung durch entsprechende domänenspezifische Dienste als Datenlieferanten abgefragt werden - eine Funktionalität, die von der SiAM-Dialogplattform bei entsprechender Modellierung unterstützt wird (Neßelrath 2016, S. 118).

Ebenso kann der Entwickler plausible Einschränkungen des Wertebereichs der generierten Datenmuster innerhalb der Data-Instanzen auf deklarative Weise vornehmen. Solche Einschränkungen können auch direkt aus syntaktischen OCL-Ausdrücken, welche im Domänenmodell hinterlegt sind, abgeleitet werden - im Allgemeinen jedoch nicht immer automatisiert.

Im Falle einer zugrundeliegenden adaptierten Prozessaktivität ($aw \in \mathcal{AW}_{\mathcal{CAA}}$) muss der Entwickler auf eine etwaig notwendige manuelle Revision hingewiesen werden. Hierbei gilt im Allgemeinen, dass die Überprüfung eines Kompositionsvorschlags incl. einer gegebenenfalls notwendigen Anpassung effizienter ist, als die Durchführung der oben beschriebenen manuellen Ergänzungen neuer Agentenaufgaben. Falls mehrere Prozessaktivitäten zu einer verschmolzen wurden ($aw \in \mathcal{AW}_{\mathcal{CAM}}$), muss aufgrund der im vorherigen Schritt der manuellen Anpassung des Anwendungsmodells durchgeführten Aggregation der Menge an Variablen, die Menge an Data-Instanzen auf die gleiche Weise zusammengefasst werden. Auch hierzu bedarf es eines Hinweises in der Entwicklungsumgebung.

Beispiel: Aufgabenmodell

Abbildung 7.6 (S. 163) illustriert die automatisierte Ableitung eines Aufgabenmodells auf Basis des Prozessmodells.

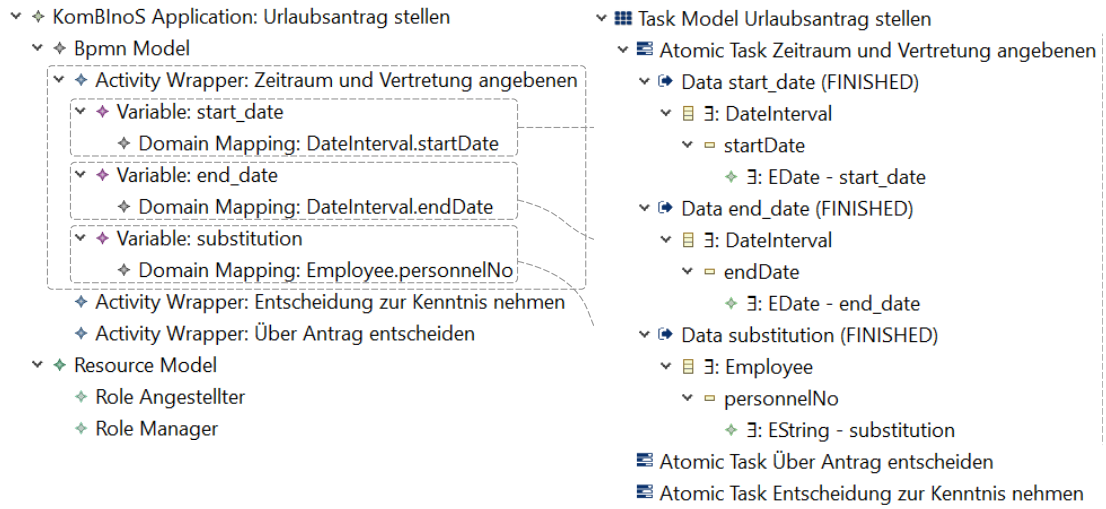


Abbildung 7.6: Beispiel - Aufgabenmodell

Innerhalb der Aufgabe *Zeitraum und Vertretung angeben* werden zu jeder Prozessvariablen und der darin spezifizieren Domänenabbildung automatisiert Muster von entsprechenden Datenobjekten aufgebaut. Abbildung 7.7 zeigt, wie diese Muster anschließend automatisiert aggregiert (links-Mitte) und in einem manuellen Schritt (Mitte-rechts) verfeinert werden. Die Aggregation lässt die Muster *start_date* und *end_date* zu einem Muster *date_interval* verschmelzen. In der Verfeinerung wird dieses zusammen mit dem Muster *substitution* zum Muster *request* unter dem Domänenkonzept ApplicationForLeave zusammengefasst. Um im Beispiel einem Angestellten zur Laufzeit nicht die Personalnummer seiner Urlaubsvertretung abzuverlangen, wird zusätzlich die Eigenschaft *name* in das Muster aufgenommen. Außerdem erfolgt eine Verfeinerung der Verfügbarkeit der Information (*FINISHED* →*SUCCESSFUL*).

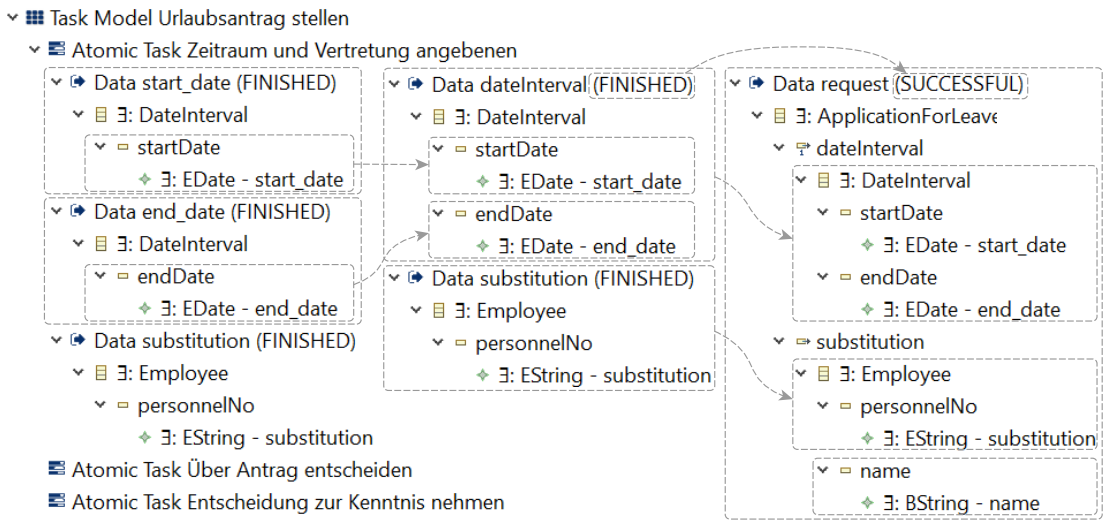


Abbildung 7.7: Beispiel - Aggregation und Verfeinerung von Datenmustern im Aufgabenmodell

7.3.5 Erstellung des Dialogaktmodells

Das Dialogaktmodell stellt als abstraktes UI das Bindeglied dar zwischen den konkreten, modalitätsspezifischen Ausprägungen und dem Kollaborationsprozess des Dienstes. Während die zuvor betrachteten Modelle hierarchisch strukturiert sind, ist das Dialogaktmodell hingegen flach strukturiert. Im Gegenzug besitzen alle darin enthaltenen anwendungsspezifischen Dialogakte Referenzen auf Aufgaben oder Datenelemente für die sie erzeugt wurden. Darüber hinaus können von der Laufzeitumgebung natürlich zusätzliche, anwendungsunabhängige Dialogakte (Kapitel 4.2.3, S. 52), etwa zum Wechsel der Gesprächsinitiative, zur Erfüllung sozialer Obligationen oder zum Metadialog verarbeitet oder erzeugt werden. Deren Spezifikation wird nicht als Bestandteil der Methode aufgefasst. Zur Erstellung des anwendungsspezifischen Dialogaktmodells werden nun lediglich solche Aufgaben des Aufgabenmodells betrachtet, die von einem Agenten durchgeführt werden können. Dabei ist anzumerken, dass die im Folgenden diskutierten Transformationsvorschriften dazu dienen, konkrete Dialogakte zu erzeugen, die im Kern von SiAM-dp als Zielplattform verarbeitet werden können. Dies erfordert Kenntnis über die internen Abläufe bei der Dialogverarbeitung. Zur Laufzeit müssen von der Dialogplattform unifizierbare Dialogakte bzw. Dialogakte nach gleichem Muster verarbeitet bzw. erzeugt werden können. Jenseits dieser Plattformabhängigkeit lassen die Transformationen dennoch ein generelles konzeptuelles Vorgehen bei der Erstellung des Dialogaktmodells erkennen, sodass eine Übertragung auf andere Zielplattformen möglich ist.

(7) Automatische Generierung des Dialogaktmodells
Sei

$$\mathcal{TASK}_{\mathcal{A}} := \{t \in tm.nodes \mid Task(t) \wedge t.agentTask\}$$

die Menge aller Aufgaben im Aufgabenmodell *tm*, die von einem Agenten durchzuführen sind. Sei außerdem

$$\begin{aligned}\mathcal{D}_{\mathcal{IN}} :=& \{d \mid Data(d) \wedge \exists t \in \mathcal{TASK}_{\mathcal{A}} : d \in t.inputData\} \\ & \cup \{d \in tm.nodes \mid Data(d) \wedge d.availableOn = ACTIVE\}\end{aligned}$$

die Menge aller darin eingehenden sowie der zum Prozessstart notwendigen Daten. Dann sei

$$\mathcal{S}_{\mathcal{IN}} := \{s \mid PSlot(s) \wedge \exists d \in \mathcal{D}_{\mathcal{I}} : s \in d.pattern\}$$

die Menge aller Slots der Muster, die die prozessrelevanten Inhalte der Daten beschreiben. Bei tiefer verschachtelten Mustern schließt die Menge $\mathcal{S}_{\mathcal{IN}}$ auch diese Slots mit ein. Aufgrund der Mengenkonstruktion tauchen mehrfach identische Slots unterschiedlicher Data-Elemente nur genau einmal auf. Diese Menge kann weiter nach prozessrelevanten ($\mathcal{S}_{\mathcal{INP}}$) und interaktionsrelevanten ($\mathcal{S}_{\mathcal{INI}}$) Inhalten partitioniert werden:

$$\mathcal{S}_{\mathcal{INP}} := \{s \in \ \mathcal{S}_{\mathcal{IN}} \wedge s.range.varName \neq null\}$$

$$\mathcal{S}_{\mathcal{INI}} := \mathcal{S}_{\mathcal{IN}} / \mathcal{S}_{\mathcal{INP}}$$

Die Mengen $\mathcal{D}_{\mathcal{OUT}}$ und $\mathcal{S}_{\mathcal{OUT}}$ sind analog definiert. Damit lässt sich die Transformation **tm2dia** (7.15) so spezifizieren, dass für deren Elemente *Adjacency Pairs* (Kapitel 4.2.3, S. 52) erzeugt werden, die konform zu den in Kapitel 4.2.5 (S. 56) geschilderten Konversationsmaximen sind. Diese Paare behandeln die Durchführung sowie das Delegieren von Agentenaufgaben und das damit verbundene Einfordern und Abfragen sowie das Preisgeben von Information. Weitere Paare können auf die gleiche Weise ergänzt werden.

$$\begin{aligned}\textbf{tm2dia} : TaskModel \times Application &\longrightarrow DialogueActModel \\ (tm, app) &\longmapsto dia\end{aligned} \tag{7.15}$$

mit

$$
dia = \left[\begin{array}{ll}
\textsc{DialogueActModel} & \\
\text{id} & \textit{app.id} + \text{„\_\_}\textit{dialogueactmodel}\text{“} \\
\text{name} & \textit{app.name} + \text{„ }\textit{DialogueActModel}\text{“} \\
\text{acts} & \left\{\begin{array}{l}
\boxed{A}\ \textit{taskmodel2ia(tm, ProcessManagementFunction)} \cup \\
\quad \textit{taskmodel2oa(tm, ProcessManagementFunction)} \cup \\
\boxed{B}\ \forall\{t \in \mathcal{TASK}_{\mathcal{U}}\}\textit{:} \\
\quad \textit{task2ia(t, TaskManagementFunction)} \cup \\
\quad \textit{task2oa(t, TaskManagementFunction)} \cup \\
\boxed{C}\ \forall\{s \in \mathcal{S}_{\mathcal{IN}}\}\textit{:} \\
\quad \textit{slot2iaSeek(s, Question)} \cup \textit{slot2iaSeek(s, Request)} \cup \\
\quad \textit{slot2oaProv(s, Inform)}\cup \\
\boxed{D}\ \forall\{s \in \mathcal{S}_{\mathcal{OUT}}\}\textit{:} \\
\quad \textit{slot2oaSeek(s, Question)} \cup \textit{slot2oaSeek(s, Request)} \cup \\
\quad \textit{slot2iaProv(s, Inform)} \cup \\
\boxed{E}\ \forall\{s \in \mathcal{S}_{\mathcal{IN}}\}\textit{:} \\
\quad \textit{slot2iaDeictic(s, Question)} \cup \textit{slot2iaDeictic(s, Request)} \cup \\
\boxed{F}\ \forall\{s \in \mathcal{S}_{\mathcal{OUT}}\}\textit{:} \\
\quad \textit{slot2iaDeictic(s, Inform)}
\end{array}\right\}
\end{array}\right]
$$

Im Fall $\boxed{A}$ werden Dialogakte zum Starten und Abbrechen von Prozessen erzeugt. Die Transformationen **taskmodel2ia** und **taskmodel2oa** entsprechen für relevante kommunikative Funktionen des Prozessmanagements gemäß Kapitel 6.6 (S. 126) im Wesentlichen den Transformationen **task2ia** (7.16) und **task2oa** (7.17) im Fall $\boxed{B}$, weshalb auf eine Darstellung verzichtet wird. Im Fall $\boxed{B}$ werden für alle Agentenaufgaben Dialogakte für die Verwaltung (z. B. UnclaimTask), Durchführung (z. B. PerformTask) und Benachrichtigung (z. B. TaskPerformed) erstellt.

$$
\begin{array}{c}
\textbf{task2ia} : Task \times EClass \longrightarrow InputAct \\
(task, cfClass) \longmapsto da
\end{array}
\tag{7.16}
$$

mit

$$
da = \left[\begin{array}{ll}
\textsc{InputAct} & \\
\text{roleRef} & \textit{task.performer} \\
\text{taskRef} & \textit{task} \\
\text{hypothesis} & \left[\begin{array}{ll}
\textsc{Hypotheses} & \\
\text{comFct} & \left[\begin{array}{ll}
\text{«cfClass»} & \\
\text{semContent} & \left[\begin{array}{ll}
\textsc{SemanticContent} & \\
\text{resolved} & \textit{RESOLVED} \\
\text{content} & \left[\begin{array}{ll}\textsc{Task} & \\ \text{id} & \textit{task.id}\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

$$
\begin{aligned}
\textbf{task2oa} : Task \times EClass &\longrightarrow OutputAct \\
(task, cfClass) &\longmapsto da
\end{aligned}
\tag{7.17}
$$

mit

$$
da = \left[\begin{array}{ll}
\textsc{OutputAct} & \\
\text{roleRef} & \textit{task.performer} \\
\text{taskRef} & \textit{task} \\
\text{comFct} & \left[\begin{array}{ll}
\text{«cfClass»} & \\
\text{semContent} & \left[\begin{array}{ll}
\textsc{SemanticContent} & \\
\text{resolved} & \textit{RESOLVED} \\
\text{content} & \left[\begin{array}{ll}\textsc{Task} & \\ \text{id} & \textit{task.id}\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Im Fall $\boxed{C}$ werden für alle vom Prozess zur Verfügung gestellten Informationseinheiten Dialogakte erzeugt, der Fall $\boxed{D}$ behandelt auf die gleiche Weise Informationseinheiten, die in den Prozess einfließen müssen. Hier werden nun die Transformationen **slot2iaSeek** (7.18) und **slot2oaSeek** zum Erzeugen von informationsabfragenden Eingabe- und Ausgabeakten sowie **slot2iaProv** (7.19) und **slot2oaProv** zum Erzeugen von informationsliefernden Eingabe- und Ausgabeakten verwendet. Die Transformationen **slot2oaSeek** und **slot2oaProv** sind bis auf die eingeschobene Hypothese analog zu ihren Pendants auf Eingabeseite aufgebaut, weshalb sie nicht gesondert aufgeführt werden.

$$
\begin{aligned}
\textbf{slot2iaSeek} : PSlot \times EClass &\longrightarrow InputAct \\
(slot, cfClass) &\longmapsto da
\end{aligned}
\tag{7.18}
$$

mit

$$da = \begin{bmatrix} \textsc{InputAct} & \\ \text{slotRef} & \textit{slot} \\ \text{hypothesis} & \begin{bmatrix} \textsc{Hypotheses} & \\ \text{comFct} & \begin{bmatrix} \text{«cfClass»} & \\ \text{semContent} & \textit{slot2ref(slot,SlotReference)} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{aligned} \textbf{slot2iaProv} : PSlot \times EClass &\longrightarrow InputAct \\ (slot, cfClass) &\longmapsto da \end{aligned} \tag{7.19}$$

mit

$$da = \begin{bmatrix} \textsc{InputAct} & & \\ \text{slotRef} & \textit{slot} & \\ \dots & \begin{bmatrix} \textsc{Hypothesis} & \\ \dots & \begin{bmatrix} \text{«cfClass»} & \\ \text{semContent} & \begin{cases} \textit{slot2content(slot)} & \textit{falls } s \in \mathcal{S}_{\mathcal{INP}} \\ \textit{slot2ref(slot,} & \textit{falls } s \in \mathcal{S}_{\mathcal{INI}} \\ \quad \textit{KnowledgeBaseReference)} & \end{cases} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

In der Transformation **slot2iaProv** (7.19) wird unterschieden zwischen prozess- und interaktionsrelevanten Slots. Für erstere wird mit Hilfe der nachgelagerten Transformation **slot2content** (7.20) zur Entwurfszeit ein Muster spezifiziert, welches beschreibt, wie eine konkrete Instanz zur Laufzeit aussieht. Für letztere wird zur Laufzeit zusätzlich eine Referenzauflösung anhand einer Wissensbasis notwendig. Die dazu verwendete KnowledgeBaseReference wird mittels der Hilfstranformation **slot2ref** (7.21) erzeugt, welche auf gleichem Wege in **slot2iaSeek** (7.18) eine SlotReference zur Spezifikation eines gesuchten Informationsbausteins erzeugt.

$$\begin{aligned} \textbf{slot2content} : PSlot &\longrightarrow SemanticContent \\ slot &\longmapsto content \end{aligned} \tag{7.20}$$

mit

$$content = \begin{bmatrix} \textsc{SemanticContent} & \\ \text{resolved} & \textit{RESOLVED} \\ \text{contentPattern} & \begin{bmatrix} \textsc{PObject} & \\ \text{type} & \textit{slot.feature.eContainingClass} \\ \text{slot} & \textit{clone(slot)} \end{bmatrix} \end{bmatrix}$$

$$\begin{aligned} \textbf{slot2ref} : PSlot \times EClass &\longrightarrow SemanticContent \\ (slot, refClass) &\longmapsto content \end{aligned} \tag{7.21}$$

mit

$$content = \begin{bmatrix} \textsc{SemanticContent} & \\ \text{resolved} & \textit{UNRESOLVED} \\ \text{reference} & \begin{bmatrix} \text{«refClass»} & \\ \text{referencePattern} & \begin{bmatrix} \textsc{PObject} & \\ \text{type} & \textit{slot.eContainer.eType} \\ \text{slot} & \textit{clone(slot)} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Für die Fälle $\boxed{E}$ und $\boxed{F}$ erzeugt die Transformation **slot2iaDeictic** (7.22) einen InputAct, der zur Laufzeit entweder als anaphorische oder deiktische Referenz im Kontext klassifiziert und somit mit Hilfe der multimodalen Fusion oder im Diskurs aufgelöst werden muss. Die Gleichbehandlung dieser Referenzarten zur Entwurfszeit vereinfacht die Modellierung und unterstützt eine synergistische Multimodalität der Dialogschnittstelle. Nach gleichem Schema können weitere von SiAM-dp unterstützte Referenzarten (Neßelrath 2016, S. 118) abgedeckt werden.

$$\begin{aligned} \textbf{slot2iaDeictic} : PSlot \times EClass &\longrightarrow InputAct \\ (slot, cfClass) &\longmapsto da \end{aligned} \tag{7.22}$$

mit

$$da = \begin{bmatrix} \textsc{InputAct} & \\ \text{slotRef} & \textit{slot} \\ \text{hypothesis} & \begin{bmatrix} \textsc{Hypotheses} & \\ \text{comFct} & \begin{bmatrix} \text{«cfClass»} & \\ \text{semContent} & \textit{slot2ref(slot,DeicticReference)} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Der Unterschied zwischen einer SlotReference und einer DeicticReference mit identischem Muster liegt z. B. bei der Frage nach dem Geburtstag einer Person darin, dass eine Äußerung im ersten Fall „*Wann ist Konrad Zuse geboren?*“ und im zweiten Fall „*Wann ist er geboren?*“ lauten kann. Die Äußerung im zweiten Fall wird z. B. beim Fehlen einer Zeigegeste als Anapher klassifiziert und behandelt.

(8) Manuelle Ergänzung des Dialogaktmodells

Das soeben erzeugte Dialogaktmodell ist in Bezug auf das zugrundeliegende Aufgabenmodell *konsistent* und *vollständig*. Um diese Eigenschaften nicht zu verletzen, ist dem Entwickler von manuellen Anpassungen abzuraten. In der Werkzeugunterstützung kann dies etwa durch Verweigern der Schreibrechte auf der Modelldatei erzwungen werden. In der Konsequenz ergibt sich, dass die Wiederverwendung von Dialogakten im Rahmen einer Komposition nicht angezeigt ist.

Beispiel: Dialogaktmodell

Abbildung 7.8 zeigt ausschnittsweise die Menge an Dialogakten, die für die Aufgabe *Zeitraum und Vertretung angeben* erzeugt werden.

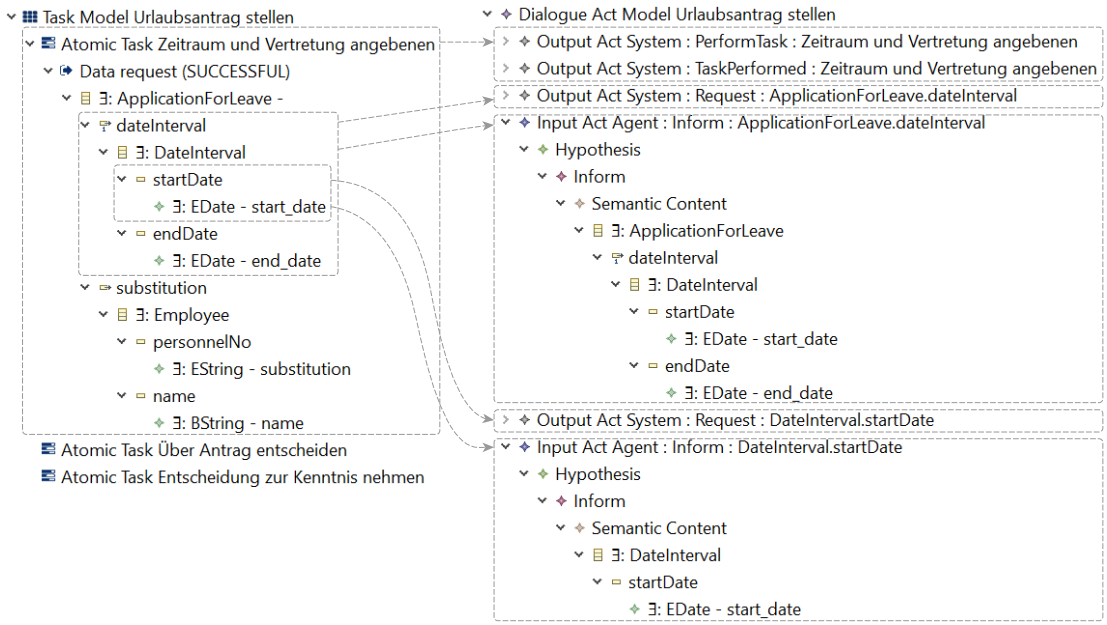


Abbildung 7.8: Beispiel - Dialogaktmodell

Es werden aufgabenspezifische Ausgabeakte erzeugt, die dem durchführenden Agenten die Aufgabenstellung (*System : PerformTask : ...*) kommunizieren und ihn informieren, wenn die Aufgabe ausgeführt wurde (*System : TaskPerformed : ...*). Des Weiteren werden Dialogakte zum Abfragen und Preisgeben von Information erzeugt. Hierbei ist festzustellen, dass - obwohl die Dialogakte im Dialogaktmodell sequenziell und damit flach angeordnet sind - die Informationsstrukturierung der Muster in den semantischen Inhalten sehr wohl eine Tiefe gemäß des zugrundeliegenden Domänenmodells aufweist. So fragt der Ausgabeakt *System : Request : ApplicationForLeave.dateInterval* den gewünschten Urlaubszeitraum ab, wohingegen der Ausgabeakt *System : Request : DateInterval.startDate* lediglich das Startdatum als die erste Komponente der DateInterval-Struktur abfragt. Umgekehrt liefert der Eingabeakt *Agent : Inform : ApplicationForLeave.dateInterval* ein vollstädiges Datumsintervall, während der Eingabeakt *Agent : Inform : DateInterval.startDate* nur den Beginn liefert.

7.3.6 Erstellung des Spracheingabemodells

Im vorangegangenen Schritt der Erzeugung des Dialogaktmodells wurde die Menge der im Prozess erfolgreich verarbeitbaren Dialogakte, insbesondere der Eingabeakte ausmultipliziert dargestellt. Das Ziel dieses Schrittes ist es nun, im übertragenen Sinne ein

Spracheingabemodell als Stammfunktion zu integrieren, sodass daraus alle Eingabeakte abgeleitet werden können. Aus diesem Modell wird dann zur Laufzeit eine von einem Spracherkenner verarbeitbare Grammatik erzeugt.

(9) Automatische Generierung des initialen Spracheingabemodells
Sei

$$\mathcal{IA} := \{act \in dia.acts \mid InputAct(act)\}$$

die Menge aller InputActs im Dialogaktmodell *dia*. Weiter enthalten die Mengen

$$\mathcal{IA}_{\mathcal{P}} := \{act \in \mathcal{IA} \mid act.taskModelRef \neq null\}$$

$$\mathcal{IA}_{\mathcal{T}} := \{act \in \mathcal{IA} \mid act.taskRef \neq null\}$$

$$\mathcal{IA}_{\mathcal{S}} := \{act \in \mathcal{IA} \mid act.slotRef \neq null\}$$

alle prozess-, aufgaben- oder datenbezogenen Eingabeakte. Aufgrund der Generierungsvorschrift der datenbezogenen Eingabeakte, existieren Akte mit den kommunikativen Funktionen Question, Request und Inform, sodass die Menge $\mathcal{IA}_{\mathcal{S}}$ weiter partioniert werden kann:

$$\mathcal{IA}_{\mathcal{SQ}} := \{act \in \mathcal{IA}_{\mathcal{S}} \mid Question(act.hypotheses.el().communicativeFunction)\}$$

$$\mathcal{IA}_{\mathcal{SR}} := \{act \in \mathcal{IA}_{\mathcal{S}} \mid Request(act.hypotheses.el().communicativeFunction)\}$$

$$\mathcal{IA}_{\mathcal{SI}} := \{act \in \mathcal{IA}_{\mathcal{S}} \mid Inform(act.hypotheses.el().communicativeFunction)\}$$

Schließlich sei

$$\mathcal{IA}_{\mathcal{SE}} := \{act \in \mathcal{IA}_{\mathcal{S}} \mid kindOf(act.slotRef.feature.eType, Entity)\}$$

die Menge aller Eingabeakte, die ein Domänenkonzept referenzieren. Die Menge

$$\mathcal{E}_{\mathcal{S}} := \{act.slotRef.feature.eType \mid act \in \mathcal{IA}_{\mathcal{SE}}\}$$

enthält alle in Slots referenzierten Domänenkonzepte. Die zuvor definierten Mengen können verschnitten werden zu

$$\mathcal{IA}_{\mathcal{SQE}} := \mathcal{IA}_{\mathcal{SE}} \cap \mathcal{IA}_{\mathcal{SQ}}$$

$$\mathcal{IA}_{\mathcal{SQA}} := \mathcal{IA}_{\mathcal{SA}} / \mathcal{IA}_{\mathcal{SQE}}$$

Die Menge

$$\mathcal{E}_{\mathcal{SQ}} := \{act.slotRef.feature.eType \mid act \in \mathcal{IA}_{\mathcal{SQE}}\}$$

bezeichnet die Menge aller Domänenkonzepte, zu denen Eingabeakte mit der kommunikativen Funktion Question existieren. Die Mengen $\mathcal{IA}_{\mathcal{SRE}}$ und $\mathcal{E}_{\mathcal{SR}}$ sowie $\mathcal{IA}_{\mathcal{SIE}}$ und

$\mathcal{E}_{\mathcal{SI}}$ sind analog konstruiert. Die initiale Transformation *dia2speechin* (7.23) zur Erzeugung des Spracheingabemodells ist dann wie folgt definiert:

$$\begin{aligned} \textbf{dia2speechin} : DialogueActModel \times Application &\longrightarrow SpeechInModel \\ (dia, app) &\longmapsto sin \end{aligned} \tag{7.23}$$

mit

$$sin = \left[\begin{array}{ll} \textsc{SpeechInModel} & \\ \text{language} & \text{„}de\text{-}DE\text{“} \\ \text{identifier} & app.name + \text{„}\_grammar\text{“} \\ \text{rules} & \left\{\begin{array}{l} \boxed{A}\ \forall\{ia \in \mathcal{IA}_{\mathcal{P}}\}\text{: } ia2utt(ia) \cup \\ \boxed{B}\ \forall\{ia \in \mathcal{IA}_{\mathcal{T}}\}\text{: } ia2utt(ia) \cup \\ \boxed{C}\ \forall\{e \in \mathcal{E}_{\mathcal{SQ}}\}\text{: } cfEntity2utt(Question,e) \cup \\ \boxed{D}\ \forall\{e \in \mathcal{E}_{\mathcal{SQ}}\}\text{: } cfEntity2smr(Question,e) \cup \\ \boxed{E}\ \forall\{e \in \mathcal{E}_{\mathcal{S}}\}\text{: } entity2ent(e) \end{array}\right\} \end{array}\right]$$

Prozess- und aufgabenbezogene Eingabeakte der Fälle $\boxed{A}$ und $\boxed{B}$ können durch die Transformation *ia2utt* (7.24) sofort in Äußerungen transformiert werden. Die Phrasen einer Äußerung müssen anschließend manuell oder unterstützt durch fallbasiertes Schließen (CBR, Case-Based Reasoning) ergänzt werden.

$$\begin{aligned} \textbf{ia2utt} : InputAct &\longrightarrow Utterance \\ ia &\longmapsto utt \end{aligned} \tag{7.24}$$

mit

$$utt = \left[\begin{array}{ll} \textsc{Utterance} & \\ \text{name} & ia.name + \text{„}\_UTT\text{“} \\ \text{dialogueActRef} & ia \\ \text{interpretation} & clone(ia.hypotheses.el().communicativeFunction) \\ \text{phrases} & \textbf{(}\rightarrow\textbf{manuell oder mittels CBR)} \end{array}\right]$$

Es verbleiben die datenbezogenen Eingabeakte, die durch den folgenden Aufbau des Spracheingabemodells alle Berücksichtigung finden. Aufgrund der Konstruktion des Spracheingabe-Metamodells, kann die kommunikative Funktion einer natürlichsprachlichen Äußerung nur durch eine Instanz vom Typ Utterance transportiert werden. Darum muss für jede auftretende Kombination aus kommunikativer Funktion und Domänenkonzept eine solche Utterance erzeugt werden. Dies wird im Fall $\boxed{C}$ durch die Transformation *cfEntity2utt* (7.25) für die kommunikative Funktion Question erreicht. Für alle anderen kommunikative Funktionen ist dies analog durchzuführen. Eine Utterance

referenziert in ihren textuellen Phrasen andere nachgelagerte Regeln, etwa eine EntityRule oder eine SemanticMappingRule, unter Verwendung des Namens dieser Regel mit vorangestelltem $-Zeichen, z. B. „*Wann wurde $PERSON_ENT geboren?*“ oder einfach „*$QUESTION_PERSON_SMR*“. Die Frage nach dem Geburtsdatum einer Person ist dann als PhraseMapping innerhalb der SemanticMappingRule hinterlegt.

$$\begin{aligned} \textbf{cfEntity2utt} : EClass \times EClass &\longrightarrow UtteranceRule \\ (cfClass, eClass) &\longmapsto utt \end{aligned} \tag{7.25}$$

mit

$$utt = \begin{bmatrix} \textsc{Utterance} & \\ \text{name} & cfClass.toString() + \text{„\_“} + eClass.toString() + \text{„\_UTT“} \\ \text{interpretation} & \begin{bmatrix} \text{«}\textsc{cfClass}\text{»} & \\ \text{semanticContent} & \begin{bmatrix} \textsc{SemanticContent} & \\ content & \left[\text{«}\textsc{eClass}\text{»}\right] \end{bmatrix} \end{bmatrix} \\ \text{phrases} & \textbf{\textit{(}}\rightarrow\textbf{\textit{manuell oder mittels CBR)}} \end{bmatrix}$$

Eine SemanticMappingRule ermöglicht technisch die Abbildung verschiedener Informationsbausteine auf ein und dieselbe Grundstruktur. Die Erzeugung einer SemanticMappingRule durch die Transformation *cfEntity2sem* (7.26) im Fall $\boxed{D}$ erfolgt ähnlich zur vorangegangenen Regel. Die Auswertung zur Laufzeit verlangt, dass der Inhalt des Attributs *mappingTarget* mit dem Inhalt des Attributs *semanticContent* einer referenzierenden Utterance unifizierbar ist.

$$\begin{aligned} \textbf{cfEntity2sem} : EClass \times EClass &\longrightarrow SemanticMappingRule \\ (cfClass, eClass) &\longmapsto rule \end{aligned} \tag{7.26}$$

mit

$$rule = \begin{bmatrix} \textsc{SemanticMappingRule} & \\ \text{name} & cfClass.toString() + \text{„\_“} + eClass.toString() + \text{„\_SMR“} \\ \text{mappingTarget} & \begin{bmatrix} \textsc{SemanticContent} & \\ \text{content} & \left[\text{«}\textsc{eClass}\text{»}\right] \end{bmatrix} \\ \text{phraseMappings} & \left\{ \begin{array}{l} \forall \{ ia \in \mathcal{IA}_{\mathcal{S}} \mid \\ \quad ia.slotRef.feature.eContainingClass = eClass \\ \quad \wedge\ typeOf(cfClass, ia.hypotheses.el().comFct) \\ \}: ia2phraseMapping(ia) \end{array} \right. \end{bmatrix}$$

Die Menge an PhraseMapping-Instanzen wird mit Hilfe der nachgelagerten Transformation *ia2phraseMapping* (7.27) gefüllt. Diese Transformation wird auf alle Eingabeakte angewandt, die der fixierten kommunikativen Funktion *cfClass* entsprechen und auf

Slots des fixierten Domänenkonzepts *eClass* abzielen. Somit sind in dieser Menge auch PhraseMappings enthalten, die in ihrer Interpretation auf noch aufzulösende semantische Inhalte verweisen, z. B. in Form von deiktischen Referenzen.

$$\begin{aligned} \textbf{ia2phraseMapping} : InputAct &\longrightarrow PhraseMapping \\ ia &\longmapsto rule \end{aligned} \tag{7.27}$$

mit

$$rule = \begin{bmatrix} \textsc{PhraseMapping} & \\ \text{interpretation} & clone(ia.hypotheses.el().comFct.semanticContent) \\ \text{phrases} & \textbf{\textit{(}}\rightarrow\textbf{\textit{manuell oder mittels CBR)}} \end{bmatrix}$$

Schließlich werden im Fall $\boxed{E}$ für alle verwendeten Domänenkonzepte EntityRules erzeugt, um Instanzen gleichen Typs, z. B. Personen, durch einen deskriptiven Namen auf eine eindeutige Id abbilden zu können. Der Wert *value* wird in der *entries*-Liste spezifiziert.

$$\begin{aligned} \textbf{entity2rule} : EClass &\longrightarrow EntityRule \\ eClass &\longmapsto rule \end{aligned} \tag{7.28}$$

mit

$$rule = \begin{bmatrix} \textsc{EntityRule} & \\ \text{name} & eClass.toString() + \text{„}\_ENT\text{“} \\ \text{type} & EntityType::dynamic \\ \text{interpretation} & \begin{bmatrix} \textsc{SemanticContent} & \\ \text{content} & \begin{bmatrix} \text{«}\textsc{eClass}\text{»} \\ \text{id} \quad \text{„}\$expr(value)\text{“} \end{bmatrix} \end{bmatrix} \\ \text{entries} & \textbf{\textit{(}}\rightarrow\textbf{\textit{manuell, mittels CBR oder zur Laufzeit)}} \end{bmatrix}$$

(10) Manuelle Ergänzung des Spracheingabemodells

Das Erzeugen von Sprachmodellen, insbesondere Modellen für die Spracheingabe, erfordert aufgrund der hohen Varianz an möglichen Nutzeräußerungen viel Zeit. Im Rahmen dieses Vorgehens wird fallbasiertes Schließen eingesetzt, um für die entsprechend markierten Stellen in den Transformationen adäquate Äußerungen anhand der zur Verfügung stehenden semantischen Interpretation zu beziehen. Hierbei ist zu beachten, dass auf diesem Wege aufgrund der hohen Ähnlichkeit der Interpretationen komponierter Aufgaben und Informationsbruchstücke sowie des strukturell identischen Aufbaus der Spracheingabemodelle der KomBInoS-UIs in der Kompositionskette dessen Äußerungen mit einer erwartbaren hohen Konfidenz oder Ähnlichkeit auch ohne Eintrag im Kompositionsmodell wiederverwendet werden. Darüber hinaus kann dieser Ansatz bei vollständig neuen, nicht komponierten Benutzerschnittstellen genutzt werden, auch

wenn die erwartbaren Ähnlichkeiten im Allgemeinen geringer ausfallen. Über einen speziellen Editor können die bezogenen Äußerungen nachjustiert und die Fallbasis zurück gespielt werden. Der technische Prozess des fallbasierten Schließens wird in Kapitel 8.6 (S. 196) im Detail erläutert.

Beispiel: Spracheingabemodell

Abbildung 7.9 illustriert beispielhaft die Generierung des Spracheingabemodells für die natürlichsprachliche Angabe des gewünschten Abwesenheitszeitraums.

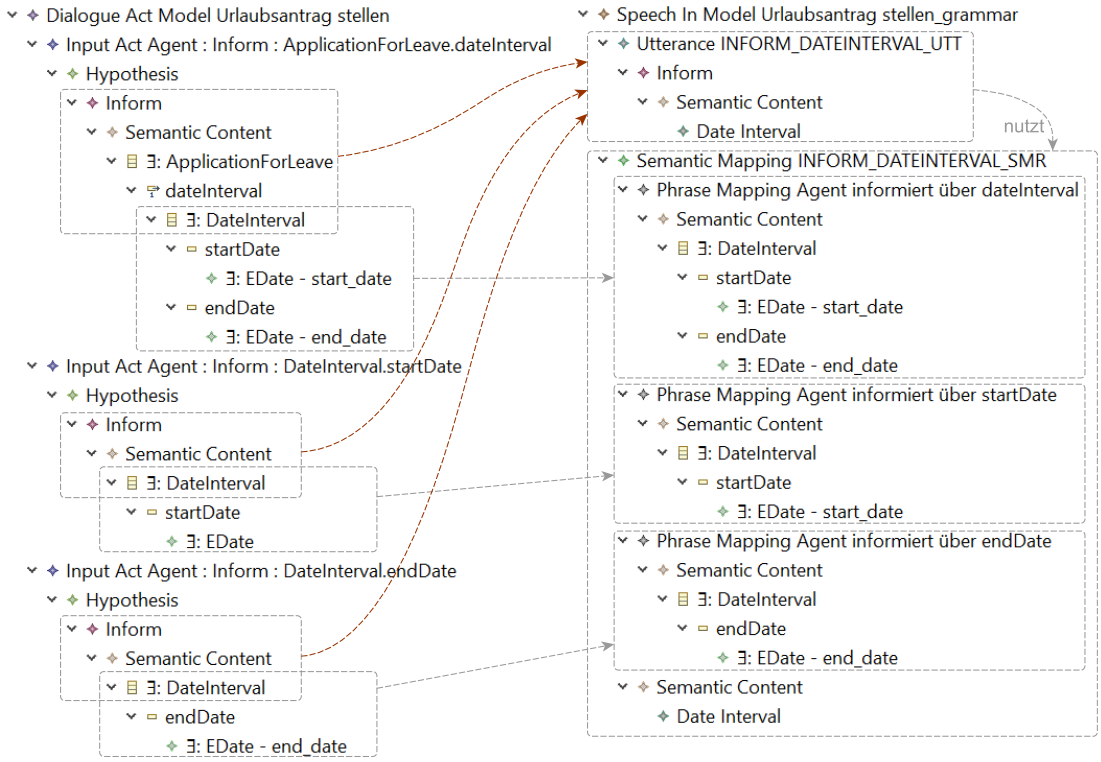


Abbildung 7.9: Beispiel - Spracheingabemodell

Während der erste Eingabeakt des Dialogaktmodells auf der linken Seite das gesamte Datumsintervall abdeckt, fokussieren die beiden anderen auf jeweils ein Attribut des Domänenkonzepts. Alle drei Dialogakte werden im Spracheingabemodell rechts durch die automatisch generierte Äußerung *INFORM_DATEINTERVAL_UTT* subsumiert (rote Pfeile), da die kommunikative Funktion (Inform) übereinstimmt und der semantische Inhalt der kommunikativen Funktion per Kontruktion in Abhängigkeit von der zur Laufzeit getätigten Äußerung mit einem Muster im Dialogaktmodell übereinstimmt (graue Pfeile). Hierzu nutzt die Äußerung die semantische Abbildungsregel *INFORM_DATEINTERVAL_SMR*, welche entsprechende Teilphrasen hierarchisch auf Teilstrukturen des im Fokus befindlichen Domänenkonzepts abbildet. Eine semantische

Abbildungsregel sammelt zu einem Domänenkonzept also spezifische Eingabeakte mit derselben kommunikativen Funktion auf. Die so aufgebaute Hierarchie kann zur Laufzeit zum adaptiven Multi-Slot-Filling (Kapitel 4.3.4, S. 64) komplexer Domänenobjekte ausgenutzt werden. Die nachfolgende Beispielgrammatik in Backus-Naur-Form verdeutlicht dies. Die Äußerung „*Ich möchte ab 22.12. in Urlaub gehen.*“ liefert lediglich den Beginn des Urlaubszeitraums, während die Äußerung „*Ich möchte von 22.12. bis 09.01. Urlaub machen.*“ ein vollständiges Datumsintervall mit zwei Slots zum Füllen der Schablone liefert. Eine Rückfrage nach dem Ende des Abwesenheitszeitraums kann in diesem Fall unterbleiben. Es wird auch deutlich, dass die semantische Abbildungsregel zur Erfassung eines Zeitraums in anderen Kontexten wiederverwendet werden kann bzw. hier bereits wiederverwendet wurde.

$$
\begin{aligned}
INFORM\_UTT &:= \textit{Ich möchte \$INFORM\_SMR ?in Urlaub (gehen|sein|manchen).}\\
INFORM\_SMR &:= \textit{\$PM\_A | \$PM\_B | \$PM\_C}\\
PM\_A &:= \textit{von \$DATUM<startDate> bis \$DATUM<endDate>}\\
PM\_B &:= \textit{ab \$DATUM<startDate>}\\
PM\_C &:= \textit{bis \$DATUM<endDate>}\\
DATUM &:= \textit{im Spracherkenner enthaltene Teilgrammatik}
\end{aligned}
$$

7.3.7 Erstellung des Sprachausgabemodells

Das Sprachausgabemodell besteht aus einer Menge von Sprachsynthesevorlagen. Eine solche Vorlage beschreibt eine Regel, die unter einer bestimmten Bedingung einen Effekt ausübt. Als eine Regelmenge ist das Sprachausgabemodell flach strukturiert. Die Regelselektion und Konfliktauflösung zur Laufzeit ist in SiAM-dp bewusst einfach gehalten und geschieht nach dem Prinzip „*First in First Served*“ (Neßelrath 2016, S. 120). Da die automatische Generierung des initialen Sprachausgabemodells auch technisch auf Mengen beruht, kann hier jedoch keine bestimmte Reihenfolge von Sprachsynthesevorlagen erzwungen werden. Diesem Laufzeitverhalten kann durch eine intelligentere Selektionsstrategie oder eine nachgelagerte automatische Sortierung gemäß abnehmender Komplexität der Bedingung begegnet werden.

(11) Automatische Generierung des initialen Sprachausgabemodells
Sei

$$\mathcal{OA} := \{act \in dia.acts \mid OutputAct(act)\}$$

die Menge aller Ausgabeakte (OutputAct) im Dialogaktmodell *dia*. Dann wird das initiale Sprachausgabemodell durch die Transformation *dia2speechout* (7.29) wie folgt erzeugt.

$$
\begin{aligned}
\textbf{dia2speechout} : DialogueActModel \times Application &\longrightarrow SpeechOutModel\\
(dia, app) &\longmapsto sout
\end{aligned}
\tag{7.29}
$$

mit

$$sout = \begin{bmatrix} \textsc{SpeechOutModel} \\ \text{rules} \quad \left\{ \forall \{oa \in \mathcal{OA}\}: \ oa2synthesis(oa) \right\} \end{bmatrix}$$

Für jeden Ausgabeakt wird mit Hilfe der Transformation *oa2synthesis* (7.30) eine Synthesevorlage erzeugt. Darin beschreibt die Bedingung ein Muster der kommunikativen Funktion samt semantischem Inhalt. Dieses Muster wird durch die Hilfsmethode *createPattern* erzeugt, welche technisch als Blackbox in Java implementiert ist (Kapitel 8.5, S. 194).

$$\begin{aligned} \textbf{oa2synthesis} : OutputAct &\longrightarrow SynthesisTemplate \\ oa &\longmapsto template \end{aligned} \tag{7.30}$$

mit

$$template = \begin{bmatrix} \textsc{SynthesisTemplate} & \\ \text{condition} & createPattern(oa.communiativeFunction) \\ \text{target} & \begin{bmatrix} \textsc{SpeechSynthesis} & \\ \text{ssmlDoc} & \begin{bmatrix} \textsc{Speak} & \\ \text{text} & \textbf{\textit{(}}\rightarrow \textbf{\textit{manuell oder mittels CBR)}} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

(12) Manuelle Ergänzung des Sprachausgabemodells

Zur initialen Spezifikation von Systemäußerungen wird analog zur manuellen Ergänzung des Spracheingabemodells fallbasiertes Schließen eingesetzt. In Kapitel 8.6 (S. 196) ist das Vorgehen im Rahmen der Werkzeugunterstützung im Detail erläutert.

Beispiel: Sprachausgabemodell

Abbildung 7.10 zeigt die beispielhafte Erstellung des Sprachausgabemodells.

Ausgabeakte zur Informationsabfrage werden hierbei auf regelbasierte Synthesevorlagen abgebildet. Die kommunikative Funktion eines Ausgabeaktes im Dialogaktmodell auf der linken Seite wird in ein konformes Muster umgewandelt, welches im Sprachausgabemodell rechts als Bedingung einer Synthesevorlage dient (rote Pfeile). Der semantische Inhalt ist Bestandteil des Musters, wird jedoch aufgrund technischer Vorgaben der Laufzeitumgebung als zeichenkettenbasiertes *knowledgeItem* repräsentiert.

7.3.8 Erstellung des GUI-Modells

Das GUI-Modell liefert den Ausgangspunkt für eine spätere Codegenerierung der finalen GUI. Außerdem dient das GUI-Modell zur Laufzeit auch als Repräsentation des Anzeigekontextes. Vor diesem Hintergrund genügt eine hinreichende Detaillierung des GUI-Modells, die beiden Anforderungen gerecht wird.

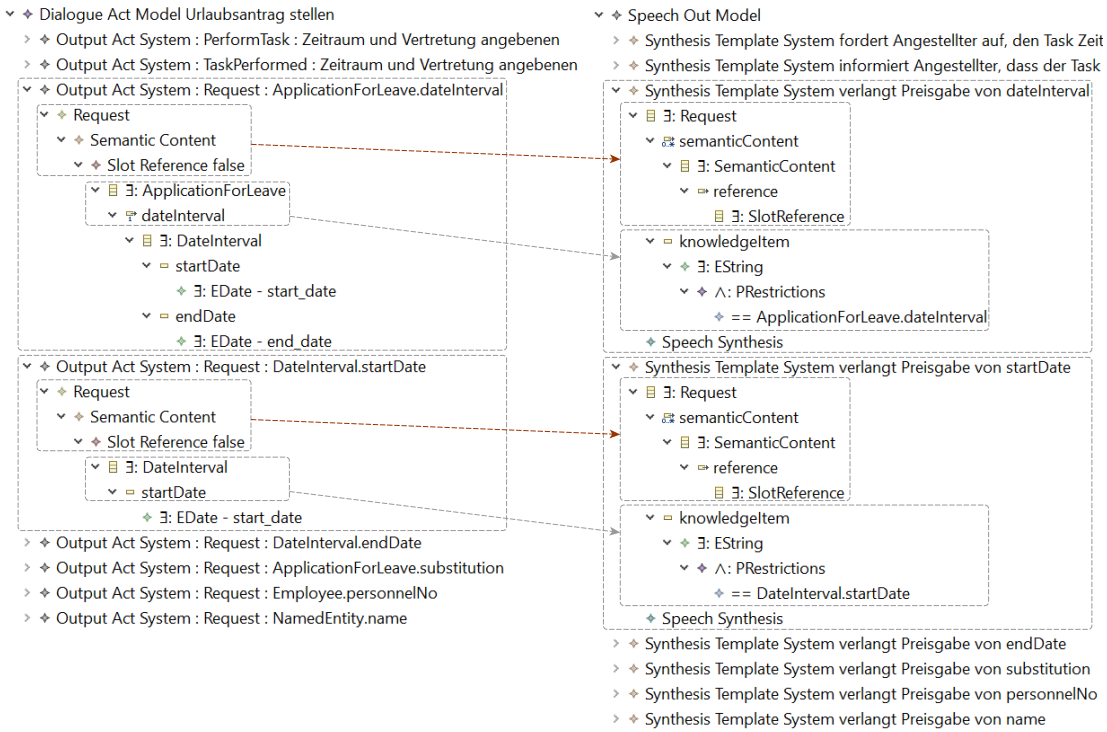


Abbildung 7.10: Beispiel - Sprachausgabemodell

(13) Automatische Generierung des initialen GUI-Modells

Die Generierung des GUI-Modells (GuiModel) erfolgt hauptsächlich auf Basis des Aufgabenmodells, da dies einerseits aufgrund des ebenfalls hierarchischen Aufbaus des GUI-Modells im Gegensatz zum flachen Dialogaktmodell günstiger ist. Andererseits berücksichtigt dieses Vorgehen die logischen und temporalen Zusammenhänge bei der aufgabenzentrierten Interaktion besser.

Die initiale Transformation *app2guimodel* (7.31) ist wiefolgt definiert.

$$\begin{aligned} \textbf{app2guimodel} : TaskModel \times Application &\longrightarrow GuiModel \\ (tm, app) &\longmapsto gm \end{aligned} \tag{7.31}$$

mit

$$gm = \begin{bmatrix} \textsc{GuiModel} \\ \text{applications} \quad \left\{ \forall \{role \in app.resourceModel.roles\} : \ role2gui(role) \right\} \end{bmatrix}$$

Im nächsten Schritt erzeugt die Transformation *role2gui* (7.32) im GUI-Modell für

alle Rollen eine rollenspezifische grafische Benutzerschnittstelle (GuiApplication).

$$\begin{aligned} \textbf{role2gui} : Role &\longrightarrow GuiApplication \\ role &\longmapsto gui \end{aligned} \tag{7.32}$$

mit

$$gui = \left[\begin{array}{ll} \textsc{GuiApplication} & \\ \text{title} & app.name + \text{„(“} + role.name + \text{„)“} \\ \text{vModels} & \left\{\begin{array}{l} taskmodel2vm(tm) \cup \\ \forall\{task \in role.tasks \mid task.agentTask \wedge \neg isComposed(task)\}: \\ \quad task2vm(task) \cup \\ \forall\{task \in role.tasks \mid task.agentTask \wedge isComposed(task)\}: \\ \quad composeVm(task) \end{array}\right\} \\ \text{views} & \left\{\begin{array}{l} \forall\{vm \in gui.viewModels \mid \neg isComposed(vm)\}: \\ \quad vm2view(vm) \cup \\ \forall\{vm \in gui.viewModels \mid isComposed(vm)\}: \\ \quad composeView(vm) \end{array}\right\} \\ \text{roleRef} & role \end{array}\right]$$

In einer rollenspezifischen GUI werden für alle Aufgaben der Rolle durch die Transformation *task2vm* (7.33) Präsentationsmodelle (ViewModel) erzeugt. Die Transformation *taskmodel2vm* erzeugt ein Präsentationsmodell, welches eine Prozessinstanz mit allen benötigten Eingabeparametern starten kann. Sie ist analog zur Transformation *task2vm* aufgebaut und wird deshalb nicht gesondert aufgeführt. Anschließend werden durch die Transformation *vm2view* (7.37) für alle so generierten Präsentationsmodelle entsprechende Präsentationen (View) erzeugt. Daraus wird ersichtlich, dass es sich hierbei um einen zweistufigen Transformationsprozess handelt, der sowohl eine Komposition im Sinne einer Wiederverwendung von Präsentationsmodellen als auch von Präsentationen getrennt ermöglicht.

$$\begin{aligned} \textbf{task2vm} : Task &\longrightarrow ViewModel \\ task &\longmapsto vm \end{aligned} \tag{7.33}$$

mit

$$vm = \left[\begin{array}{ll} \textsc{ViewModel} & \\ \text{taskRef} & task \\ \text{elements} & \left\{\begin{array}{l} \boxed{A}\ \forall\{cf \in \{\mathsf{UnclaimTask},\ \mathsf{TaskPerformed},\ ...\}\}: \\ \quad task2vmMethod(task,cf) \\ \boxed{B}\ \forall\{s \in \mathcal{S}_{\mathcal{IN}}|_{task}\}: slot2vmProperty(s,\ true) \\ \boxed{C}\ \forall\{s \in \mathcal{S}_{\mathcal{OUT}}|_{task}\}: slot2vmProperty(s,\ false) \end{array}\right\} \end{array}\right]$$

Das Präsentationsmodell einer Aufgabe setzt sich aus Methoden (ViewModelMethod) für aufgabenbezogene kommunikative Funktionen, z. B. UnclaimTask oder CompleteTask, welche im Fall $\boxed{A}$ durch die Transformation *task2vmMethod* (7.34) erzeugt werden, sowie aus Eigenschaften (ViewModelProperties) für aufgabenspezifische Slots, welche durch die Transformation *slot2vmProperty* (7.35) in den Fällen $\boxed{B}$ und $\boxed{C}$ erzeugt werden, zusammen. Die Schreibweise $\mathcal{S}_{\mathcal{IN}}|_{\text{task}}$ drückt dabei aus, dass die Menge $\mathcal{S}_{\mathcal{IN}}$, welche in Kapitel 7.3.5 (S. 164) definiert ist, auf für die Aufgabe *task* relevante Elemente eingeschränkt ist.

$$\begin{aligned} \textbf{task2vmMethod} : Task \times EClass &\longrightarrow ViewModelMethod \\ (task, cf) &\longmapsto vmmethod \end{aligned} \tag{7.34}$$

mit

$$vmmethod = \begin{bmatrix} \textsc{ViewModelMethod} & \\ \text{name} & \textit{cf.toString()} \\ \text{onInvoke} & \textit{selectInputAct(task, cf)} \end{bmatrix}$$

Die Hilfsmethode *selectInputAct* sucht im Dialogaktmodell einen Eingabeakt mit der kommunikativen Funktion *cf* und aufgelöstem bzw. durch eine KnowledgeBaseReference beschriebenen semantischen Inhalt, der das Objekt *task* referenziert. Laut Konstruktion gibt es genau einen solchen Akt. Die Hilfsmethode *selectOutputAct*, welche in der folgenden Transformation verwendet wird, ist analog definiert.

$$\begin{aligned} \textbf{slot2vmProperty} : PSlot \times Boolean &\longrightarrow ViewModelProperty \\ (slot, editable) &\longmapsto vmproperty \end{aligned} \tag{7.35}$$

mit

$$vmproperty = \begin{bmatrix} \textsc{ViewModelProperty} & & \\ \text{name} & \textit{slot.featureName} & \\ \text{editable} & \textit{editable} & \\ \text{slotRef} & \textit{slot} & \\ \text{array} & \textit{slot.feature.isMany()} & \\ \text{onChange} & \begin{Bmatrix} \textit{selectInputAct(slot,Inform)} & \text{falls } \textit{editable} \\ \textit{selectOutputAct(slot,Inform)} & \text{sonst} \end{Bmatrix} & \end{bmatrix}$$

Die Transformation *composeVm* (7.36) sorgt im Falle einer zugrundeliegenden komponierten Aufgabe für die Komposition des zugehörigen Präsentationsmodells.

$$\begin{aligned} \textbf{composeVm} : Task &\longrightarrow ViewModel \\ task &\longmapsto vm \end{aligned} \tag{7.36}$$

mit

$$vm = \begin{bmatrix} \textsc{ViewModel} & \\ \text{taskRef} & task \\ \text{elements} & \left\{ el \mid el \in cm.inputApps.guiModel.guiApps.viewModels \wedge \right. \\ & \left. \quad el.eContainer().taskRef \in getInputs(task) \right\} \end{bmatrix}$$

Nach erfolgter Komposition eines Präsentationsmodells wird als Nebeneffekt wieder das Kompositionsmodell ergänzt.

$$\begin{bmatrix} \textsc{ViewModelComposition} & \\ \text{parent} & getComposition(task) \\ \text{type} & getComposition(task).type \\ \text{inputElements} & \text{Auswahlmenge der } el.eContainer() \\ \text{outputElement} & vm \end{bmatrix}$$

Die Transformation *vm2view* (7.37) strukturiert die Kindelemente der erzeugten Präsentation in Abhängigkeit vom konkreten Typ und der Ausprägung der Bestandteile des Präsentationsmodells. Im Fall $\boxed{A}$ werden mittels der Transformation *vmMethod2button* (7.38) Schaltflächen (Button) zum Ausführen von Methoden erzeugt. Im Fall $\boxed{B}$ werden editierbare Eigenschaften des Präsentationsmodells durch die Transformation *vmProperty2input* in Steuerelemente zur Präsentation und Eingabe von Information (Input) überführt. Analog werden im Fall $\boxed{C}$ für unveränderliche Eigenschaften durch die Transformation *vmProperty2label* Label lediglich zur Darstellung der Information erzeugt.

$$\begin{aligned} \textbf{vm2view} : ViewModel &\longrightarrow View \\ vm &\longmapsto view \end{aligned} \tag{7.37}$$

mit

$$
view = \left[\begin{array}{ll}
\textsc{View} & \\
\text{scope} & vm \\
\text{children} & \left\{\begin{array}{ll}
\boxed{A} & \left[\begin{array}{ll}
\textsc{GuiElement} & \\
\text{children} & \left\{\begin{array}{l}\forall\{el \in vm.elements \mid \\ \quad ViewModelMethod(el)\}: \\ \quad\quad vmMethod2button(el)\end{array}\right\}
\end{array}\right], \\
\boxed{B} & \left[\begin{array}{ll}
\textsc{GuiElement} & \\
\text{children} & \left\{\begin{array}{l}\forall\{el \in vm.elements \mid \\ \quad ViewModelProperty(el) \wedge el.editable\}: \\ \quad\quad vmProperty2input(el)\end{array}\right\}
\end{array}\right], \\
\boxed{C} & \left[\begin{array}{ll}
\textsc{GuiElement} & \\
\text{children} & \left\{\begin{array}{l}\forall\{el \in vm.elements \mid \\ \quad ViewModelProperty(el) \wedge \neg el.editable\}: \\ \quad\quad vmProperty2label(el)\end{array}\right\}
\end{array}\right]
\end{array}\right\}
\end{array}\right]
$$

Die Transformation *vmMethod2button* (7.38) ist schließlich auch beispielhaft für die Transformationen *vmProperty2label* und *vmProperty2input* ausgeführt, die analog aufgebaut sind.

$$
\begin{array}{r}
\textbf{vmMethod2button} : ViewModelMethod \longrightarrow Button \\
el \longmapsto button
\end{array}
\tag{7.38}
$$

mit

$$
button = \left[\begin{array}{ll}
\textsc{Button} & \\
\text{binding} & \left[\begin{array}{ll}
\textsc{DataBinding} & \\
\text{type} & click \\
\text{valueElement} & el
\end{array}\right]
\end{array}\right]
$$

Die Komposition einer Präsentation anhand eines komponierten Präsentationsmodells erfolgt durch die Transformation *composeView* (7.39). Hierbei braucht nun zwar keine Unterscheidung mehr anhand des Kompositionstyps getroffen zu werden. Allerdings gilt es, die hierarchische Struktur der Präsentation beizubehalten. Dies wird durch die Hilfsmethode *clonePath()* realisiert, welche ausgehend von einem grafischen Darstellungsobjekt *element* alle Elternobjekte incl. *element* unter Zuhilfenahme eindeutiger Elementbezeichner in ihrer Hierarchie dupliziert. Diese einzelnen Pfade werden anschließend wieder unifikationsbasiert zusammengeführt. Die Hilfsmethode *getView* sucht zu einem Bestandteil des Präsentationsmodells das entsprechende Darstellungsobjekt.

$$\begin{aligned} \textbf{composeView} : ViewModel &\longrightarrow View \\ vm &\longmapsto view \end{aligned} \tag{7.39}$$

mit

$$view = \begin{bmatrix} \textsc{View} & \\ \text{scope} & vm \\ \text{children} & \left\{ \begin{array}{l} unify(\\ \quad \forall el \in vm.elements: \\ \quad clonePath(getView(getInputs(el).el())) \\ ).children \end{array} \right\} \end{bmatrix}$$

Nach erfolgter Aktualisierung der Datenbindung auf das eigentliche Element des Präsentationsmodells *el* erfolgt schließlich für alle Elemente einer komponierten Präsentation *vm* eine letztmalige Ergänzung des Kompositionsmodells.

$$\begin{bmatrix} \textsc{ViewComposition} & \\ \text{parent} & getComposition(el) \\ \text{type} & getComposition(el).type \\ \text{inputElements} & getInputs(el) \\ \text{outputElement} & Blatt\ des\ geklonten\ Pfades \end{bmatrix}$$

(14) Manuelle Ergänzung des GUI-Modells

Nach der Generierung des initialen GUI-Modells stellt dieses im Ordnungsrahmen von Garrett (Kapitel 5.2.1, S. 72) das Skelett der grafischen Benutzerschnittstelle dar. Deren genaue Präsentation im Rahmen der Oberflächenebene gilt es noch zu spezifizieren. Hierzu können auch die generierten Elemente einer Ansicht neu arrangiert werden und dabei ihre vorgegebene Struktur im Modell verlassen. Durch die Komposition ist sichergestellt, dass diese manuellen Anpassungen und Verfeinerung übertragen werden.

Beispiel: GUI-Modell

Wie in Abbildung 7.11 ersichtlich, wird für jede Rolle des Kollaborationsprozesses ein Modell einer eigenständigen grafischen Benutzerschnittstelle erzeugt.

Für die Aufgabe *Zeitraum und Vertretung angeben* der Rolle *Angestellter* wird in der ersten Phase der Generierung ein Präsentationsmodell erzeugt. Auf Basis der im Kontext der Aufgabe erzeugten Eingabeakte wird das Präsentationsmodell während der Transformation mit entsprechenden Eigenschafen befüllt und mit den Eingabeakten verknüpft. In der zweiten Phase wird für das Präsentationsmodell eine korrespondierende grafische Präsentation erzeugt. Jede Eigenschaft des Präsentationsmodells wird auf grafische Elemente innerhalb der Präsentation abgebildet. Diese Inhalte werden wiederum durch bidirektionale Datenbindungen mit den korrespondierenden Eigenschaften des Präsentationsmodells verknüpft.

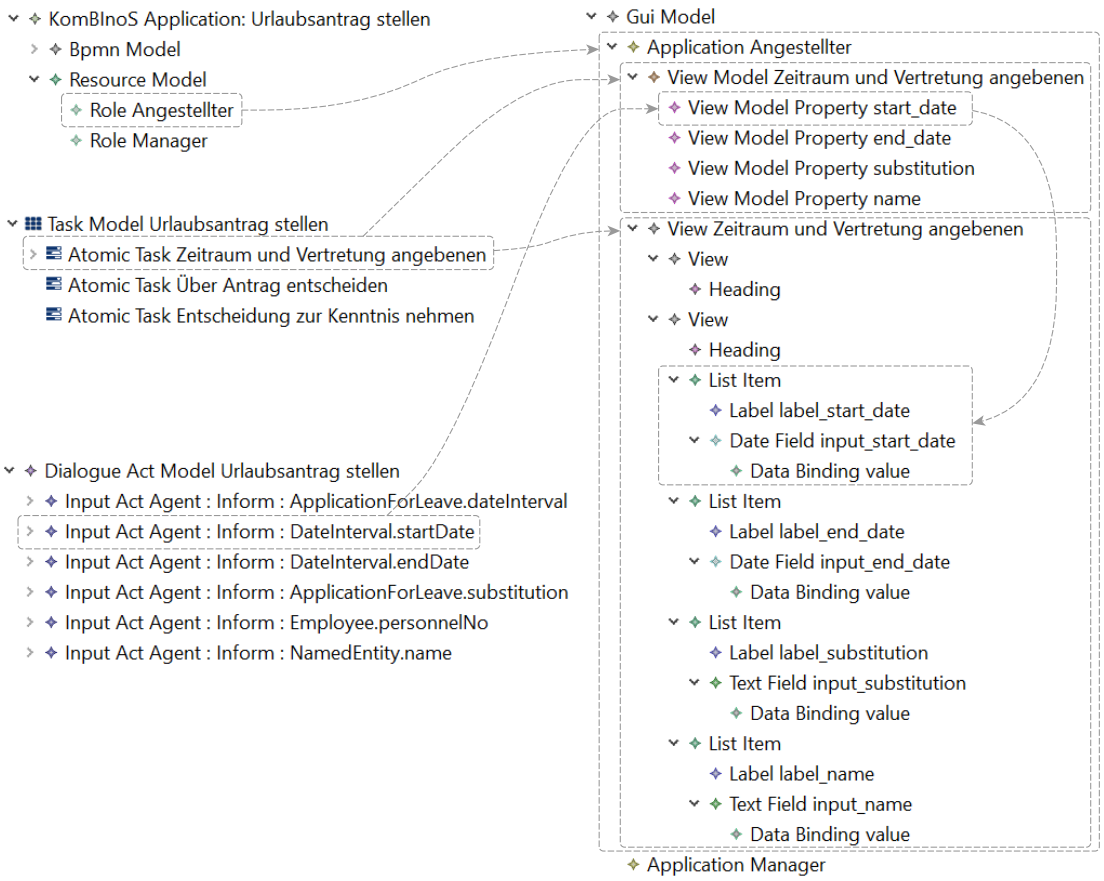


Abbildung 7.11: Beispiel - GUI-Modell

(15) Finale Codegenerierung

Für die finale Codegenerierung nach HTML5 und JavaScript müssen die entsprechenden M2C-Transformationen so gestaltet sein, dass einerseits möglichst wenige Ergänzungen am finalen Code der clientseitigen GUI notwendig sind. Andererseits sollen notwendige Ergänzungen etwa durch TODO-Kommentare gekennzeichnet sein und so gekapselt werden können, dass sie von einer wiederholten Codegenerierung nicht tangiert werden. Dies kann durch die Trennung von generiertem und händisch erstelltem Code auf unterschiedliche physische Dateien, z. B. in Form von partiellen Klassen oder Mixins, oder durch die Verwendung von Code-Annotationen erreicht werden. In jedem Fall sollte die finale GUI strukturell dem GUI-Modell entsprechen, da letzteres zur Laufzeit als Anzeigekontext fungiert. Zur Laufzeit können hierzu auch Synchronisationsmechanismen eingesetzt werden. In Kapitel 8.5 (S. 194) wird exemplarisch die Codegenerierungsschablone zur Erzeugung von rollenabhängigen GUIs aufgelistet.

7.3.9 Erstellung weiterer paralinguistischer Ein- und Ausgabemodelle

Mit der Erzeugung der konkreten Modelle für die natürlichsprachliche Interaktion (Eingabe und Ausgabe) sowie rollenspezifische grafische Benutzerschnittstellen wurden komplexe Transformationen umgesetzt und dokumentiert. In Kapitel 6.7.4 (S. 136) wurde ein weiteres Eingabemodell zur Gesteninteraktion sowie ein Ausgabemodell zur Steuerung eines virtuellen Charakters beschrieben. Die Definition von Transformationen zur modellgetriebenen Erstellung dieser und weiterer paralinguistischer Ein- und Ausgabemodelle erfolgt nach den gleichen Prinzipien und Techniken wie bereits erläutert. Auf eine weiterer Detaillierung wird daher verzichtet.

7.4 Fazit

In diesem Kapitel wurde die integrierte KomBInoS-Methode zur modellgetriebenen Entwicklung und Komposition von multimodalen Dialogschnittstellen für Smart Services entwickelt. Die Methode alterniert zwischen automatischen Modelltransformationen und manuellen Ergänzungen entlang der KomBInoS-Metamodell-Architektur. Die Transformationen wurden formal spezifiziert und mittels der MDA-konformen Transformationssprache QVT (Kapitel 2.5, S. 26) umgesetzt. Es wurde zudem erläutert, um welche Information jedes Modell angereichert werden muss, damit eine weitere Transformation sinnvoll angestoßen werden kann. Auf dieser Basis wurden manuelle Tätigkeiten abgeleitet. Hierdurch wird die Forschungsfrage (1) „*Wie kann man multimodale Dialogschnittstellen für Smart Services schnell und mit geringem Programmieraufwand entwickeln? Wie sehen eine notwendige Metamodell-Architektur und entsprechende Modelltransformationen aus, die funktional vollständige und über Modalitätsgrenzen hinweg konsistente Benutzerschnittstellen erschaffen?*“ unter dem Gesichtspunkt der Gestaltung von notwendigen Modelltransformationen im Rahmen der UI-Entwicklung umfassend beantwortet. Außerdem wird in den Transformationen auf Grundlage des Kompositionsmodells konsequent zwischen der Erzeugung neuer und der Wiederverwendung und Adaption existierender UI-Artefakte unterschieden. Insgesamt wird somit die Forschungsfrage (3) „*Wie sieht ein strukturiertes Vorgehen aus, welches sowohl Entwicklungs- als auch Kompositionstätigkeiten bei der Erstellung einer multimodalen Dialogschnittstelle für einen Smart Service integriert?*“ vollständig beantwortet. Die KomBInoS-Methode wurde auf der Basis einer eingehenden Anforderungsanalyse bzgl. der prinzipiellen Anwendbarkeit, der Herausforderungen und Grenzen eines modellgetriebenen Entwicklungs- und Kompositionsverfahrens für Benutzerschnittstellen für Dienste herausgearbeitet. In diesem Zusammenhang wurde festgehalten, dass generative Ansätze über mehrere Abstraktionsebenen nahezu ausschließlich im Forward-Engineering funktionieren können. Zur Abhängigkeitsverwaltung im Rahmen einer modellgetriebenen Komposition wurde ein entsprechendes Verfahren skizziert, das bis an die Grenzen einer sinnhaften Komposition heranreicht. Kurzum, das Verfahren stößt im Allgemeinen bei einer sehr hohen Kompositionskomplexität, etwa wenn mehrere Pro-

zessaktivitäten adaptiert und zu einer einzigen verschmolzen wurden, an seine Grenzen. Wenn jedoch im Speziellen die miteinander komponierten Aktivitäten auf Basis eines gemeinsamen abstrakten Domänenmodells aufeinander abgestimmt sind (etwa im Rahmen einer Vorlagenbibliothek von typischen Kollaborationsmustern, wie in Kapitel 6.8.1 (S. 137) unter Punkt (3) erläutert), dann kann die Methode in diesem praxisrelevanten Fall bei der Komposition erfolgreich unterstützen. Insofern ist nun auch die Forschungsfrage (2) *"Wie kann man existierende multimodale Dialogschnittstellen von Smart Services wiederverwenden? Bis zu welchem Grad ist dies sinnvoll möglich? Wie können darüber hinaus auch Modellartefakte multimodaler Dialogschnittstellen wiederverwendet werden, deren zugehörige Dienste nicht Teil der Kompositionskette sind?*" unter dem Aspekt der Grenzen einer sinnhaften Komposition von UIs vollständig beantwortet.

KAPITEL 8

Die KomBInoS-Werkbank

Dieses Kapitel beschreibt die nahtlose Werkzeugkette der KomBInoS-Plattform als Implementierung der im vorherigen Kapitel vorgestellten integrierten Entwicklungs- und Kompositionsmethode. Sie unterstützt Entwickler zur Entwurfszeit bei der Anwendung der Methode und auch bei der verteilten Erstellung einer UI. Sie beantwortet folglich die eingangs formulierte ingenieurtechnische Fragestellung (4) bzgl. einer nahtlosen, adäquaten Werkzeugunterstützung.

8.1 Einleitung

Die Werkzeugkette zur Erstellung von KomBInoS-UIs bildet das im vorherigen Kapitel beschriebene methodische Vorgehen in Software ab. Die einzelnen Werkzeuge sind in der KomBInoS-Werkbank gebündelt, welche auf einer Standardinstallation der Eclipse-IDE aufsetzt. Die Werkzeuge werden im Folgenden entsprechend der Reihenfolge ihrer erstmaligen Verwendung beschrieben. Dabei wird technisch auch auf die benutzten Eclipse-Erweiterungspunkte eingegangen, die für die nahtlose Integration der Werkzeuge in die IDE sorgen. Schließlich wird im Fazit ein Überblick über die Softwarearchitektur der Werkbank gegeben.

8.2 Wizard zur Erstellung einer neuen KomBInoS-UI

Eine KomBInoS-UI wird durch ein Bündel von Modellressourcen, Konfigurationseinstellungen und Diensten für eine OSGi-basierte Diensteplattform repräsentiert. Nach Fertigstellung und einer anschließenden Bereitstellung kann ein solches Bündel innerhalb der KomBInoS-Laufzeitumgebung, welche im folgenden Kapitel 9 (S. 211) detailliert erläutert wird, ausgeführt werden.

Mit Hilfe eines dedizierten Wizards erstellt ein Entwickler das Grundgerüst einer neuen KomBInoS-UI. Der Wizard führt dabei Schritt für Schritt sicher durch die initiale Konfiguration der zu entwickelnden multimodalen Dialogschnittstelle für einen Smart Service. Abbildung 8.1 zeigt die entsprechenden Schritte.

In den Schritten (1) und (2) werden allgemeine Eigenschaften für OSGi-Projekte festgelegt. Bis auf die obligatorische Angabe des Projektnamens in (1) und gegebenenfalls die Anpassung der daraus automatisch abgeleiteten Projekt-Id in (2), welche den standardmäßigen Namensraum für Java-Klassen darstellt, besteht nur in Ausnahmefällen weiterer Handlungsbedarf. Die Entwicklung einer neuen KomBInoS-UI startet ausgehend von einer formalen Beschreibung eines Kollaborationsprozesses und einer optionalen, begleitenden Beschreibung der Komposition auf Prozessebene. In Schritt (3) wählt der Entwickler dementsprechend einen vorhandenen Kollaborationsprozess entweder in BPMN- oder CMMN-Notation (Kapitel 3.4.3.2, S. 41) aus. Auch hier ist die Nutzung eines Kollaborationsprozesses in einer zukünftig standardisierten (erweiterten) EPK-Notation bereits vorbereitet. Nach erfolgter Selektion werden die restlichen Optionen ausgegraut. Falls vorhanden, kann er zusätzlich sowohl ein Domänenmodell in Ecore-Notation (Kapitel 2.5, S. 24) als auch ein Kompositionsmodell auswählen. Anschließend wird auf Basis einer eigens vorbereiteten Schablone das Grundgerüst eines KomBInoS-Projekts erzeugt, welches in (4) dargestellt ist. Dieses Grundgerüst umfasst Java-Code und eine Komponentendefinition (component.xml) für die OSGi-basierte Integration in die KomBInoS-Laufzeitumgebung, eine Servlet-Definition (web.xml) zur Bereitstellung von generierten Artefakten via HTTP, z. B. HTML-Code der grafischen Benutzerschnittstelle im noch leeren www-Verzeichnis, ein Skript zum Bündeln und Bereitstellen einer KomBInoS-UI auf Basis von Apache Ant[1] (build.xml), einem Java-basierten Werkzeugs zur Automatisierung von wiederkehrenden Aufgaben in der Softwareentwicklung mit nahtloser Eclipse-Integration, sowie initial leere Modelle für die einzelnen Facetten einer KomBInoS-UI. Die in (3) angegebenen Modellressourcen werden jedoch automatisch in das erstellte Projekt importiert und im Anwendungsmodell entsprechend verknüpft.

Die Eingabemasken der Schritte (1) und (2) stammen aus dem Eclipse PDE UI Wizard[2]. Durch Ausnutzung des Erweiterungspunkts org.eclipse.pde.ui.pluginContent kann die eigens implementierte Maske aus Schritt (3) nahtlos angegliedert werden. Mit Hilfe des Erweiterungspunkts org.eclipse.ui.newWizards wird der so komponierte Wizard in Menüstrukturen für die Erstellung neuer Projekte (z. B. [Datei] →[Neu] →[Projekt]) integriert und für den Entwickler leicht auffind- und zugreifbar.

[1] siehe https://ant.apache.org/ (letzter Zugriff: 04.11.2017)
[2] siehe https://eclipse.org/pde/pde-ui/ (letzter Zugriff: 04.11.2017)

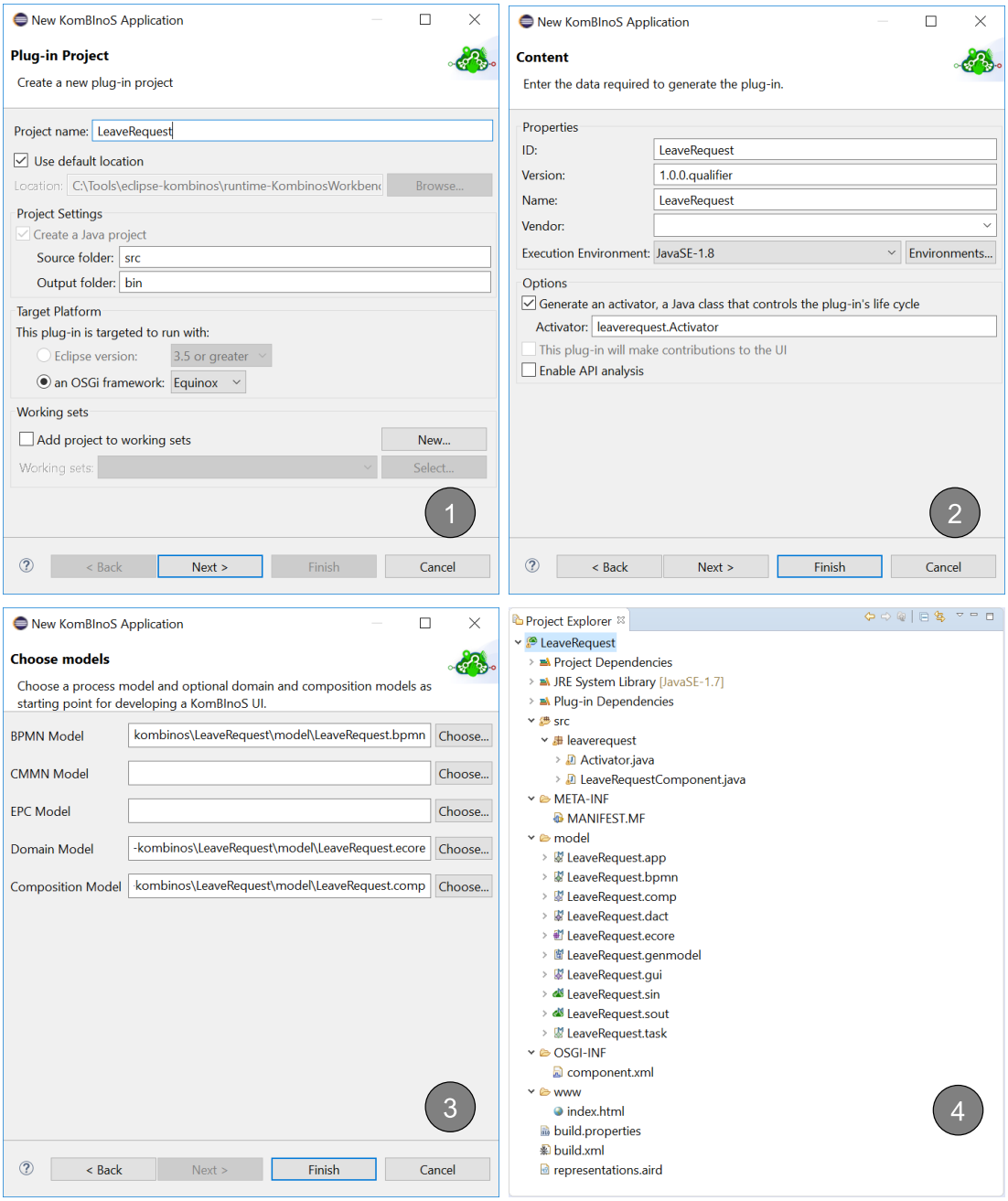


Abbildung 8.1: Wizard für neue KomBInoS-Anwendungen

8.3 Angepasste Fensterperspektive

Beim Anlegen eines neuen KomBInoS-Projekts kann der Entwickler die IDE in die mit dem Projekttyp assoziierte KomBInoS-Perspektive umschalten. Diese Perspektive selektiert und arrangiert häufig benutzte Fenster und spezialisierte Werkzeuge, sodass die IDE optimal zur Erstellung von KomBInoS-UIs angepasst wird. Während Abbildung 8.2 die entsprechende Dialogbox zum Umschalten zeigt, gibt Abbildung 8.3 eine Übersicht über die einzelnen Bereiche und Funktionen der Perspektive.

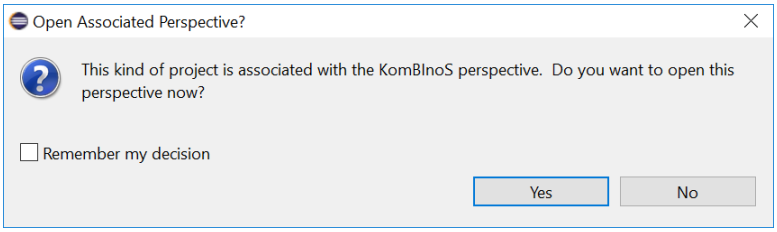


Abbildung 8.2: Dialogbox zum Wechseln in die KomBInoS-Entwicklungsperspektive

(1) **Die aktive KomBInoS-Perspektive.** Hier kann auch zwischen anderen Perspektiven, z. B. zur Java-Entwicklung von OSGi-Bündel, gewechselt werden. Die jeweils letzte Fensteranordnung wird wiederhergestellt, solange eine Perspektive nicht zurückgesetzt wird. Eine Perspektive wird in Eclipse über den Erweiterungspunkt org.eclipse.ui.perspectives definiert.

(2) **Schnellzugriff auf KomBInoS-spezifische Wizards** zum Erstellen neuer Projekte und Modelle über die [Neu]-Schaltfläche der Eclipse-Werkzeugleiste.

(3) **Projektspezifisches Kontextmenü zum Schnellzugriff auf wichtige Funktionen.** Die Integration in die IDE erfolgt über den Erweiterungspunkt org.eclipse.ui.menus. Enthaltene Kommandos, z. B. *Deploy*, werden abstrakt und so an anderen Stellen wiederverwendbar mit Hilfe des Erweiterungspunkts org.eclipse.ui.commands modelliert. Die Abbildung eines solchen Kommandos auf eine konkrete Java-Funktion erfolgt durch den Erweiterungspunkt org.eclipse.ui.handlers.

(4) **KomBInoS-spezifische Aufgaben- und Problemansichten für Entwickler.** Der Entwickler kann sich einen schnellen Überblick über noch ausstehende Tätigkeiten im Rahmen der UI-Erstellung sowie Hinweise auf potentielle Probleme und noch zu behebende Fehler verschaffen. Durch einen Inhaltsfilter können beide Ansichten so konfiguriert werden, dass jeweils nur KomBInoS-spezifische Einträge angezeigt und nicht relevante ausgeblendet werden. Entsprechende Aufgaben- und Problemmarker wurden mit Hilfe des Erweiterungspunkts org.eclipse.core.resources.markers definiert.

(5) **Dashboard zur Entwicklung und Komposition von KomBInoS-UIs.** Dieses bietet dem Entwickler eine methodische Unterstützung und wird im nächsten Abschnitt gesondert im Detail vorgestellt.

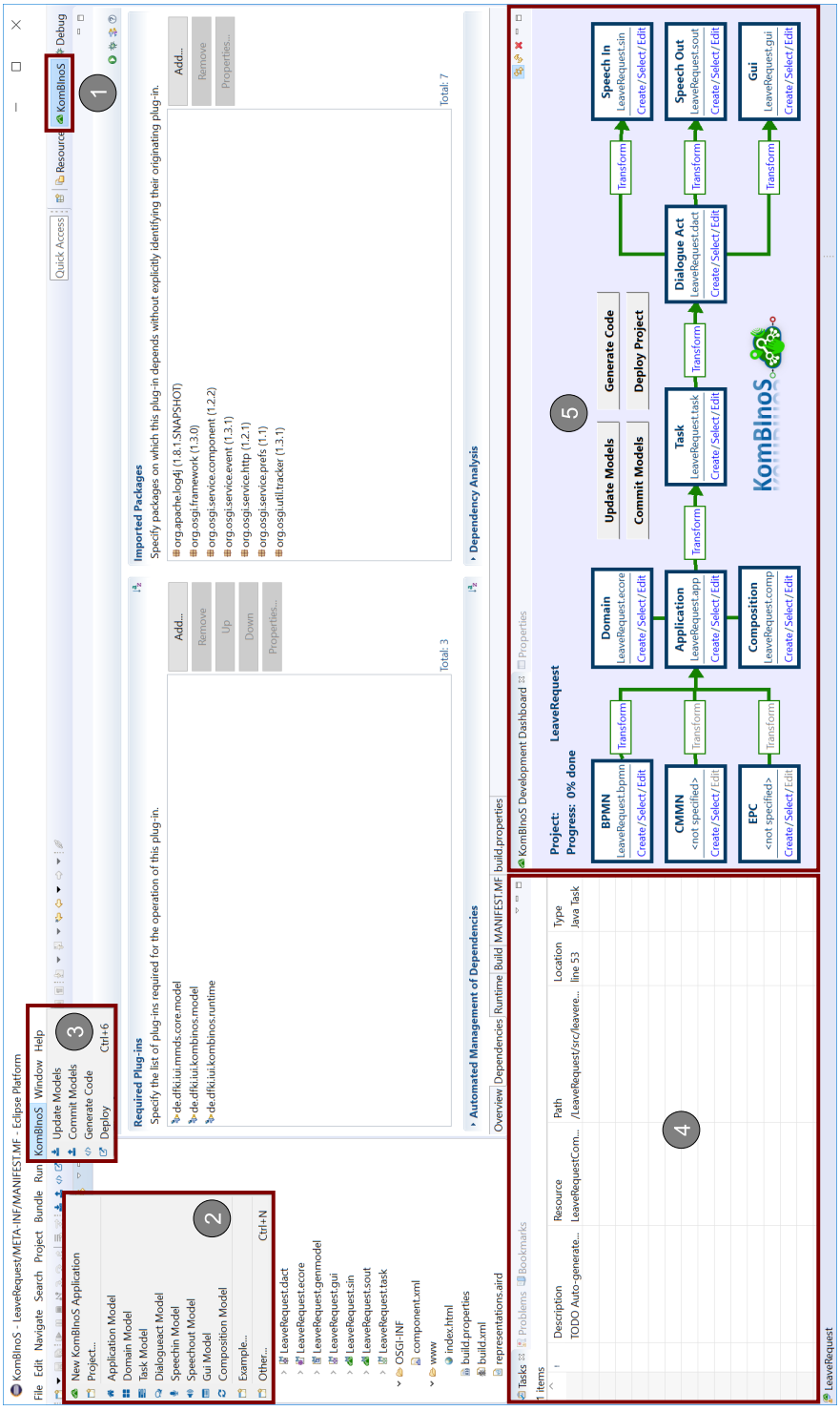


Abbildung 8.3: Die KomBInoS-Entwicklungsperspektive im Überblick

8.4 Dashboard zur methodischen Unterstützung

Das Dashboard, dargestellt in Abbildung 8.3, (5), stellt den Kern der methodischen Unterstützung bei der modellgetriebenen Entwicklung und Komposition von multimodalen Dialogschnittstellen für Smart Services dar. Es setzt den in Abbildung 7.2 (S. 148) gezeigten, konzeptuellen Entwicklungs- und Kompositionsprozess 1-zu-1 in einer interaktiven Prozessvisualisierung um. Der Entwickler sieht auf den ersten Blick, wie weit die UI-Erstellung des im Eclipse-Projekt-Explorer selektierten KomBInoS-Projekts in Abbildung 8.1, (4), vorangeschritten ist und welche Modelle in den Erstellungsprozess involviert sind. Hierbei werden bereits die im Projekt-Wizard importierten oder generierten Modelle voreingestellt, sodass das Dashboard sofort genutzt werden kann. Dazu können die Modelle zum Bearbeiten direkt über das Dashboard geöffnet werden. Alternativ können, etwa zu einem späteren Zeitpunkt, auch neue Modelle erzeugt oder anhand der registrierten Dateiendung typsicher andere Modelle ausgewählt werden. Durch einen Klick auf eine *Transform*-Schaltfläche zwischen einzelnen Modellen wird eine automatische M2M-Transformation, wie Kapitel 7.3 (S. 147) beschrieben, durchgeführt. Die finale *Deploy*-Schaltfläche ist erst verfügbar, wenn die Fortschrittsanzeige 100 Prozent erreicht hat, also alle notwendigen Modelle vollständig erzeugt wurden und die Konsistenzprüfung auf Basis der in den Metamodellen hinterlegten Invarianten (Abbildung 6.10, S. 124) erfolgreich war. Erkannte Defizite und Probleme werden durch Einträge in der Aufgaben- und Problemansicht gesammelt und auch an der entsprechenden Stelle im Modell gekennzeichnet.

Die Idee des Entwicklungsdashboards stammt vom Dashboard des Eclipse Graphical Modeling Frameworks (Gronback 2009, S. 67). Die Implementierung dieses Dashboards wurde im Rahmen dieser Arbeit auf Basis des offenen Quellcodes stark erweitert und flexibel für eigene Zwecke eröffnet. Das Ergebnis ist ein konfigurier- und wiederverwendbares Rahmenwerk für jedwede Form von Dashboards zur modellgetriebenen Softwareentwicklung. Dazu wurden die wesentlichen Teile des Quellcodes konzeptionell analysiert und in ein eigenes EMF-basiertes Metamodell abstrahiert. Abbildung 8.4 illustriert dieses ausschnittsweise. Das Dashboard-Rahmenwerk ist in eigenständige Eclipse-Plugins aufgeteilt, die einen eigenen Erweiterungspunkt de.dfki.iui.kombinos.dashboard.views zur Deklaration von beliebigen Dashboards bereitstellen.

Abbildung 8.5 zeigt die Definition des KomBInoS-Dashboards, welche für die Ansicht in Abbildung 8.3, (5), verantwortlich ist. Ein Dashboard (Board) beinhaltet mehrere grafische Modellelemente (ModelElement) und Transformationselemente (TransformationElement). Jedes Transformationselement verknüpft mehrere Quell- und Zielmodelle. Zur Realisierung der Transformation über das Dashboard wird für jedes grafische Transformationselement eine konkrete QVT-Transformation anhand eines eindeutigen Bezeichners bestimmt, die die Quellmodelle automatisiert in die Zielmodelle überführen. Die Einbindung der QVT-Transformationen in die KomBInoS-Werkbank ist Gegenstand des nächsten Abschnitts. Außerdem enthält das Dashboard zusätzlich Schaltflächen, z. B. zum Aktualisieren der Modelle oder zum Generieren von Code.

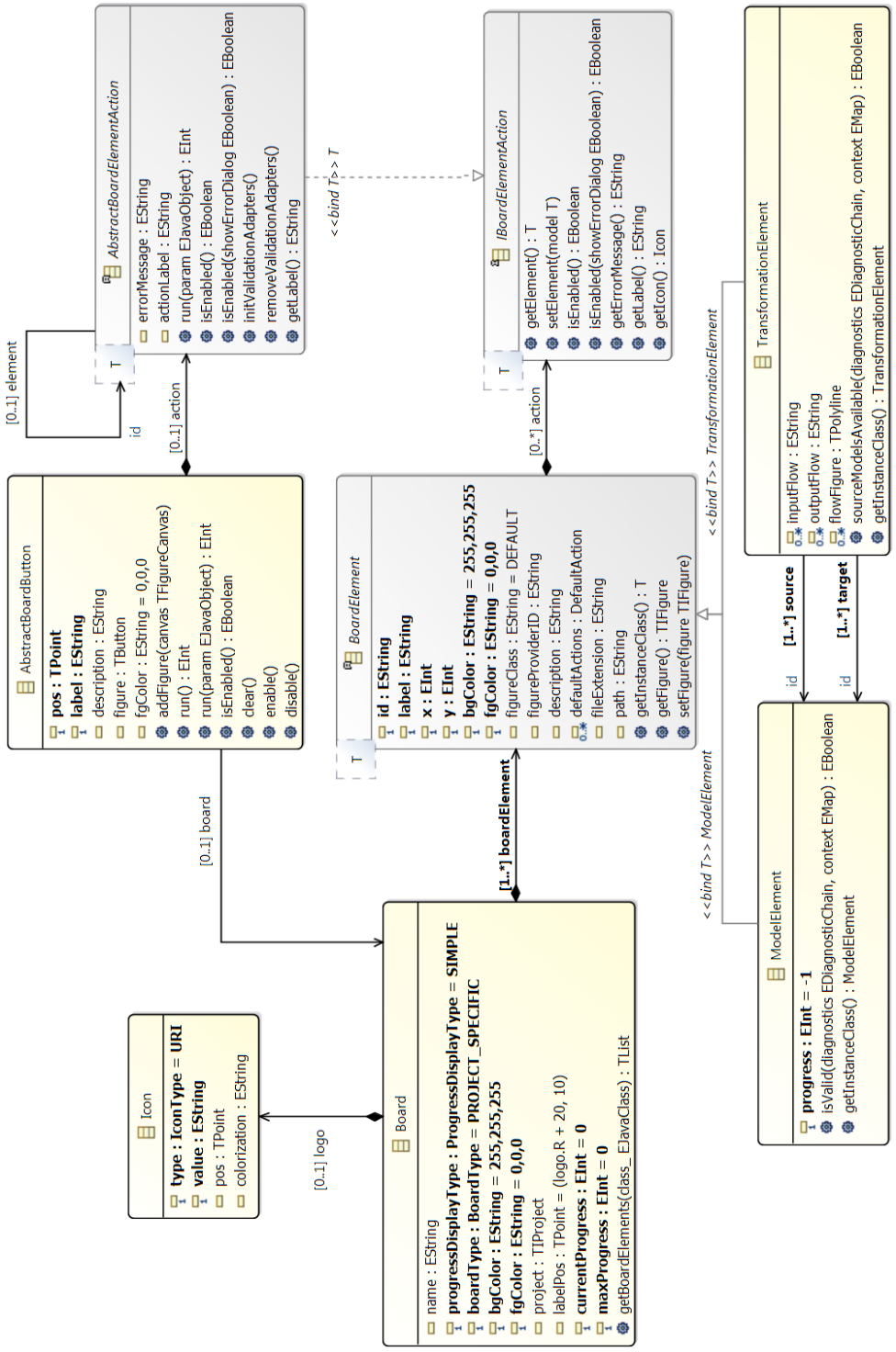


Abbildung 8.4: Das Dashboard-Metamodell

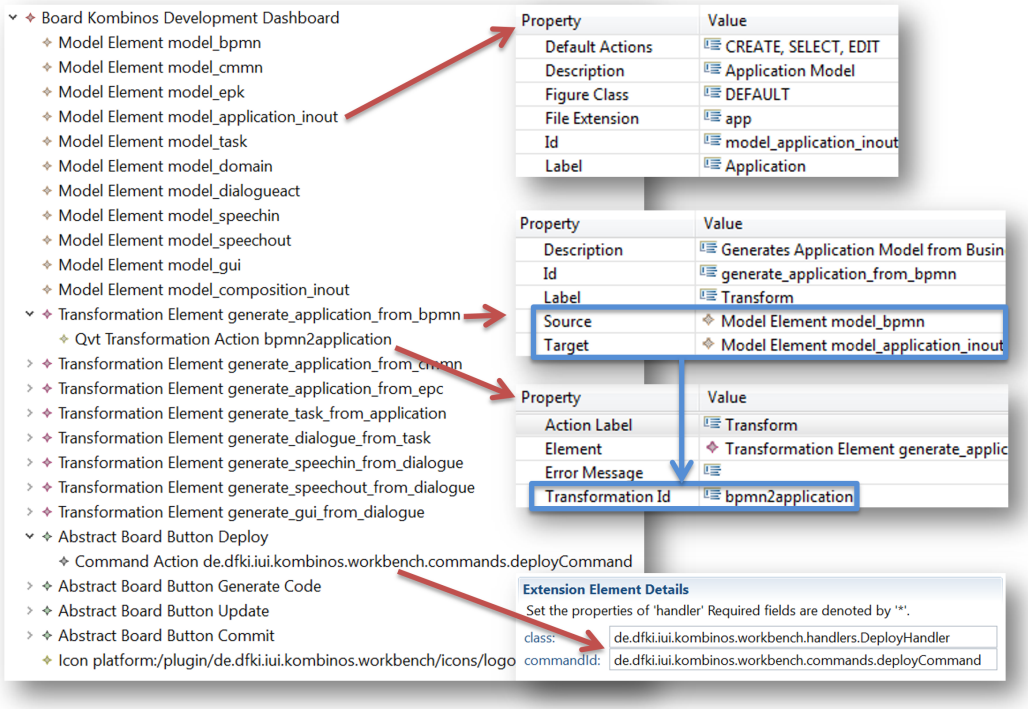


Abbildung 8.5: Das Modell des KomBInoS-Entwicklungsdashboards

8.5 Modelltransformationen

Die Implementierung der in Kapitel 7.3 (S. 147) definierten M2M-Transformationen erfolgte mit QVT in der operationalen Variante (Kapitel 2.5, S. 26), da eine bidirektionale Anwendung von Transformationen im Rahmen eines Roundtrip-Engineerings in KomBInoS u. a. aufgrund der enthaltenen Kompositionslogik nicht zielführend ist. Die entsprechenden Skripte sind Bestandteil der Werkbank und implementieren die in Kapitel 7.3 (S. 147) formal definierten Abbildungen. Zur Durchführung einer solchen Transformation wurde die Laufzeitumgebung des Eclipse Modeling Projekts[3] als Referenzimplementierung für die operationale QVT in die Werkbank integriert. Diese stellt mit dem Erweiterungspunkt org.eclipse.m2m.qvt.oml.runtime.qvtTransformation einen Mechanismus zur Verfügung, um QVT-Transformationsdateien unter einem eindeutigen Bezeichner innerhalb der IDE bekannt zu machen. Abbildung 8.6 zeigt die Verwendung des Erweiterungspunktes im Rahmen der Werkbank. Diese Bezeichner werden, wie im vorangegangenen Abschnitt beschrieben, im Dashboard wiederverwendet, um Transformationen per Knopfdruck ausführen zu können. Zusätzlich werden über den Erweiterungspunkt org.eclipse.m2m.qvt.oml.javaBlackboxUnit sogenannte

[3]siehe http://projects.eclipse.org/projects/modeling.mmt.qvt-oml (letzter Zugriff: 04.11.2017)

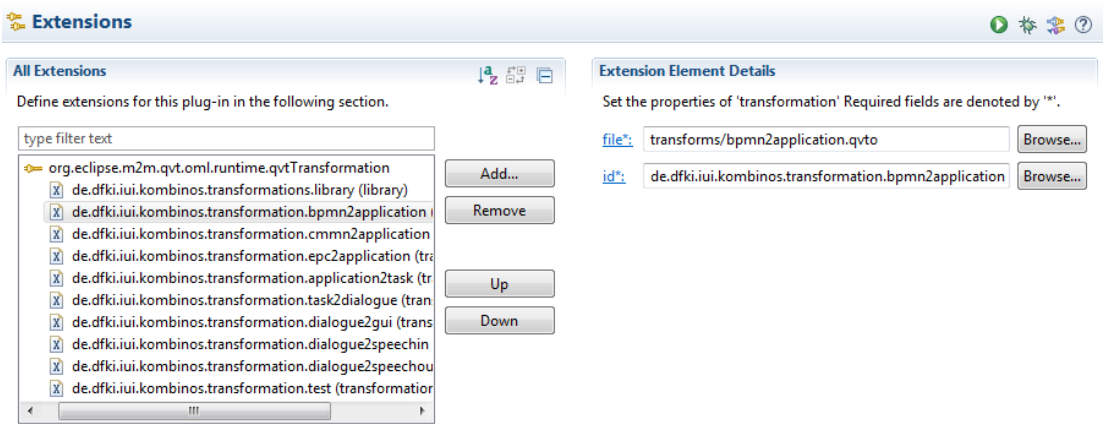


Abbildung 8.6: Deklaration der operationalen QVT-Transformationen durch den Erweiterungspunkt org.eclipse.m2m.qvt.oml.runtime.qvtTransformation

Java-Blackboxen registriert. Diese enthalten Bibliotheken von in Java entwickelten, seiteneffektfreien Helfermethoden, die in den Transformationen verwendet werden können.

Zur Durchführung von M2C-Transformationen wird das Acceleo-Rahmenwerk[4] als Java-basierte Implementierung des entsprechenden MDA-Standards mit nahtloser Eclipse-Integration eingesetzt. Entsprechende Codegenerierungsschablonen zur Erzeugung von HTML- und JavaScript-Code für rollenspezifische grafische UIs auf Basis des GUI-Modells können aus dem Dashboard heraus über das im Projekt enthaltene Ant-Skript zur Ausführung gebracht werden. Listing 8.1 zeigt den Einstiegspunkt in die rollenabhängige GUI-Generierung sowie in die Umsetzung des MVVM-Musters (Kapitel 6.7.3, S. 133) über mehrere physische Dateien hinweg.

Listing 8.1: Rollenabhängige GUI-Generierung gemäß MVVM-Muster mit Acceleo

```
1  [template public generateGui(guiModel : GuiModel)]
2  [file ('index.html', false, 'UTF-8')]
3  <span>Dies ist eine Übersicht über die rollenspezifischen GUIs.</span>
4  <table>
5    <thead><tr><th>Title</th><th>Role</th></tr></thead>
6    <tbody>
7  [for (guiApplication : GuiApplication | guiModel.applications)]
8      <tr>
9        <td><a href='[guiApplication.id.concat('/index.html') /]'>
10         [guiApplication.title/]</a></td>
11       <td>[guiApplication.roleReference.name/]</td>
12     </tr>
13 [/for]
14   </tbody>
15 </table>
16 [/file]
```

[4] siehe http://projects.eclipse.org/projects/modeling.m2t.acceleo (letzter Zugriff: 04.11.2017)

```
17 [for (guiApplication : GuiApplication | guiModel.applications)]
18   [guiApplication.generateGuiApplication()/]
19 [/for]
20 [/template]
21
22 [template public generateGuiApplication(guiApplication : GuiApplication)]
23 [for (aViewModel : ViewModel | guiApplication.viewModels)]
24   <!-- Generates view model classes in JavaScript -->
25   [aViewModel.generateViewModel(guiApplication.id)/]
26 [/for]
27 [file (guiApplication.id.concat('/index.html'), false, 'UTF-8')]
28   <!-- JavaScript-Includes and HTML Code here -->
29 [/file]
30 [/template]
```

8.6 Spezialisierte Modell-Editoren

Nach der Durchführung einer Transformation muss das Zielmodell manuell nachbearbeitet und um neue Information ergänzt werden. Dazu werden geeignete Editoren benötigt. Durch die EMF-basierte Codegenerierung der Metamodelle aus Kapitel 6 (S. 111) erhält man bereits ohne weiteres Zutun automatisch generierte Baumeditoren, die ein Modell wie in Abbildung 8.5 darstellen. Für viele Modellarten ist diese Form der Darstellung ausreichend. Für sehr textlastige oder aber prozessbeschreibende Modelle sind diese Editoren jedoch ungeeignet, da sowohl die Art der Visualisierung als auch der Erstellung nicht adäquat sind. So wurde im Rahmen der Arbeit auf Basis des Sirius-Rahmenwerks[5] zur deklarativen Erstellung von Modelleditoren ein grafischer Editor für Aufgabenmodelle entwickelt. Die deklarative Beschreibung eines komplexen Editors in Kombination mit dem Sirius-Modellinterpreter ohne die Notwendigkeit, Code für einen solchen Editor zu generieren oder neu zu entwickeln, erhöht im Vergleich zum bereits erwähnten Eclipse Graphical Modeling Framework die Robustheit und Anpassungsgeschwindigkeit eines solchen Editors. Abbildung 8.7 zeigt auf der linken Seite unter Punkt (1) beispielhaft ein Aufgabenmodell, welches mit dem automatisch generierten Baumeditor visualisiert wird. Das gleiche Modell wird in der Mitte (2) mit Hilfe des grafischen Editors angezeigt. Auf der rechten Seite (3) ist die zugrundeliegende Definition des Editors als Instanz des Sirius-Metamodells zu sehen. Darin können u. a. auch Werkzeuge zur Modellerstellung deklarativ spezifiziert werden.

8.7 Ablage zur Versionierung von UI-Modellen

Zur Durchführung einer Komposition müssen Modellartefakte einer KomBInoS-UI in unterschiedlichen Versionierungen gleichermaßen zentral hinterlegt als auch für andere

[5] siehe http://eclipse.org/sirius/ (letzter Zugriff: 04.11.2017)

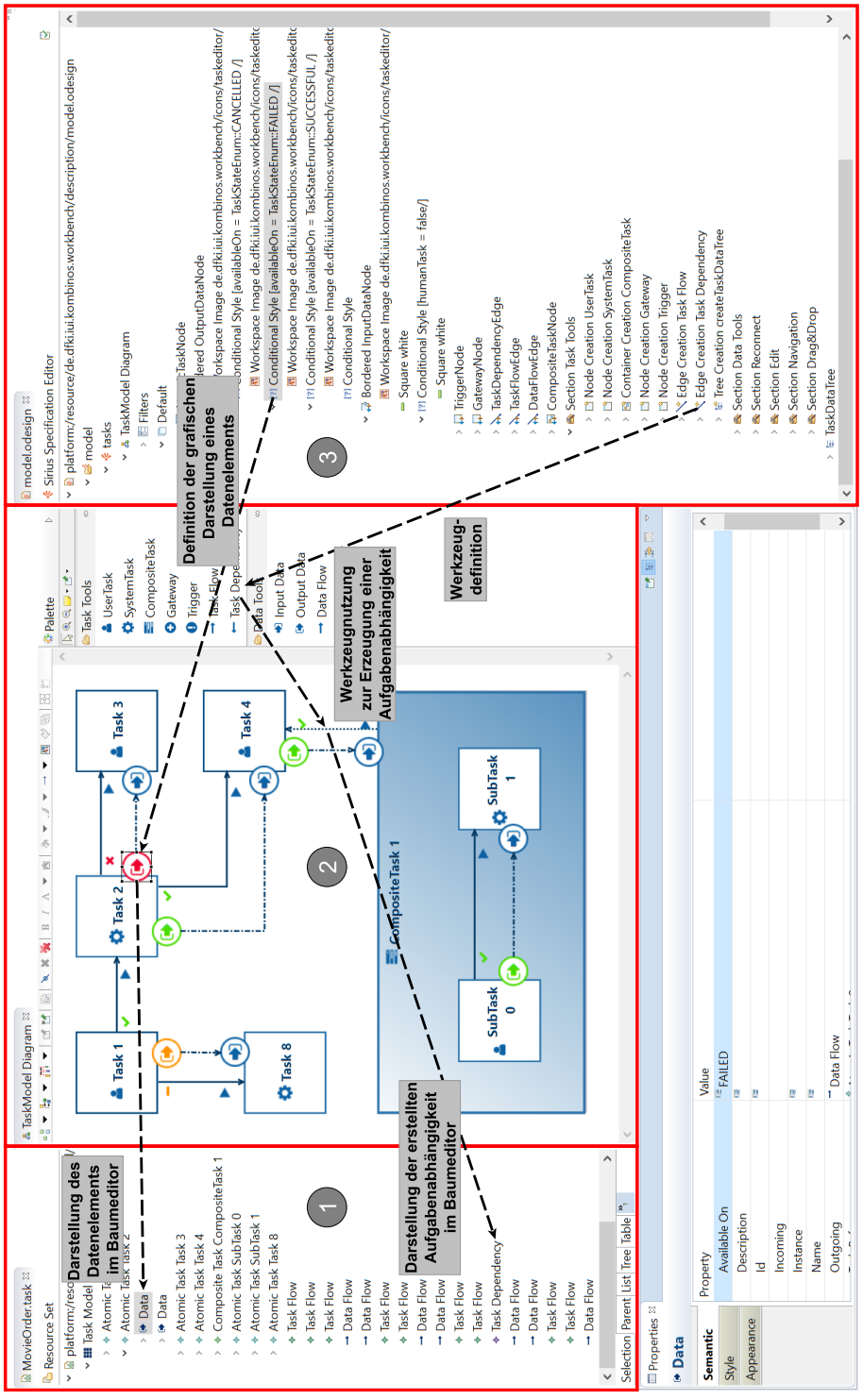


Abbildung 8.7: Grafischer Taskeditor samt deklarativer Editordefinition

Entwickler abrufbar sein. Herkömmliche Versionskontrollsysteme wie Git oder Subversion arbeiten lediglich auf der konkreten textuellen Syntax der serialisierten Modelle. Im Gegensatz dazu arbeitet der EMFStore[6] als Modellablage für Ecore-basierte Modelle direkt auf der abstrakten Syntax der entsprechenden Metamodelle. Dadurch können Konflikte auf Modellebene besser erkannt und behoben werden. Dies wird durch die inhärente Fähigkeit von EMF erreicht, Änderungen an Modellen strukturiert zu erfassen und interessierten Komponenten ereignisbasiert zur Verfügung zu stellen. Hierzu müssen die Modelle in Form der umschließenden EMF-Projekte unter Versionsverwaltung stehen, wodurch ein kollaboratives Editieren der Modelle möglich wird.

Ein EMFStore-Client zeichnet nun lokal alle Modelländerungen auf und übermittelt diese im Falle einer gewünschten Synchronisierung mit der Ablage gemäß der in Kapitel 7.2 (S. 143) geschilderten Methodenanforderung (5) bzgl. Komposition als Delta dem EMFStore-Server. Im Falle einer Aktualisierung lokaler Modelle vom Server wird ebenfalls ein Delta an den Client übertragen, der die Änderungen lokal nachvollzieht. Per EMFStore-API[7] können zudem Deltas zwischen unterschiedlichen Modellversionen abgefragt werden.

Ein speziell für KomBInoS konfigurierter EMFStore-Server ist über Inter- oder Intranet zugreifbar. Dieser enthält die für die KomBInoS-Metamodelle notwendigen Abhängigkeiten in Form von zusätzlichen OSGi-Bündeln. In der KomBInoS-Werkbank selbst wird der zur Verfügung stehende EMFStore-Client als GUI nahtlos in die KomBInoS-Perspektive integriert. Darüber hinaus wird über die EMFStore-API eine noch tiefere Integration erreicht.

- In den KomBInoS-Einstellungen (Abbildung 8.8) können beim ersten Start der Werkbank der Endpunkt für die KomBInoS-Modellablage sowie Benutzername und Passwort angegeben werden. Die Integration der KomBInoS-spezifischen Ansicht in die Eclipse-Einstellungen erfolgt über den Erweiterungspunkt org.eclipse.ui.preferencePages.
- Nach der Erstellung eines neuen Projekts durch den Projekt-Wizard wird dieses Projekt unverzüglich automatisch mit der konfigurierten Modellablage geteilt. Falls ein Kompositionsmodell angegeben wurde, werden anschließend sämtliche darin referenzierten Projekte in der lokalen KomBInoS-Werkbank verfügbar gemacht. Diese Logik ist Bestandteil des Wizard und wird nach Projekterzeugung (Abbildung 8.1,(3)) im Hintergrund automatisch ausgeführt.
- Projektspezifische Kommandos zur Aktualisierung und Synchronisierung der enthaltenen Modelle sind in bestehende Menüstrukturen, wie das projektspezifische Kontextmenü und das globale KomBInoS-Menü, integriert. Sie stehen außerdem durch entsprechende Schaltflächen im Dashboard zur Verfügung. Bei Modellkonflikten wird der generische EMFStore-Client zur interaktiven Konfliktauflösung

[6] siehe http://eclipse.org/emfstore/ (letzter Zugriff: 04.11.2017)

[7] siehe http://download.eclipse.org/emfstore/releases_18/javadoc/index.html (letzter Zugriff: 04.11.2017)

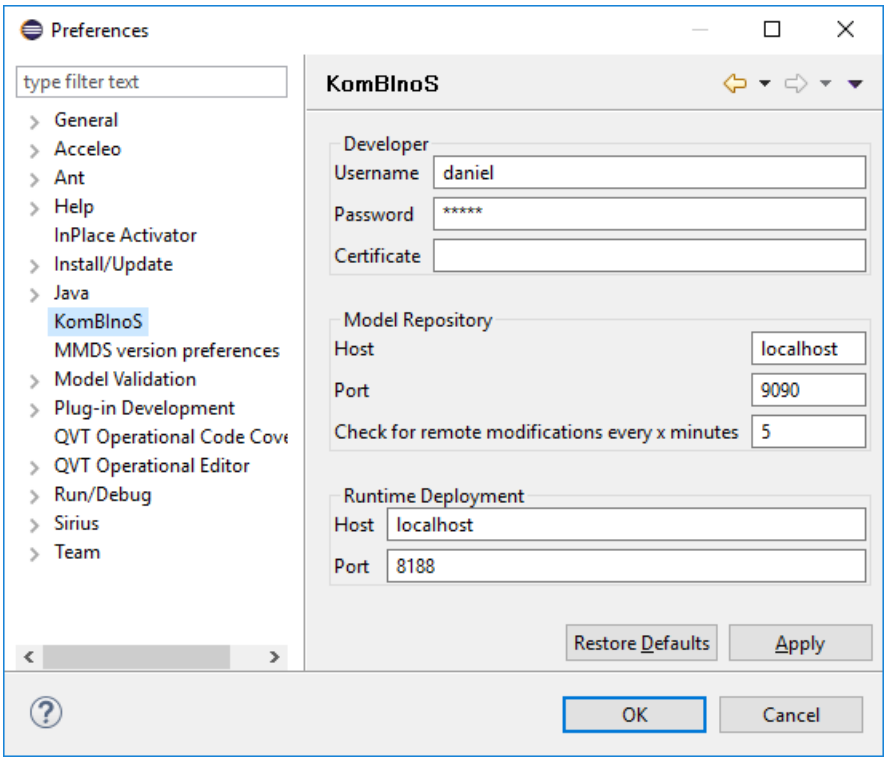


Abbildung 8.8: Dedizierter Bereich innerhalb der zentralen Eclipse-Einstellungen zum Spezifizieren KomBInoS-relevanter Parameter

geöffnet.

- Periodisch fragt ein Java-TimerTask die Modellablage nach von anderen Entwicklern eingespielten Modelländerungen lokal vorhandener Projekte an. Die Zeitspanne kann ebenfalls in den KomBInoS-Einstellungen spezifiziert werden. Eine Zeitspanne von 0 Minuten bedeutet keine Prüfung. Sind Änderungen vorhanden, wird dies zusammen mit Zusatzinformation („*Wer hat was wann geändert?*“) und der entsprechenden Änderungsnachricht dem Entwickler, wie in Abbildung 8.9 ersichtlich, angezeigt. Über eine „*Jetzt aktualisieren*“-Schaltfläche kann die Änderung schließlich lokal eingespielt werden.
- Die Bündelung und Bereitstellung einer KomBInoS-UI löst automatisch eine obligatorische Übergabe des letzten Modellstandes an die Modellablage aus.

8.8 Fallbasiertes Schließen auf Ecore-Metamodellen

Die Vervollständigung von Modellen zur natürlichsprachlichen Interaktion erfolgt durch fallbasiertes Schließen (CBR). Fallbasiertes Schließen ist nach Aamodt und Plaza (1994) eine auf Erfahrungen beruhende Problemlösungsstrategie und wird als solche dem über-

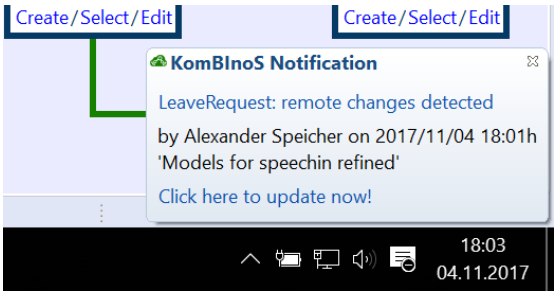


Abbildung 8.9: Benachrichtigung über extern eingespielte Modelländerungen

wachten maschinellen Lernen zugeordnet. Erfahrungen beschreiben konkrete Problemsituationen samt deren Lösung. Sie werden in Form von Fällen in einer Fallbasis abgelegt. Diese Fallbasis wird zur Lösung eines neuen Problems herangezogen, indem durch eine ähnlichkeitsbasierte Suche vergleichbare Fälle ermittelt und deren Lösungen auf die aktuelle Situation übertragen und angepasst werden. Eine anschließend manuell verfeinerte Lösung wird überprüft, verbessert, bewertet und gegebenenfalls als ein neuer Fall in der Fallbasis abgelegt. Die Nachvollziehbarkeit von Lösungen führt zu einer hohen Akzeptanz der Methode. Die Lösungsfindung folgt dabei dem sogenannten CBR-Zyklus, welcher in Abbildung 8.10 dargestellt ist.

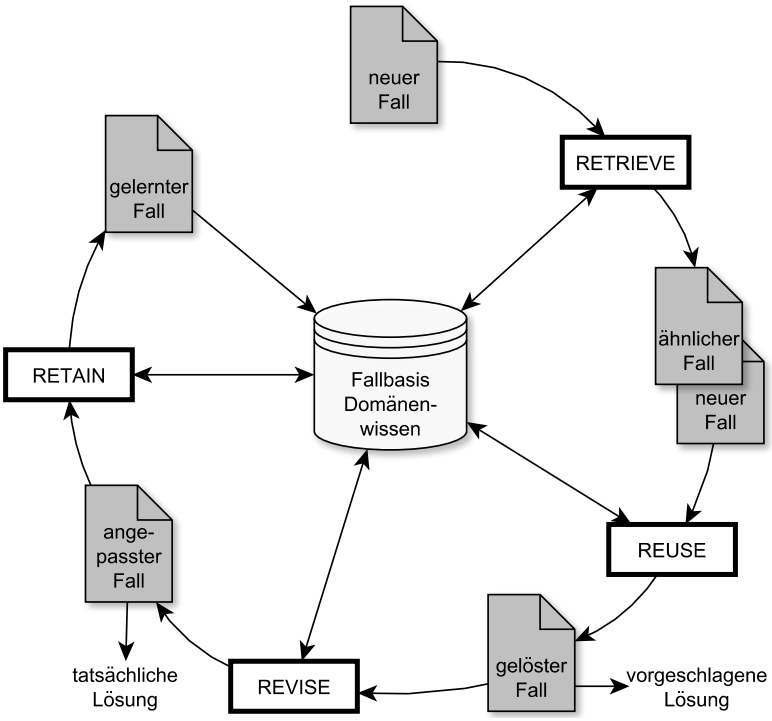


Abbildung 8.10: Der CBR-Zyklus

- **Retrieve:** Die Fallbasis wird nach ähnlichen Fällen angefragt.
- **Reuse:** Die korrespondierenden Lösungen der ermittelten ähnlichen Fälle werden auf die neue Situation übertragen und adaptiert.
- **Revise:** Die ermittelte Lösung wird auf Korrektheit überprüft und bei Bedarf verbessert.
- **Retain:** Die verbesserte Lösung wird samt Problembeschreibung als neuer Fall in die Fallbasis integriert.

Die initiale technische Umsetzung des hier vorgestellen Ansatzes zum fallbasierten Schließen auf Ecore-Metamodellen erfolgte im Rahmen der Bachelorarbeit von Speicher (2015) und beruht auf der CBR-Plattform jCOLIBRI (Recio-García u. a. 2014). Die Einbindung von jCOLIBRI in Eclipse ist aufgrund der OSGi-konformen Strukturierung der Software reibungslos. Ebenso zählen konfigurierbare Selektions- und Klassifikationsalgorithmen, u. a. k-Nächste Nachbarn (Russell und Norvig 2003, S. 733), zum Lieferumfang der quelloffenen Plattform[8]. Anhand der ausführlichen Dokumentation lassen sich eigene Erweiterungen transparent umsetzen. So wurde jCOLIBRI im Rahmen dieser Arbeit maßgeblich für fallbasiertes Schließen auf Ecore-basierten Metamodellen erschlossen. Die zugrundeliegende MVC-basierte Architektur der Umsetzung ist in Abbildung 8.11 dargestellt und wird im Folgenden detailliert erläutert.

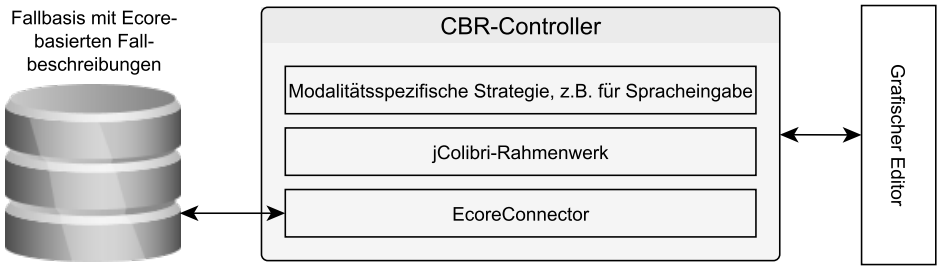


Abbildung 8.11: Architektur der MVC-basierte CBR-Anwendung

Zunächst wurde mit Hilfe von EMF auf Basis von Ecore ein generisches Metamodell geschaffen, welches als Ausgangspunkt zur Modellierung domänenspezifischer Fallrepräsentationen im Rahmen einer Fallbasis dient. Das Metamodell ist in Abbildung 8.12 dargestellt. Es wurden relevante Schnittstellen (CaseComponent, TypeAdaptor, Attribute) des jCOLIBRI-Rahmenwerks innerhalb des Metamodells als externe Datentypen bekanntgemacht, um diese in nachmodellierten Konzepten einbinden zu können. Die Fallbasis (EcoreCaseStore) enthält alle vorhandenen Fälle (EcoreCase) und stellt diese zur weiteren Verarbeitung zur Verfügung. Das Konzept EcoreCase lehnt sich an den Typen jcolibri.cbrcore.CBRCase an und besteht jCOLIBRI-konform aus den vier Komponenten Fallbeschreibung, Ergebnis, Lösung und Rechtfertigung der Lösung. Die in Abbildung 8.12 gezeigten Typvariablen D, R, S und J dienen als Platzhalter für konkrete, domänenspezifische Komponentenmodellierungen. Die jeweilige Einschränkung der

[8] siehe http://gaia.fdi.ucm.es/research/colibri/jcolibri (letzter Zugriff: 04.11.2017)

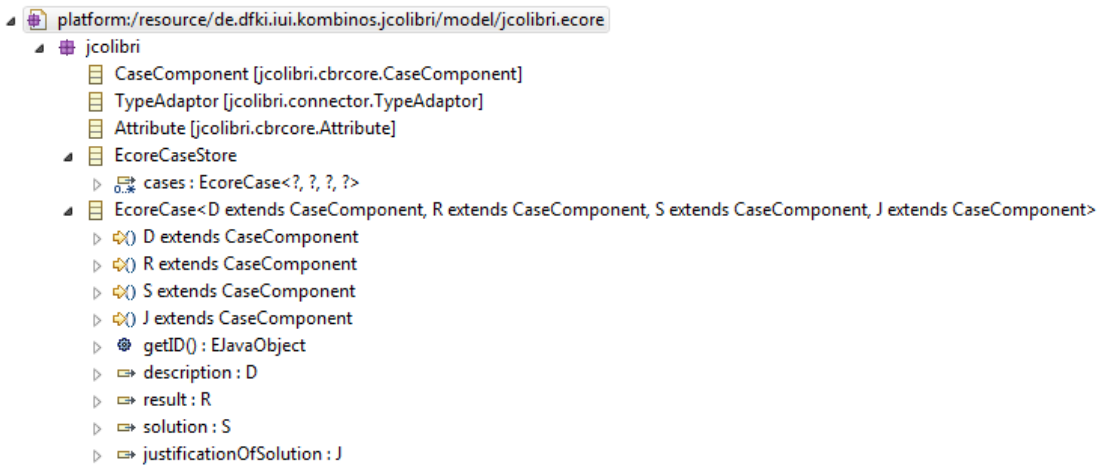


Abbildung 8.12: KomBInoS-spezifische Erweiterung von jColibri zur Verwaltung von Ecore-basierten Fallrepräsentationen

Typvariablen auf CaseComponent sorgt für die gebotene Typsicherheit bei der Spezialisierung.

Die funktionale Integration zwischen einer Ecore-basierten Falbasis und dem jCOLIBRI-Rahmenwerk erfolgt innerhalb des CBR-Controllers durch den eigens entwickelten EcoreConnector, welcher die jCOLIBRI-Schnittstelle jcolibri.cbrcore.Connector implementiert. Er kapselt die Persistierung von Ecore-basierten Fallrepräsentationen und befüllt initial eine Ecore-basierte Fallbasis. Außerdem trägt er Sorge für eine sichere Abbildung zwischen jCOLIBRI-internen Fallrepräsentationen (jcolibri.cbrcore.CBRCase) und den vorliegenden Ecore-basierten Fallrepräsentationen (EcoreCase).

Nach außen in Richtung domänenspezifischer Editoren bietet der CBR-Controller eine einheitliche API an, welche in Abbildung 8.13 gezeigt wird.

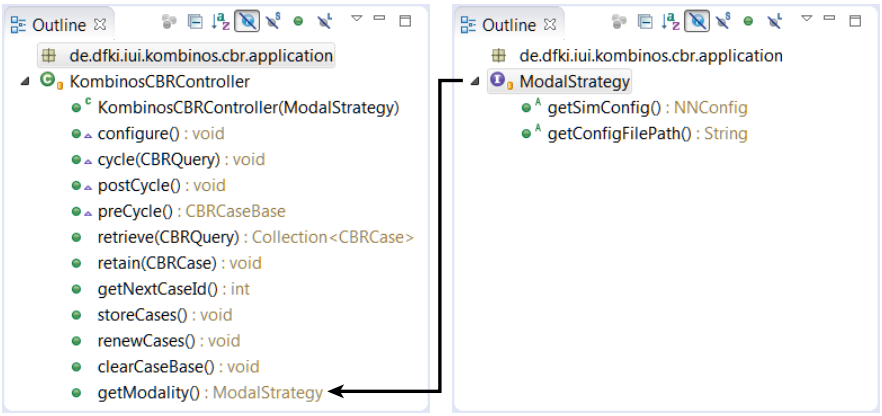


Abbildung 8.13: API des CBR-Controllers

Unter Anwendung des Strategie-Verhaltensmusters (Gamma u. a. 1994, S. 349ff) wird diese API und der dadurch abgebildete CBR-Zyklus durch im vorliegenden Fall modalitätsspezifische Vorgaben für das Retrieval konfiguriert, etwa durch die Angabe spezialisierter Vergleichsalgorithmen und Gewichtungen. Somit wird im Allgemeinen technisch eine breite Einsatzmöglichkeit von fallbasiertem Schließen auf Ecore-basierten Fallrepräsentationen bei maximaler Wiederverwendbarkeit generischer Codeanteile erreicht.

Um diesen Abschnitt zu beschließen, wird in Abbildung 8.14 auf die modalitätsspezifischen Ausprägungen der Fallrepräsentationen zum Retrieval und Reuse von Nutzer- als auch Systemäußerungen eingegangen. Abbildung 8.14 zeigt unter (1) die Spezialisierung der generischen Ecore-basierten Fallbasis und -beschreibung für die Spracheingabe (SpeechInCaseStore, SpeechInCase) und die Sprachausgabe (SpeechOutCaseStore, SpeechOutCase). Unter (2) sind die spezifischen Komponenten des SpeechInCase dargestellt. Die Fallbeschreibung (SpeechInDescription) enthält aufgrund der Referenz auf das abstrakte Konzept Rule entweder eine vollständige Äußerung (Utterance) oder eine von einer solchen abhängige Entity- bzw. SemanticMapping-Regel, wie in Kapitel 6.7.1 (S. 130) geschildert. Hierbei ist lediglich die semantische Interpretation der Regel verfügbar. Die jeweilige syntaktische Abbildung ist relevanter Bestandteil der je nach Regeltyp anwendbaren Lösungsmodellierung als Spezialisierung der SpeechInSolution. Weiterhin kann z. B. eine Äußerung in ihren Phrasen Unterregeln referenzieren. Diese sind in der Lösung über die Referenz *subRules* explizit enthalten. Der spezifische Vergleichsoperator für CBR auf Spracheingabemodellen prüft in der Retrieve-Phase nun auf Ähnlichkeit zwischen den semantischen Interpretationen der vorliegenden Problembeschreibung sowie der vorhandenen Fallbeschreibungen. Dabei ist die pure Anwendung der in SiAM-dp verfügbaren Algoritmen zur Unifikation oder zum Musterabgleich (Neßelrath 2016, S. 89ff.) aufgrund zahlreicher hier irrelevanter aber sicher unterschiedlicher Informationseinheiten zu streng. Die betroffenen Slots müssen also vor einem Vergleich temporär bereinigt werden. Außerdem sind zusätzliche Relaxierungen bzw. Abstrahierungen von Informationseinheiten vorteilhaft, die dann etwa mit einem geringeren Gewicht in das Ähnlichkeitsmaß eingehen, welches für einen Eingabeakt *ia* und eine Fallbeschreibung *cd* nach folgendem Schema berechnet wird:

(1) Bereinige *ia* um irrelevante Information, z. B. Referenzen auf Aufgaben oder Rollendefinitionen.

(2) Vergleiche die Konzepte der kommunikativen Funktionen von *ia* und *cd*. Sind diese im Rahmen der Taxonomie der kommunikativen Funktionen (Kapitel 4.2.2, S. 51) zwar verschieden aber in der Bedeutung (noch) nicht gegensätzlich (z. B. SetQuestion und Question), dann vermindere die initiale Ähnlichkeit von 1.0 um einen Faktor (0.2 * Abstand). Gültige Kombinationen müssen zuvor entsprechend modelliert worden sein.

(3) Vergleiche die semantischen Inhalte der kommunikativen Funktionen von *ia* und *cd* mit Hilfe des Overlay-Algorithmus (Kapitel B, S. 271). Der Algorithmus liefert bei Gleichheit ein Ähnlichkeitsmaß von 1.0. Addiere dieses Ähnlichkeitsmaß mit dem aus dem vorangegangen Schritt.

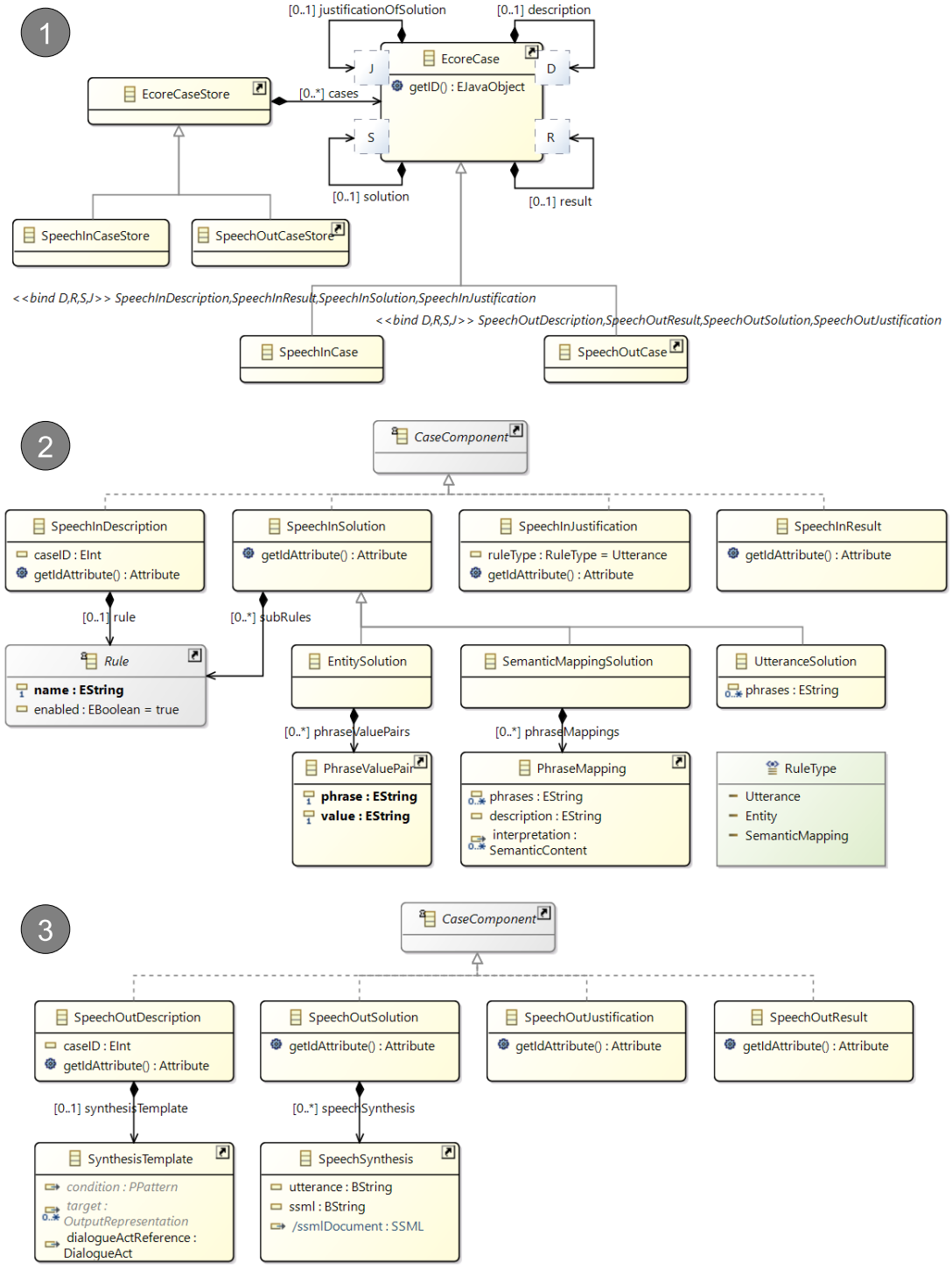


Abbildung 8.14: Modell der Fallbeschreibungen von Nutzer- und Systemäußerungen

(4) Normalisiere das Ähnlichkeitsmaß durch Division von 2.

Das Ähnlichkeitsmaß *sim* für einen Eingabeakt *ia* und eine Fallbeschreibung *cd* ergibt sich also nach Bereinigung in (1) sowie gleicher Gewichtung der Punkte (2) und (3) durch die Formel.

$$sim = \frac{1.0 - dist(ia.comFkt, cd.comFkt) + overlayscore(ia.semCont, cd.semCont)}{2}$$

Im Punkt (3) der Abbildung 8.14 sind die spezifischen Komponenten des SpeechOutCase dargestellt. Die Fallbeschreibung enthält ein Synthesetemplate, welches wiederum einen Dialogakt als semantische Interpretation enthält, die zum Vergleich in einem analog aufgebauten Vergleichsoperator herangezogen wird. Aufgrund des einfacheren Modellaufbaus (Kapitel 6.7.2, S. 131) muss die Lösungsbeschreibung lediglich eine Sprachsyntheseanweisung (SpeechSynthesis) enthalten.

Für die CBR-unterstützte Erstellung von Modellen zur Spracheingabe und -ausgabe wurden spezialisierte Editoren entlang des CBR-Zyklus implementiert. Die Editoren sind im Zusammenspiel mit dem CBR-Controller in der Lage, die jeweilige Fallbasis nach ähnlichen Fällen mittels domänenspezifischer Vergleichsalgorithmen anzufragen (Retrieve), die möglichen Problemlösungen nach Ähnlichkeit gruppiert darzustellen (Reuse) und die händisch überarbeiteten Lösungen (Revise) wieder in die Fallbasis zu überspielen (Retain). Abbildung 8.15 zeigt den CBR-Editor für Spracheingabemodelle.

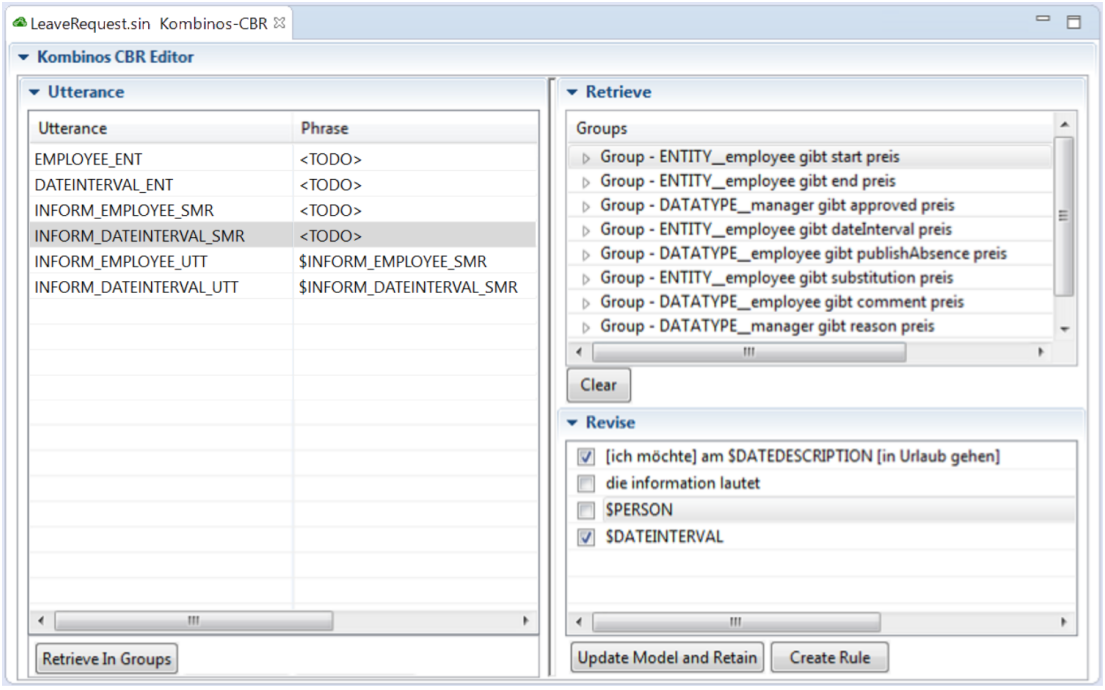


Abbildung 8.15: CBR-Editor für Spracheingabemodell

8.9 Einbringung in die KomBInoS-Laufzeitumgebung

Ressourcenbündelung ist eines der wesentlichen Merkmale des Cloud Computings (Kapitel 3.4.4, S. 46). Darum kann die im nächsten Kapitel vorgestellte Laufzeitumgebung mehrere KomBInoS-UIs gleichzeitig verwalten. Um zum Ende der Entwurfszeit nach erfolgreichen lokalen Tests eine KomBInoS-UI direkt aus der Werkbank heraus auf eine externe Laufzeitumgebung aufzuspielen, ist ein entsprechender Bereitstellungsprozess notwendig. Das generelle Vorgehen dazu ist in Abbildung 8.16 dargestellt.

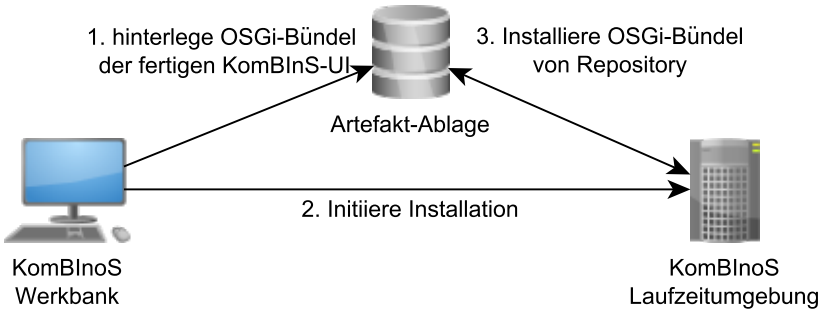


Abbildung 8.16: Deployment einer KomBInoS-UI

Die Einbringung einer KomBInoS-UI in den Wirkbetrieb kann u. a. direkt aus dem Dashboard heraus angestoßen werden. Hierbei wird mit Hilfe des im Projekt enthaltenen Ant-Skripts (build.xml) zunächst ein versioniertes OSGi-Bündel als komprimiertes Java-Archiv gebaut. Dieses wird dann in eine zentrale Artefaktablage übertragen, von wo aus es mit einer Abhängigkeitsverwaltung wie etwa Apache Maven[9] in darauf aufbauende Software eingebunden werden kann. Als Artefaktablage wird hier der Sonatype Nexus[10] aufgrund seiner freien Verfügbarkeit und leichten Inbetriebnahme eingesetzt. Zuletzt wird die KomBInoS-Laufzeitumgebung instruiert, das soeben publizierte Bündel, welches die fertige KomBInoS-UI enthält, herunterzuladen, zu installieren und zu starten. Um dies möglichst mit Bordmitteln umsetzen zu können, basiert die KomBInoS-Laufzeitumgebung auf Apache Karaf[11] und der OSGi-Implementierung Equinox, die auch die Grundlage der Eclipse IDE darstellt. Apache Karaf erweitert Standard-OSGi-Implementierungen wie Apache Felix oder Eclipse Equinox um reichhaltige Verwaltungsfunktionen, etwa das hier benötigte direkte Installieren von OSGi-Bündeln aus Artefakt-Repositories durch einen Kommandozeilenbefehl. Die eigentliche Instruktion zum Installieren und Starten des soeben publizierten Bündels erfolgt dann auch per Secure Shell-Verbindung zur Karaf-basierten Laufzeitumgebung aus der Werkbank, bzw. dem Ant-Skript heraus. Listing 8.2 zeigt exemplarisch den entsprechenden Ausschnitt des Skripts samt Kommandozeilenbefehl. Die darin benötigten Zugangsdaten zum Aufbau der SSH-Verbindung werden aus den KomBInoS-Einstellungen bezogen.

[9] siehe https://maven.apache.org/ (letzter Zugriff: 04.11.2017)
[10] siehe http://www.sonatype.org/nexus/ (letzter Zugriff: 04.11.2017)
[11] siehe https://karaf.apache.org/ (letzter Zugriff: 04.11.2017)

Listing 8.2: Ausschnitt des Ant-Skripts zum Installieren einer KomBInoS-UI via SSH

```
<project name="<name>" default="dist" basedir=".">
  <property name="groupId" value="de.dfki.iui.kombinos"/>
  <property name="artifactId" value="de.dfki.iui.kombinos.<name>"/>
  <property name="version" value="1.0.0"/>
  ...
  <target name="install" depends="upload">
    <sshexec host="${host}"
      username="${username}"
      password="${password}"
      command="bundle:install -s mvn:${groupId}/${artifactId}/${version}"
    />
  </target>
</project>
```

8.10 Fazit

In diesem Kapitel wurde die KomBInoS-Werkbank als nahtlose Werkzeugkette auf Basis der Eclipse-Plattform und Implementierung der in Kapitel 7 (S. 143) behandelten integrierten Entwicklungs- und Kompositionsmethode vorgestellt. Abbildung 8.17 zeigt zusammenfassend einen Überblick über die Softwarearchitektur der Werkbank.

Ein Projektwizard erstellt das Grundgerüst einer KomBInoS-UI. Der Entwickler wird von einem Dashboard Schritt für Schritt durch den modellgetriebenen Erstellungsprozess geleitet. Hierbei kann er direkt aus dem Dashboard heraus Modelltransformationen anstoßen. Zur manuellen Vervollständigung der dadurch sukzessive erzeugten Teilmodelle stehen geeignete Editoren zur Verfügung. Diese sind in einer eigenen Perspektive angeordnet, um alle KomBInoS-relevanten Informationsaspekte auf einen Blick erfassen zu können. Hervorzuheben sind die speziellen Editoren für Spracheingabe- und Sprachausgabemodelle, welche die Infrastruktur zum fallbasierten Schließen auf Ecore-basierten Metamodellen für den Entwickler vollständig kapseln. Dennoch können aufgrund der gleichen technologischen sowie Modellbasis auch alle von SiAM-dp bereitgestellten Editoren und Werkzeuge in der KomBInoS-Werkbank genutzt werden. Diese sind vollständig integriert und werden zusätzlich zu dem bisher genannten vervollständigt durch (1) die KomBInoS-Modellablage zur Unterstützung einer verteilten UI-Entwicklung in Teams durch Versionierung und Synchronisation von Modellartefakten auf semantischer Ebene sowie (2) der Möglichkeit zur direkten Inbetriebnahme einer fertigen KomBInoS-UI in die im nächsten Kapitel adressierte Laufzeitumgebung. Die KomBInoS-Werkbank deckt den vielschichtigen Erstellungsprozess technisch vollständig ab. Methodisch wird der Entwickler sicher durch diesen komplexen Prozess geleitet. Der aktuelle Stand der Erstellung, ausstehende Tätigkeiten und Abhängigkeiten sind ihm so jederzeit bekannt. Außerdem erfährt der Entwickler Unterstützung bei der verteilten UI-Erstellung im Team. Bei der Realisierung der KomBInoS-Werkbank wurde ebenfalls ein modellgetriebener Ansatz verfolgt. Quantitative Kennzahlen zur Werkbank befinden sich in Anhang

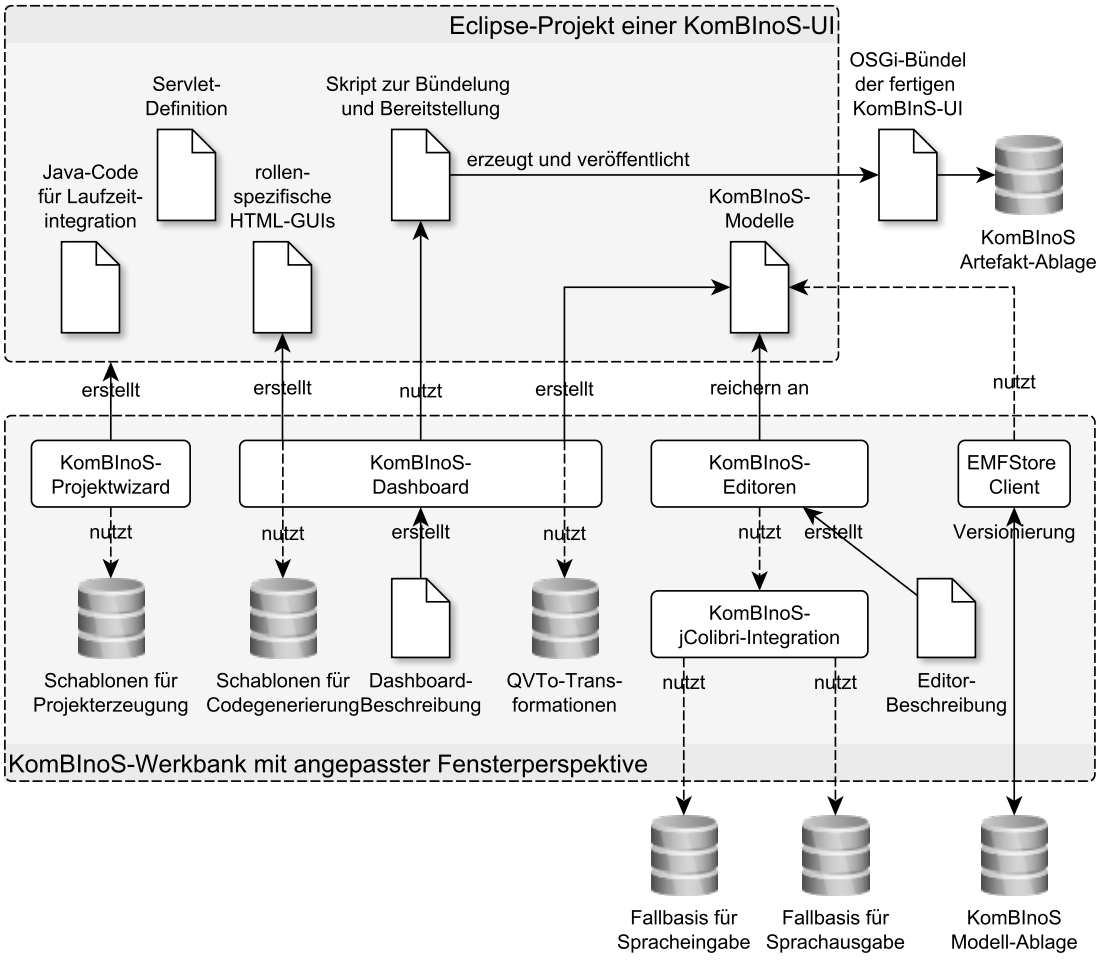


Abbildung 8.17: Architektur der KomBInoS-Werkbank

A (S. 269).

Insgesamt wird somit die in Kapitel 1.2.3 (S. 4) formulierte ingenieurtechnische Fragestellung (4) „*Wie können Entwickler bei der Erstellung von multimodalen Dialogschnittstellen für Smart Services bei ihrer Aufgabe unterstützt werden?*“ umfassend beantwortet.

Teil III

Mobile Nutzungsumgebung und Anwendungen

KAPITEL 9

Die KomBInoS-Laufzeitumgebung

Dieses Kapitel behandelt die KomBInoS-Laufzeitumgebung zum mobilen multimodalen Dialog mit Smart Services. Die Komponenten der Laufzeitumgebung sind verteilt und kommunizieren untereinander durch etablierte Kommunikationsstandards. Das Kapitel beantwortet somit die noch verbliebene ingenieurtechnische Fragestellung (5) bzgl. einer offenen und mobilen Nutzungsumgebung für KomBInoS-UIs.

9.1 Einleitung

Eine Laufzeitumgebung zum mobilen multimodalen Dialog mit Smart Services durch KomBInoS-UIs benötigt unterschiedlichste Komponenten mit komplementären Aufgaben, die in einer verteilten Infrastruktur über Standards kommunizieren. So werden formal modellierte und implementierte Prozesse im betrieblichen Umfeld üblicherweise durch dafür vorgesehene Prozessausführungsmaschinen verwaltet und ausgeführt. Solche Maschinen führen ebenso die notwendigen Dienstaufrufe durch und stellen die auf diesem Wege erhaltenen fachlichen Daten für den weiteren Prozessverlauf bereit. Sie sind skalierbar und unterstützen eine große Zahl unterschiedlicher Prozesse sowie parallel laufender Prozessinstanzen.

Gerade wenn mehrere Benutzer, Gruppen oder allgemein Agenten, um gegebenenfalls auch nicht menschliche Prozessteilnehmer im Rahmen kollaborativer Robotik mit einzuschließen, an einer Prozessausführung beteiligt sind, so liegt die Dauer der vollständigen Bearbeitung aufgrund unterschiedlicher Verfügbarkeiten nicht im Bereich von wenigen Minuten, sondern vielmehr im Bereich von Stunden oder auch Tagen. Aus Sicht eines einzelnen Prozessbeteiligten, der unter Umständen mehrfach in eine Ausführung involviert ist, handelt es sich also um eine langlaufende Interaktion, die durch Unterbrechungen unterschiedlicher Dauer gekennzeichnet sein kann. In Abhängigkeit von der Unterbrechungsdauer ist es folglich notwendig, einen Benutzer erneut in den Pro-

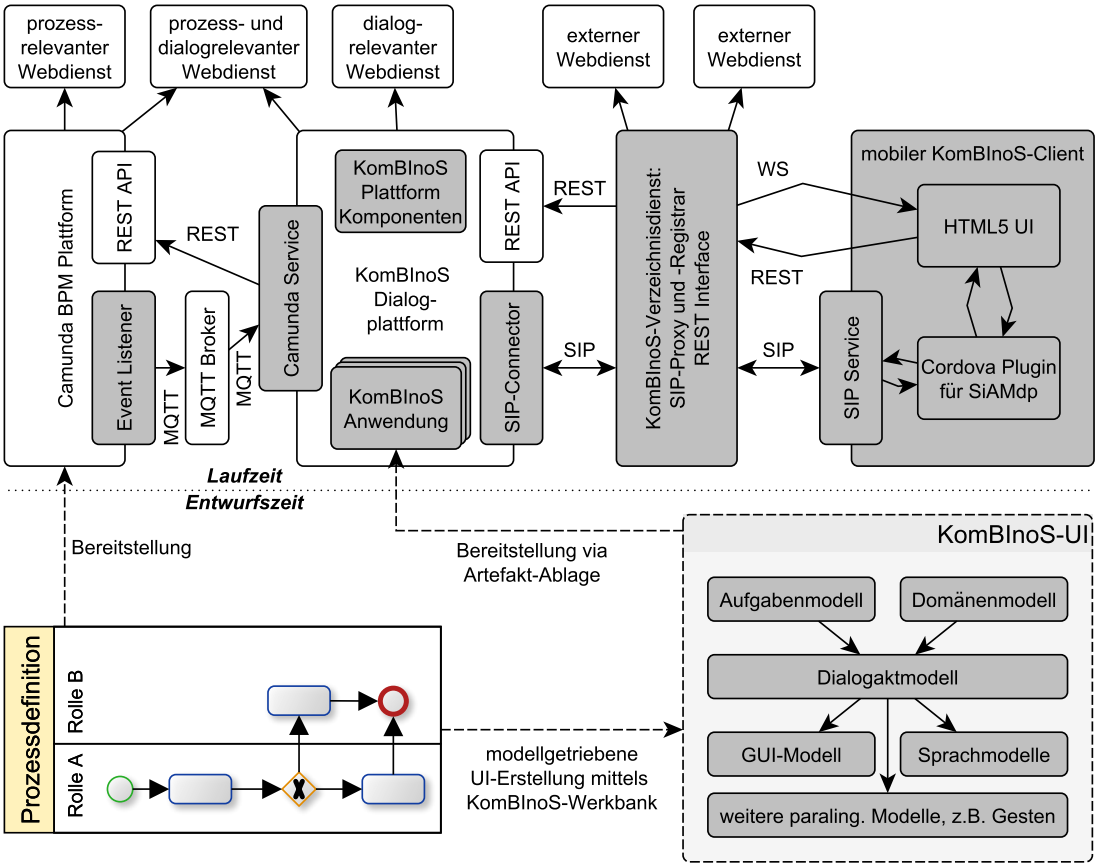


Abbildung 9.1: Die Verbundarchitektur der KomBInoS-Laufzeitumgebung

zesskontext einzuführen. Dies kann bis zu einem gewissen Grad durch eine intelligente Dialogplattform ohne Implikationen auf den zuvor durchlaufenen UI-Erstellungsprozess bewerkstelligt werden, etwa auf Ebene eines Metadialogs oder durch eine grafische Darstellung des Prozessfortschritts. Natürlich obliegt es einer solchen Plattform in erster Linie, den fachlichen Dialog mit Prozessbeteiligten unter Berücksichtigung des aktuellen Interaktionskontextes zu führen, um einen Prozess effizient zum intendierten Ende voranzutreiben. Ein Verzeichnisdienst hilft dabei, auf logischer Ebene von den konkreten Interaktionsgeräten eines Benutzers - in der vorliegenden Arbeit vornehmlich mobile Endgeräte - zu abstrahieren.

Im weiteren Verlauf dieses Kapitels wird die dementsprechende Verbundarchitektur der KomBInoS-Laufzeitumgebung vorgestellt und detailliert auf die einzelnen Bestandteile eingegangen.

9.2 Verbundarchitektur der KomBInoS-Laufzeitumgebung

Die Verbundarchitektur der KomBInoS-Laufzeitumgebung ist in Abbildung 9.1 dargestellt. Sie setzt sich aus vier großen Komponenten zusammen:

(1) der **Camunda BPM-Plattform**, welche einen oder mehrere instanziierbare Kollaborationprozesse in BPMN- oder CMMN-Notation enthält, die die Grundlage zur Entwicklung von KomBInoS-UIs darstellen,

(2) der **KomBInoS-Dialogplattform**, einer erweiterten SiAM-Dialogplattform auf Basis der OSGi-Serviceplattform, speziell für die multimodale Dialoginteraktion mit Smart Services,

(3) dem **KomBInoS-Verzeichnisdienst** mit Proxyfunktionalität und Integrationsmöglichkeiten für externe Dienste, sowie

(4) dem **KomBInoS-Client** als leichtgewichtige mobile Anwendung zur multimodalen dialogischen Nutzung eines Smart Service durch eine KomBInoS-UI.

Die Komponenten sind logisch wie auch technisch voneinander getrennt. Sie können auf unterschiedliche Netzwerkknoten verteilt werden. Die Kommunikation untereinander erfolgt über unterschiedliche Netzwerkkanäle und adäquate, standardisierte Protokolle. Die Komponenten werden in den folgenden Unterkapiteln vorgestellt und detailliert beschrieben. Dabei wird auch auf die ebenfalls abgebildeten Subkomponenten eingegangen.

9.3 Die Camunda BPM-Plattform

Die Camunda BPM-Plattform[1] ist eine quelloffene Plattform zur Automatisierung von Arbeitsabläufen und Geschäftsprozessen. Die Plattform ist in der Programmiersprache Java implementiert und in der Lage, sowohl BPMN 2.0- als auch CMMN 1.0-basierte Geschäftsprozesse direkt auszuführen und zu verwalten. Ebenso bringt die Plattform eine leichtgewichtige Benutzer-, Prozess- und Aufgabenverwaltung mit. Abbildung 9.2 zeigt die Aufgabenverwaltung und Abbildung 9.3 das Management-Cockpit.

Zur technischen Interaktion mit Prozessen über die KomBInoS-Plattform ist es nun einerseits erforderlich, dass diese über relevante Ereignisse bei der Prozessausführung von der Camunda-Plattform in Kenntnis gesetzt wird. Andererseits muss die KomBInoS-Plattform in laufende Prozessinstanzen eingreifen können.

Die reichhaltige Camunda REST API erlaubt es hierzu Prozesse auf der Plattform in Betrieb zu nehmen und zu verwalten, Prozessinstanzen zu starten sowie Aufgaben zu delegieren und zu bearbeiten. Für die gezeigte Verbundarchitektur wird die Camunda BPM-Plattform in Kombination mit einem Wildfly-Applikationsserver[2] eingesetzt. Ein entsprechendes Softwarebündel kann ohne Konfigurationsaufwand installiert und ge-

[1] siehe http://camunda.org/ (letzter Zugriff: 04.11.2017)
[2] siehe http://wildfly.org/ (letzter Zugriff: 04.11.2017)

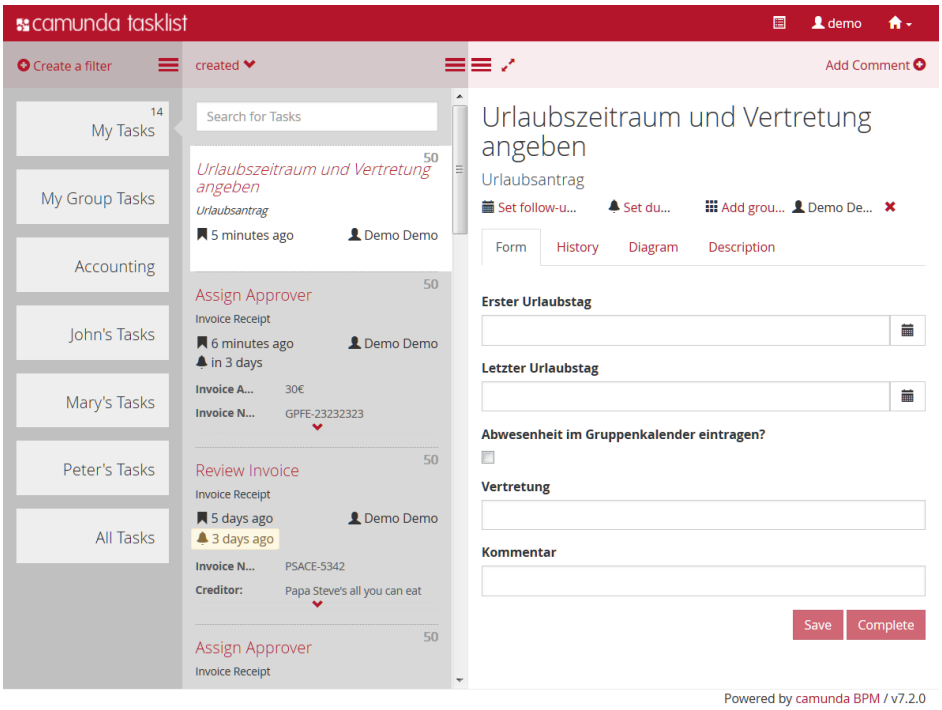


Abbildung 9.2: Die Camunda Aufgabenverwaltung samt Eingabemaske zur Bearbeitung einer ausgewählten zugewiesenen Aufgabe

nutzt werden. Der Wildfly-Applikationsserver unterstützt zudem die Spezifikation der Java Enterprise Edition in der aktuellen Version 7, welche zusätzliche unternehmensrelevanten Softwarekomponenten und Dienste innerhalb der Java-Laufzeitumgebung zur Verfügung stellt. So wurde zur Implementierung der in Listing 9.1 gezeigten Ereignisbehandlung auf den Contexts and Dependency Injection (CDI)-Mechanismus zurückgegriffen, um auf elegante Weise über prozessrelevante Ereignisse während der Ausführung eines Prozesses informiert zu werden. Mit einem Einsatz von Camunda im alternativ angebotenen Apache Tomcat-Applikationsserver wäre dies aufgrund des fehlenden CDI-Mechanismus nicht ohne Weiteres möglich gewesen. Die via CDI intern empfangenen Ereignisse werden nun mit Hilfe des im Internet der Dinge populären Protokolls Message Queue Telemetry Transport (MQTT)[3] auf einem MQTT-Broker publiziert. MQTT ist ein leichtgewichtiges Publish-Subscribe-Kommunikationsprotokoll zur Übertragung von Telemetriedaten. Die Nutzung von MQTT erleichtert aufgrund der dortigen Verbreitung zudem den verstärkten Einsatz von KomBInoS im Industrie 4.0-Umfeld. Ein erstes Anwendungsbeispiel im Rahmen des Projekts Hybr-IT ist in Kapitel 10.8 (S. 247) beschrieben. Über entsprechende Programmierschnittstellen und Softwarebibliotheken kann MQTT in vielen, auch leistungsschwachen, Zielplattformen genutzt werden. Als

[3]siehe http://mqtt.org/ (letzter Zugriff: 04.11.2017)

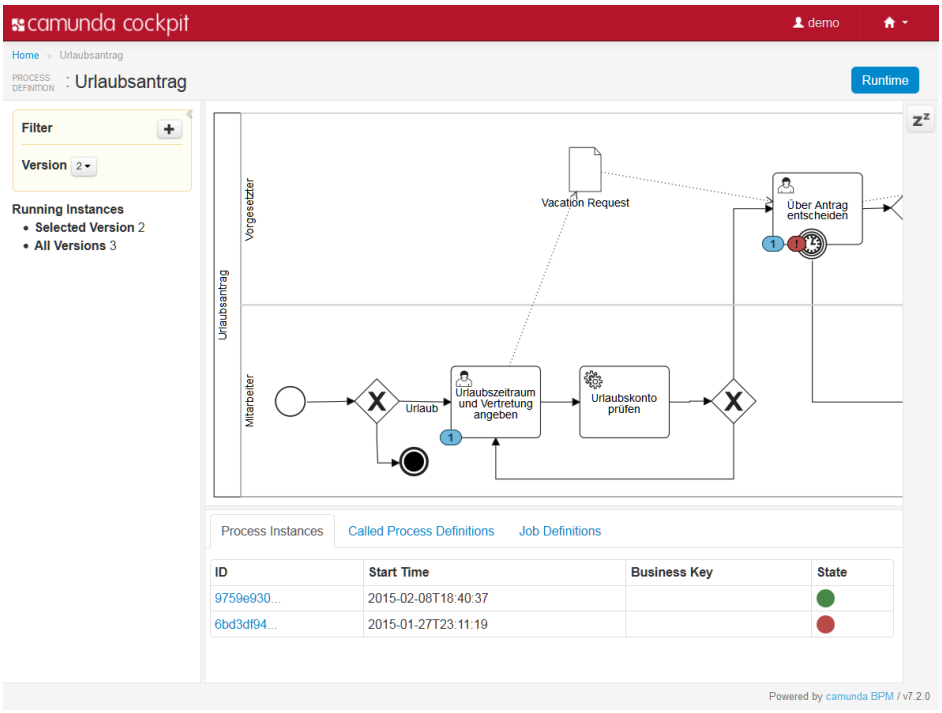


Abbildung 9.3: Das Camunda Management-Cockpit zum Verwalten und Inspizieren von Prozessen und Prozessinstanzen

Java-basierte MQTT-Client-Bibliothek kommt in KomBInoS Eclipse Paho[4] zum Einsatz. Die quelloffene Software deckt den vollständigen Funktionsumfang des Standards ab, ist weit verbreitet und wird durch die Open-Source-Gemeinschaft kontinuierlich gepflegt.

Listing 9.1: Camunda Event Listener mit CDI-Annotationen

```
1  @Singleton
2  @LocalBean
3  public class BusinessProcessEventListener {
4
5    @Inject KombinosMqttClient mqttClient;
6
7    public void onBusinessProcessEvent(@Observes final BusinessProcessEvent
         ev) {
8      BusinessProcessEventDto dto = new BusinessProcessEventDto(ev);
9      String topic = dto.getType() + '/' + dto.getProcessDefinitionId();
10     JSONObject payload = new JSONObject( dto );
11     mqttClient.publish('camunda/' + topic, payload.toString());
12   }
13 }
```

[4]siehe https://eclipse.org/paho/clients/java/ (letzter Zugriff: 04.11.2017)

Tabelle 9.1 führt die relevanten Ereignisse der Prozessausführungsmaschine auf, die gleichzeitig in Verbindung mit einer Konkatenation der Id der Prozessdefinition als Hauptbestandteil von MQTT-Themen fungieren, Diese können interessierte Komponenten, etwa die im nächsten Abschnitt beschriebene KomBInoS-Dialogplattform, abonnieren.

Ereignis	**Beschreibung**
EXECUTION_START	Die Ausführung einer Aktivität wurde begonnen.
EXECUTION_END	Die Ausführung einer Aktivität wurde beendet.
TRANSITION_TAKE	Eine Transition wurde genommen.
TASK_CREATE	Eine Aufgabe wurde instanziiert.
TASK_ASSIGN	Eine instanziierte Aufgabe wurde einem Agenten zugewiesen.
TASK_COMPLETE	Eine instanziierte Aufgabe wurde abgeschlossen.
TASK_DELETE	Eine instanziierte Aufgabe wurde verworfen.

Tabelle 9.1: Relevante Ereignisse während der Ausführung eines Prozesses.

9.4 Die KomBInoS-Dialogplattform für Smart Services

Die KomBInoS-Dialogplattform erweitert die SiAM-Dialogplattform (Neßelrath 2016) um eine Menge von OSGi-basierten Komponenten, speziell für die multimodale dialogische Interaktion mit Smart Services. Dazu zählen:

Die Camunda-Integration zur Interaktion mit Kollaborationsprozessen im Back-End

Die Camunda-Integration ist als OSGi-Service innerhalb der KomBInoS-Plattform umgesetzt. Der Dienst empfängt einerseits Prozess-Ereignisse der Camunda BPM-Plattform via MQTT. Dazu richtet er beim MQTT-Broker entsprechende Abonnements auf die Tabelle 9.1 aufgelisteten Ereignisse ein. Beim Empfang eines solchen Ereignisses wird dieses nun in ein strukturell ähnlich aufgebautes OSGi-Ereignis transformiert und innerhalb der Serviceplattform durch den OSGi-EventAdmin interessierten Diensten und Komponenten, etwa dem TaskManager (Listing 9.3, S. 222), zur Verfügung gestellt.

Andererseits implementiert der Dienst eine reichhaltige Java-API zur direkten Kommunikation mit der Camunda BPM-Platform. Dabei wird die von Camunda zur Verfügung gestellte REST-basierte Dienstschnittstelle gekapselt. Die Java-API wurde mit Hilfe modellgetriebener Techniken zunächst modelliert und anschließend in Teilen generiert. Das Modell ist in Abbildung 9.4 auszugsweise dargestellt.

Die Umsetzung erfolgte mit Hilfe von Jersey[5], der Referenzimplementierung von Had-

[5]siehe https://jersey.java.net/ (letzter Zugriff: 04.11.2017)

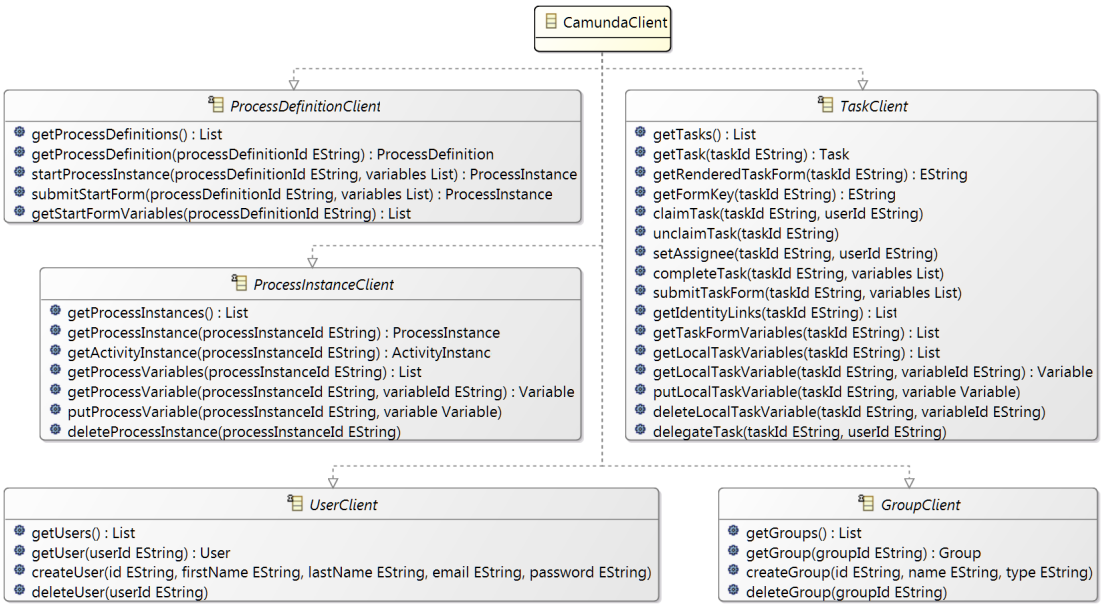


Abbildung 9.4: Die Client-API des Camunda OSGi-Services

ley und Sandozs (2009) Spezifikation Java API for RESTful Web Services. Diese API wird von Plattformkomponenten zur Informationsabfrage und Interaktion mit dem Camunda Back-End verwendet, etwa um Prozesse zu starten, kontextsensitiv Aufgaben an bestimmte Agenten zuzuweisen oder Eingaben im Zuge der Bearbeitung einer Aufgabe in die Prozessausführungsmaschine einzupflegen.

SIP-Connector zum Sitzungs- und Gerätemanagement im Front-End

Während die Kommunikation mit dem Camunda Back-End über MQTT realisiert ist, erfolgt der Verbindungsaufbau im Front-End zwischen einem Klienten und der Dialogplattform durch das Session Initiation Protocol (SIP) (Rosenberg u. a. 2002), welches zugleich die Grundlage für internetbasierte Telefonie darstellt. Jedoch ist der Einsatzzweck von SIP nicht auf Telefonie beschränkt, sondern kann zum Aufbau und Aushandeln von Sitzungen aller Art verwendet werden. Die Beschreibung einer aufzubauenden Sitzung erfolgt im Rahmen von SIP zwingend durch das Session Description Protocol (SDP) (Handley u. a. 2006). Dabei sind die Möglichkeiten zur Ressourcenverhandlung, etwa bzgl. vorhandener oder vorausgesetzter Mediacodecs, die Veränderung von Sitzungen oder deren Übertragung und Erweiterung auf andere Interaktionsgeräte dem Protokoll inhärent. Bei wechselnden Interaktionskontexten wie etwa im SmartKom-Projekt (Wahlster 2006) kann diese Fähigkeit auf technischer Ebene ein nahtloses Benutzererlebnis ermöglichen. Die Transmissionen unterliegen einer bestimmten Dienstgüte, welche auf Anwendungsebene den Empfang von Nachrichten durch Quittierungen sicherstellt und ausbleibende Quittierungen detektiert. Dies ist insbesondere beim verbindungslosen User Datagram Protocol (UDP) als Standard-Transportprotokoll von SIP-Nachrichten

aufgrund fehlender Kontrolle zugunsten einer schnelleren Übertragungsgeschwindigkeit von großer Wichtigkeit.

Die KomBInoS-Dialogplattform kann mittels SIP Agenten bei langlaufenden Prozessen und Aufgaben aktiv benachrichtigen und durch *Anrufen* (wieder) in den Kollaborationsprozess einbeziehen. Der KomBInoS-Verzeichnisdienst, welcher im nächsten Unterkapitel vorgestellt wird, fungiert als *Telefonbuch* und sorgt dabei für die notwendige Abstraktion von konkreten dynamischen Endpunkten, unter denen ein Agent erreichbar ist. SIP ermöglicht zudem die nahtlose Integration netzwerkbasierter Softwarekomponenten zur Spracherkennung und Sprachsynthese. Diese können anschließend in der KomBInoS-Plattform durch das Media Resource Control Protocol (MRCP) in Version 2 (Burnett und Shanmugham 2012) über ein Netzwerk gesteuert werden. Abbildung 9.5 verdeutlicht den Prozess zur Erstellung einer Sitzung samt involvierter Komponenten auf Initiative eines Agenten bzw. seines Klienten.

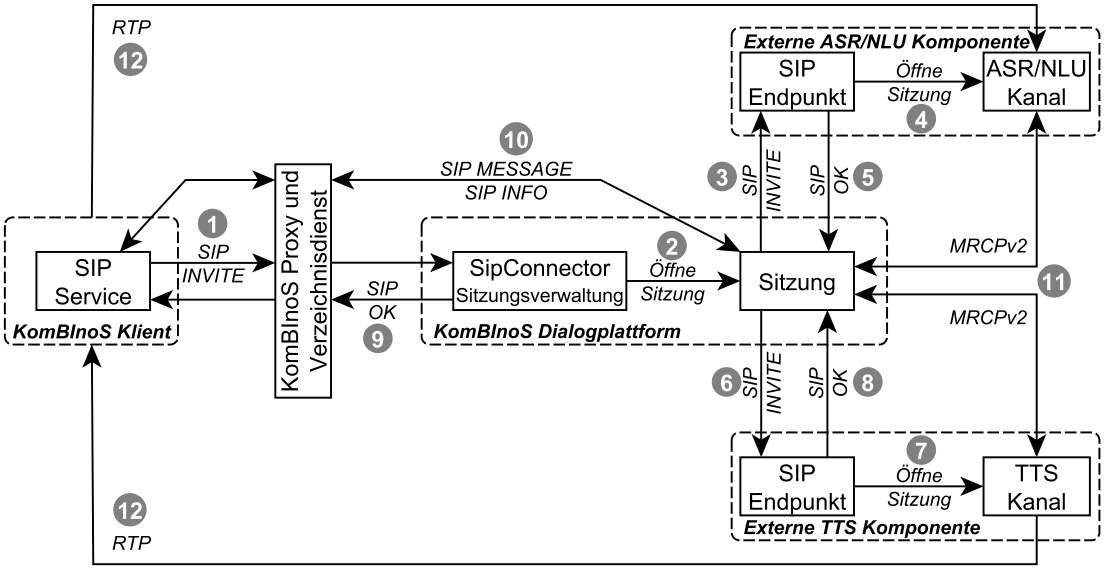


Abbildung 9.5: Prozess zur Erstellung einer Sitzung samt involvierter Komponenten

(1) Ein KomBInoS-Klient wünscht den Verbindungsaufbau mit der Dialogplattform und sendet eine SIP INVITE Nachricht über den KomBInoS-Verzeichnisdienst und Proxy an die Dialogplattform. Darin ist mit Hilfe von SDP codiert, dass bidirektionale Sprachinteraktion präferiert wird.

(2) Die Dialogplattform erzeugt mit Hilfe der Sitzungsverwaltung eine Sitzung, die später auch über diesen bezogen werden kann.

(3) Die Sitzung fragt beim Netzwerkspracherkenner einen eingehenden Sprachkanal an.

(4) Ist ein solcher verfügbar, etwa aufgrund ausreichender technischer und lizenzrechtlicher Kapazitäten, wird ein Kanal geöffnet.

(5) Der Spracherkenner meldet zurück, dass fortan ein Kanal für den Agenten zur Verfügung steht.

(6) Die Sitzung fragt bei der über Netzwerk verfügbaren Sprachsynthese einen ausgehenden Sprachkanal an.

(7) Ist ein solcher verfügbar, wird ein Kanal geöffnet.

(8) Die Sprachsynthese meldet, dass ein Kanal für den Agenten geöffnet wurde.

(9) Nun meldet die Dialogplattform dem Klienten zurück, dass der Verbindungsaufbau mit den gewünschten Interaktionskanälen erfolgreich war.

(10) Fortan können textbasierte Steuercodes, z. B. zum Öffnen des Mikrofons oder zum Mitteilen von Touch-Gesten, mittels SIP MESSAGE Nachrichten bidirektional zwischen Klient und einer Sitzung in der Dialogplattform ausgetauscht werden. Im Gegensatz dazu können SIP INFO Nachrichten auch ohne aktive Sitzung immer ausgetauscht werden, z. B. zum Signalisieren von Neuigkeiten.

(11) Die Steuerung der Sprachkanäle erfolgt plattformseitig durch das MRCP-Protokoll. Die entsprechende klientenseitige Java-API zur Steuerung des Spracheingabekanals ist in Listing 9.2 aufgeführt.

(12) Die Sprachdaten werden mit Hilfe des Realtime Transport Protocols (RTP) (Schulzrinne u. a. 2003) auf Basis von UDP übermittelt. Diese können entweder, wie abgebildet, direkt zwischen Klient und Sprachkanal ausgetauscht oder aber durch die Dialogplattform geschleust werden.

Insgesamt wurde SiAM-dp um Bibliotheken und Dienste für die Protokolle SIP, SDP, MRCP und RTP ergänzt. Diese werden vom SipConnector und den vorhandenen Sitzungen verwendet. Aufgrund der eingesetzten Standards kann nun ein Agent eine Verbindung zur Dialogplattform sogar per (Mobil-) Telefon aufbauen. Umgekehrt ist dies ebenso möglich. Diese Fähigkeit wurde im Rahmen des Projekts DIASIM[6] eingesetzt. Der SipConnector erbt außerdem von der SiAM-Basisklasse AbstractDeviceComponent (Neßelrath 2016, S. 180ff.). Dadurch wird eine nahtlose Integration der verbundenen Geräte erreicht, u. a. durch Zugriff auf den SiAM-internen Mechanismus zur Ereignisbehandlung.

Listing 9.2: MRCP-Contoller zur Steuerung des Spracheingabekanals

```
1 package de.dfki.iui.kombinos.runtime.mrcp.controller;
2
3 import java.util.Collection;
4 import de.dfki.iui.kombinos.runtime.mrcp.delegate.RecognitionDelegate;
5 import de.dfki.iui.kombinos.runtime.mrcp.resource.RecognitionResource;
6
```

[6] Ziel des Projekts war die Implementierung eines Dialogsystems zur interaktiven und realitätsnahen Schulung von Callcenter-Mitarbeitern eines Schweizer Telekommunikationsanbieters. Diese konnten so in ihrer gewohnten Arbeitsumgebung oder von unterwegs einen simulierten Kundendialog zur Bestellung eines Informationsprospekts führen. Ein virtueller Lehrer gab währenddessen und anschließend Tipps zur Verbesserung der Kommunikation.

```
7  public interface RecognitionController extends MediaController<
       RecognitionResource, RecognitionDelegate> {
8    public void startRecognition();
9    public void startHotwordRecognition(int minDuration, int maxDuration);
10   public void cancelRecognition();
11   public void interpretText(String text);
12   public String getContext();
13   public void setContext(String context);
14   public Collection<Grammar> getGrammars();
15   public Collection<Grammar> getGrammars(String context);
16   public void addGrammar(Grammar grammar);
17   public Grammar getGrammar(String id);
18   public void removeGrammar(String id);
19 }
```

Komponenten der KomBInoS-Dialogplattform

Sowohl die zuvor beschriebene Camunda-Integration als auch der SipConnector weisen naturgemäß eine starke Kopplung zu jeweils einer konkreten Technologie auf. Für die multimodale Dialoginteraktion mit Prozessen wurden im Kern der KomBInoS-Dialogplattform aufeinander abgestimmte Komponentenschnittstellen für u. a. die Aufgaben- und Dialogverwaltung im Kontext der Interaktion mit Smart Services technologieunabhängig konzipiert und prototypisch umgesetzt. Dadurch kann sowohl die Anbindung an eine spezifische Prozessausführungsmaschine im Back-End als auch die Umsetzung der Klientenanbindung im Front-End schnell und ohne Auswirkungen auf die interne Verarbeitungslogik ausgetauscht oder angepasst werden. Die entsprechenden laufzeitrelevanten Schnittstellen sind in Abbildung 9.6 dargestellt.

Die Konzepte Application und Task entstammen der KomBInoS-Metamodell-Architektur (Kapitel 6, S. 111). Auf der linken Seite sind Komponenten platziert, die als Singleton genau einmal in der KomBInoS-Plattform instanziiert und über die OSGi-Dienstregistratur verfügbar sind. Eine Sitzung sowie ein Dialogmanager werden hingegen pro Agent instanziiert.

- Die **Anwendungsverwaltung** (ApplicationManager) dient zur Inbetriebnahme von KomBInoS-Anwendungen in der Plattform sowie zum Starten und Abfragen von (laufenden) Instanzen (ApplicationInstance). Dies geschieht ebenfalls unter Zuhilfenahme der Camunda-Integration, welche in diesem Zuge auch einen zugeordneten ausführbaren Prozess auf der Camunda BPM-Plattform instanziieren muss.
- Der **Aufgabenverwaltung** (TaskManager) fällt der größte Koordinationsaufwand zu. Sie reagiert einerseits auf Ereignisse des Camunda Back-Ends mittels der dafür zuständigen Ereignisbehandlungsmethoden (z. B. *onTaskAssigned*). Andererseits kann die Aufgabenverwaltung steuernd in einen laufenden Prozess eingreifen und in Bearbeitung befindliche Aufgaben (TaskInstance) abschließen oder an einen anderen Agenten delegieren. Hierbei kommen wiederum die Camunda-Integration, aber auch die Agenten- und Sitzungsverwaltung zum Einsatz. Listing 9.3 zeigt die

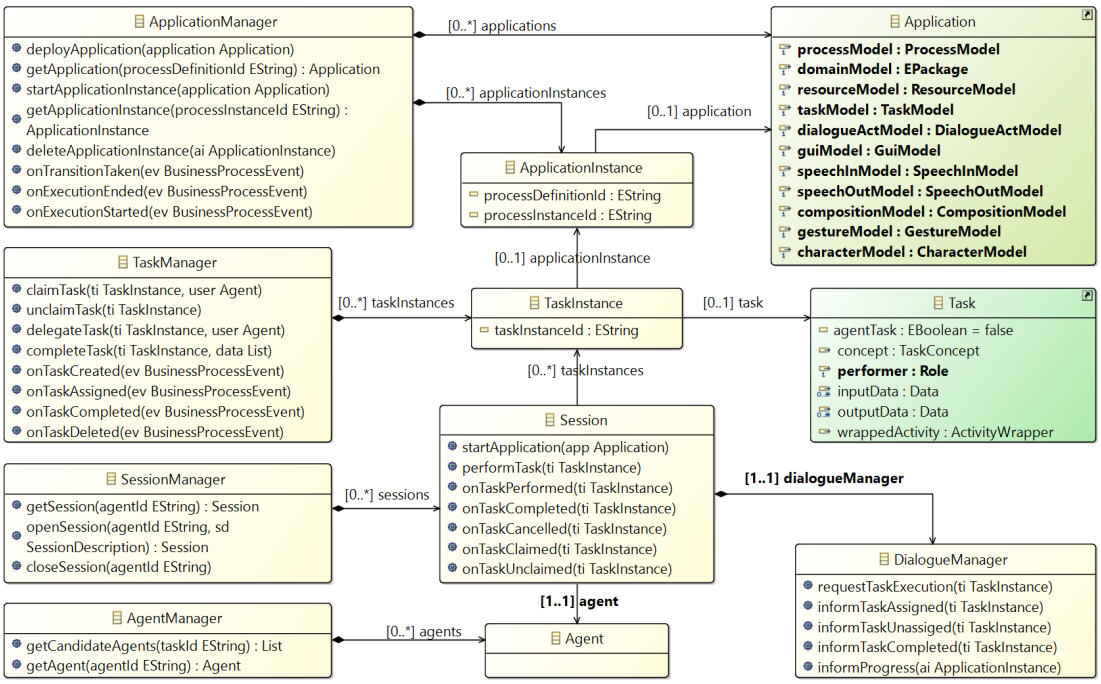


Abbildung 9.6: Schnittstellen der KomBInoS-Laufzeitkomponenten

OSGi-Komponentenbeschreibung der Aufgabenverwaltung.

- Die **Agentenverwaltung** (AgentManager) bietet unter Zuhilfenahme der Camunda-Integration Zugriff auf die Stammdaten registrierter Agenten innerhalb der Camunda BPM-Plattform. Die Daten werden in der KomBInoS-eigenen Agentenmodellierung (Agent aus Kapitel 6.4 (S. 118)) bereitgestellt. Die Agentenverwaltung kann außerdem geeignete Kandidaten zur Durchführung einer Aufgabe auf Basis eines Rollen- und Fähigkeitenabgleichs vorschlagen, die bei einer späteren Zuweisung durch die Aufgabenverwaltung berücksichtigt werden.
- Die **Sitzungsverwaltung** (SessionManager) verwaltet Sitzungen aktiver Agenten. Unter Angabe einer Agenten-Id kann die Komponente Sitzungen erstellen sowie vorhandene Sitzungen angemeldeter Agenten zurückliefern oder schließen.
- Eine **Sitzung** (Session) bildet aus Sicht von SiAM-dp eine vollständig Dialogverarbeitungskette mit Komponenten für die Eingabeinterpretation, die Modalitätenfusion, das Dialogmanagement sowie der Präsentationsplanung und der Modalitätenfission ab. Aus Sicht von KomBInoS kapselt eine solche Sitzung außerdem agentenspezifische Methoden zur Durchführung und Statusüberwachung von Aufgaben. Diese werden sowohl von der Aufgabenverwaltung als auch vom Dialogmanager der Sitzung nach erfolgter Interaktion aufgerufen.
- Ein **Dialogmanager** (DialogueManager) tritt mit dem ihm zugeordneten Agenten in Interaktion, um etwa die Durchführung einer Aufgabe anzustoßen und de-

ren Bearbeitung situationsgerecht voranzutreiben und zu begleiten. Während alle anderen Komponenten der Dialogverarbeitungskette von SiAM-dp aufgrund ihrer Generizität unverändert eingesetzt werden können, ist für den Dialogmanager eine zum geschilderten Zweck definierte API vonnöten. Eine geeignete Dialogstrategie erlaubt, wie in Kapitel 7.3.2 (S. 151) ausgeführt, eine adaptive Dialogführung gemäß Zanten (1998). Der Zugriff auf einen umfänglichen Diskurskontext ermöglicht zudem das Zwischenspeichern von Information, etwa bei Überbeantwortung von Systemfragen durch den Agenten, bis zu einem späteren Zeitpunkt, an dem die Information (gegebenenfalls wiederholt) benötigt wird. Ebenso informiert er den Agenten über eingetretene Ereignisse bzgl. des Prozessfortschritts. Hierzu hat der Dialogmanager Zugriff auf die während der Entwicklung erzeugten Dialogakte. Außerdem kann er Methoden der Anwendungsverwaltung, der Aufgabenverwaltung und der Sitzung aufrufen. Diese Komponenten stellen also einen Teil des domänenspezifischen Back-Ends eines Dialogsystems (Kapitel 4.3.2, S. 60) dar.

Listing 9.3: OSGi-Komponentenbeschreibung der KomBInoS-Aufgabenverwaltung

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="de.
      dfki.iui.kombinos.runtime.TaskManager">
3   <implementation class="de.dfki.iui.kombinos.runtime.impl.
        TaskManagerImpl"/>
4   <service>
5     <provide interface="org.osgi.service.event.EventHandler"/>
6     <provide interface="de.dfki.iui.kombinos.runtime.TaskManager"/>
7   </service>
8   <reference name="CamundaIntegration" interface="de.dfki.iui.kombinos.
        camunda.CamundaService" cardinality="1..1" policy="static"/>
9   <reference name="AgentManager" interface="de.dfki.iui.kombinos.runtime.
        AgentManager" cardinality="1..1" policy="static" />
10  <reference name="SessionManager" interface="de.dfki.iui.kombinos.
        runtime.SessionManager" cardinality="1..1" policy="static" />
11  <property name="event.topics" type="String" value="de/dfki/iui/kombinos
        /camunda/*"/>
12 </scr:component>
```

In Abbildung 9.7 ist die Interaktion beteiligter Komponenten bei der Durchführung einer an einen Agenten zugewiesenen Aufgabe beispielhaft als Sequenzdiagramm aufgeschlüsselt.

Die Aufgabenverwaltung bezieht über die Sitzungsverwaltung die aktuelle Sitzung des Agenten, der für die Aufgabe vorgesehen ist und beauftragt diesen über seine Sitzung mit der Durchführung. Hierzu greift die Sitzung auf den ihr zugeordneten Dialogmanager zurück, welcher schließlich mit dem Agenten in Interaktion tritt.

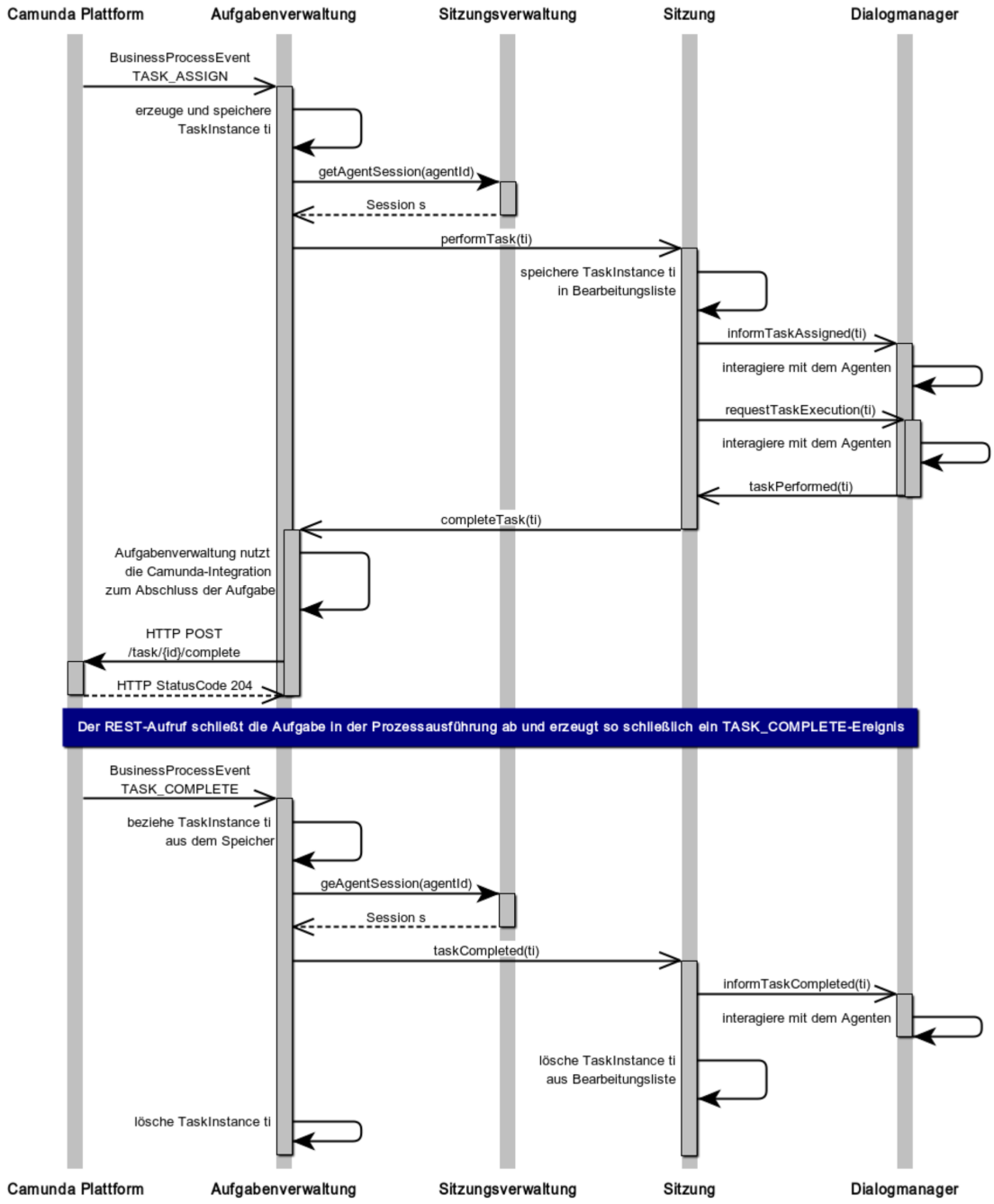


Abbildung 9.7: Sequenzdiagramm der beteiligten Laufzeitkomponenten bei der Durchführung einer Aufgabe durch einen Agenten

Die KomBInoS-Webkonsole zum Fernverwaltung der Dialogplattform

Die KomBInoS-Webconsole ist in Abbildung 9.8 dargestellt. Sie basiert auf der Apache Felix Webconsole[7] und dient zum Verwalten der Dialogplattform via Webbrowser. Ein Administrator kann sich über den Status sämtlicher Bündel der zugrundeliegenden OSGi-Umgebung informieren, sie (neu-) starten oder stoppen. Er kann außerdem neue KomBInoS-UI- oder allgemein OSGi-Bündel installieren, Dienste und Komponenten zur Laufzeit umkonfigurieren und Logdateien einsehen.

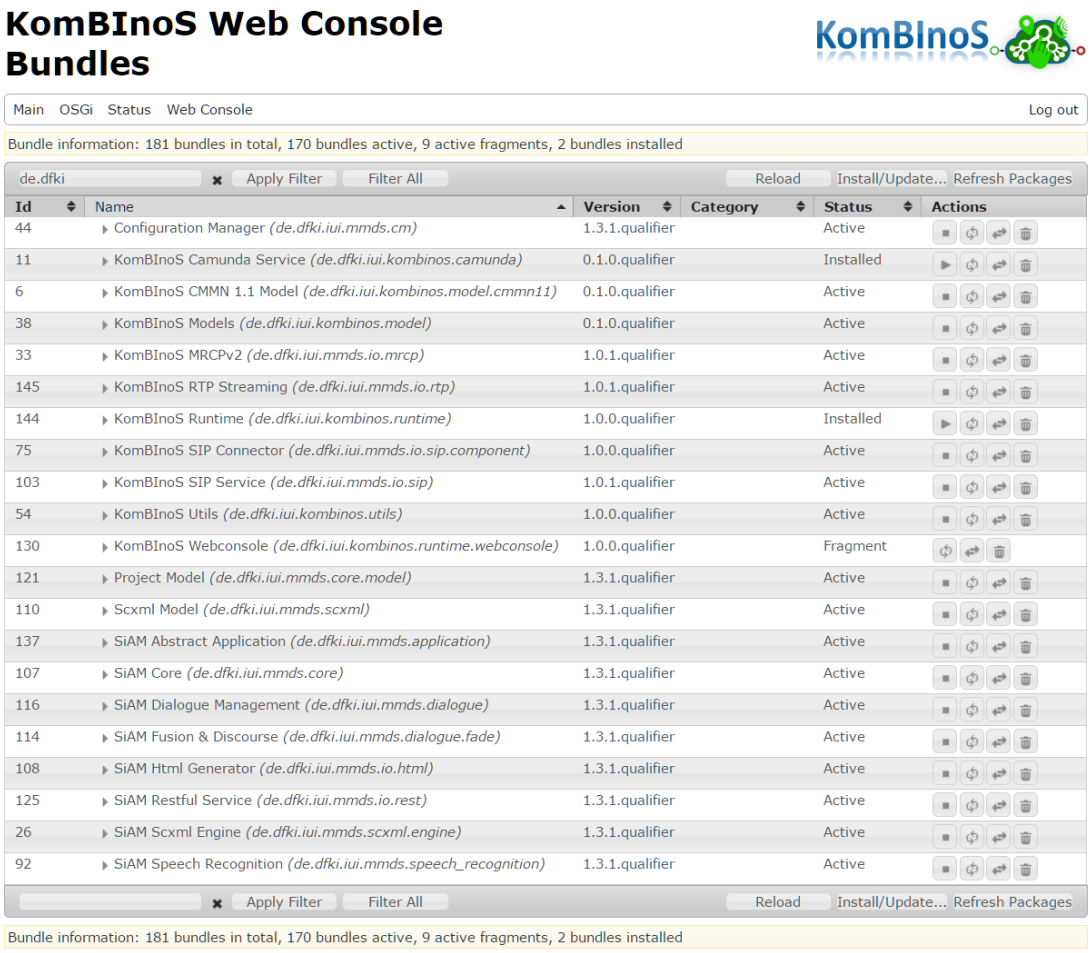


Abbildung 9.8: KomBInoS-Webkonsole zur entfernten Verwaltung der Dialogplattform

[7] siehe http://felix.apache.org/documentation/subprojects/apache-felix-web-console.html (letzter Zugriff: 04.11.2017)

9.5 Der KomBInoS-Verzeichnisdienst

Der KomBInoS-Verzeichnisdienst stellt die notwendige Infrastruktur für die SIP-basierte Kommunikation zwischen Dialogplattform und einer beliebigen Anzahl an Dialogklienten in Form eines leichtgewichtigen Registrars zur Verfügung. Der netzwerktechnische Endpunkt des Registrars in Form einer IP-Adresse und einem Port ist allgemein bekannt. Dialogplattform und –klienten werden in der Form gleich behandelt, als dass sie sich alle beim Registrar mit ihrer Agenten-Id und dem aktuell gültigen Endpunkt registrieren. Die Agenten-Id der Dialogplattform lautet *kombinos*. Listing 9.4 zeigt einen Schnappschuss der Registrierungsdatenbank des KomBInoS-Verzeichnisdiensts in JSON.

Listing 9.4: Registrierungsdatenbank des KomBInoS-Verzeichnisdienstes

```
1  {
2      "kombinos": [
3          {
4              "name": "'KomBInoS'",
5              "uri": "sip:kombinos@127.0.0.1:60923;transport=udp"
6          }
7      ],
8      "alice": [
9          {
10             "name": "'Alice'",
11             "uri": "sip:alice@192.168.11.20:48028;transport=udp"
12         }
13     ],
14     "bob": [
15         {
16             "name": "'Bob'",
17             "uri": "sip:bob@192.168.11.21:42245;transport=udp"
18         }
19     ]
20 }
```

Eine Registrierung wird periodisch gegebenenfalls mit einem neuen Endpunkt erneuert. So kann vom netzwerktechnischen Endpunkt abstrahiert werden, was u. a. den Konfigurationsaufwand reduziert und gleichzeitig die Identifizierung eines Kommunikationspartners erleichtert. Dies ist von Vorteil, gerade wenn mehrere Klienten in unterschiedlichen Rollen über die Dialogplattform mit einem Prozess interagieren. Der KomBInoS-Verzeichnisdienst bietet bei entsprechender Konfiguration der Teilnehmer außerdem die Funktionalität eines SIP-OutboundProxies. Hierbei werden optional alle SIP-Nachrichten effizient durch den Verzeichnisdienst geschleust. Dadurch kann der Nachrichtenfluss in chronologischer Reihenfolge aufgelistet werden, was u. a. die Fehlerbehebung bei der Entwicklung vereinfacht.

Die Implementierung erfolgte auf Basis von NodeJS[8]. NodeJS ist eine stabile und per-

[8] siehe https://nodejs.org (letzter Zugriff: 04.11.2017)

formante Serverplattform, die über eine reichhaltige API Betriebssystemfunktionalität sowie Fähigkeiten zur Netzwerkkommunikation in JavaScript zur Verfügung stellt. So sind entsprechende Module für die standardkonforme SIP-Kommunikation einschließlich einer Registrar-Funktionalität für NodeJS verfügbar und aufgrund der Implementierung in JavaScript leicht anpassbar. Neben der geschilderten Telefonbuch- und Vermittlungsfunktion bietet der Verzeichnisdienst außerdem schnelle Integrationsmöglichkeiten zum Einbinden externer Dienste, die nicht zwangsläufig dialog- bzw. prozessrelevant sind. Diese Fähigkeit wurde etwa im Rahmen des LWP Demonstrators (Kapitel 10.5, S. 241) zur Abfrage von Wetter- und Fahrzeuginformation genutzt.

9.6 Der mobile KomBInoS-Klient

Der mobile KomBInoS-Klient stellt das Interaktionsgerät eines Benutzers dar. Ein Entwurfsziel des mobilen KomBInoS-Klienten besteht in einer möglichst große Plattformunabhängigkeit sowie einer möglichst leichtgewichtigen mobilen Anwendung, um eine einfache Übertragbarkeit auf eine große Anzahl mobiler Plattformen zu ermöglichen. Zu diesem Zweck nutzt die Implementierung Apache Cordova[9], einem quelloffenen Rahmenwerk zur plattformübergreifenden Entwicklung browserbasierter mobiler Anwendungen. Hierbei kommen Standard-Web-Technologien wie HTML5, kaskadierende Stylesheets in Version 3 und JavaScript für die Cross-Plattform-Entwicklung zum Einsatz. Der plattformabhängige Zugriff auf geräte- oder plattformspezifische Sensoren, Dienste oder das Dateisystem wird von Cordova plattformunabhängig auf eine einheitliche JavaScript-API abgebildet. Dazu existiert eine Vielzahl an Plugins, die je nach Bedarf in eine App integriert werden können. Durch die Implementierung eines eigenen Cordova-Plugins für die KomBInoS- bzw. die im Kern enthaltene SiAM-Dialogplattform kann diese leicht über JavaScript angesteuert werden, z. B. zum Öffnen des Sprachkanals. Umgekehrt kann die lokale Sprachgenerierung und -ausgabe auf dem Gerät von der Dialogplattform aus angestoßen werden. Listing 9.5 zeigt die entsprechende Definition des Cordova-Plugins. Die JavaScript-API ist im Browser unter dem Namen kombinos registriert und verwendbar (Zeilen 17-19) und wird für die Android-Plattform auf eine Java-Klasse abgebildet (Zeilen 20-27). Eine Unterstützung weiterer mobiler Plattformen, z. B. iOS, ist leicht möglich. Hierzu müssen lediglich native Codeanteile zur Sprachübertragung oder -erzeugung auf dem Endgerät in die mobile Zielplattform überführt werden. Entsprechende Bibliotheken wurden bereits für frühe Versionen von Apache Cordova und für iOS entwickelt (Sonntag u. a. 2010a).

Listing 9.5: Definition des Cordova-Plugins

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <plugin xmlns="http://apache.org/cordova/ns/plugins/1.0"
3   xmlns:android="http://schemas.android.com/apk/res/android"
4   id="de.dfki.iui.kombinos.client.plugins.SiamConnector"
```

[9] siehe https://cordova.apache.org/ (letzter Zugriff: 04.11.2017)

```
5    version="1.0.0">
6
7    <name> KombinosConnector </name>
8    <keywords> kombinos, siam </keywords>
9    <description> Cordova plugin for connecting to the KomBInoS Dialogue
         Platform. </description>
10   <license> individual </license>
11   <author> Daniel Porta </author>
12   <engines>
13       <engine name="cordova" version=">=3.0" />
14   </engines>
15   <js-module src="www/cordova-kombinos.js" name="kombinos">
16     <clobbers target="window.kombinos" />
17   </js-module>
18
19   <!-- android -->
20   <platform name="android">
21       <config-file target="res/xml/config.xml" parent="/*">
22           <feature name="KombinosConnector">
23               <param name="android-package" value="de.dfki.iui.kombinos.
                     client.plugins.KombinosConnector" />
24           </feature>
25       </config-file>
26       <source-file src="src/android/KombinosConnector.java" target-dir="
             src/de/dfki/iui/kombinos/client/plugins" />
27   </platform>
28 </plugin>
```

Listing 9.6 gibt die JavaScript-API um Fehlerbehandlung verkürzt wieder.

Listing 9.6: JavaScript-API zur Kommunikation mit der Dialogplattform

```
1  var exec = require('cordova/exec');
2
3  var SiamConnector = function() {
4    var self = this;
5    var initialized = false;
6  }
7
8  SiamConnector.prototype.startup = function(args) {
9    exec(function() { self.initialized = true;},
10        function(e) {  /*handle initialization failed*/ },
11       "SiamConnector", "startup", args);
12 };
13
14 SiamConnector.prototype.invite = function() {
15   exec(null, null, "SiamConnector", "invite", []);
16 };
17
18 SiamConnector.prototype.bye = function() { ... };
19 SiamConnector.prototype.mute = function() { ... };
20 SiamConnector.prototype.unmute = function() { ... };
21
```

```
22 SiamConnector.prototype.sendMessage = function(receiver,message) {
23   exec(null, null, "SiamConnector", "message", [receiver, message]);
24 };
25
26 SiamConnector.prototype.sendInfo = function(message) {
27   exec(null, null, "SiamConnector", "info", [message]);
28 };
29
30 SiamConnector.prototype.onCallEstablished = function(cb) {
31   exec(cb, null, "SiamConnector", "oncallestablished", []);
32 };
33
34 SiamConnector.prototype.onCallEnded = function(cb) { ...};
35 SiamConnector.prototype.onError = function(cb) { ...};
36 SiamConnector.prototype.onMessage = function(cb) { ... };
37 SiamConnector.prototype.onInfo = function(cb) { ... };
38
39 module.exports = new SiamConnector();
```

Die tatsächliche Kommunikation erfolgt dabei wie bereits geschildert über SIP. Dadurch können Unterbrechungen in langlaufenden Interaktionsprozessen bei Prozessfortschritt durch Initiative des Dialogsystems wieder aufgenommen werden. Zur Unterstützung von Instant-Messaging-Funktionalität wurde der Android-eigene SIP-Service im Quellcode[10] um die fehlenden Methoden INFO und MESSAGE standardkonform gemäß Campbell u. a. (2002) ergänzt und in den KomBInoS-Klienten integriert, wie das AndroidManifest des Clients in Listing 9.7 (Zeile 35) zeigt. Dieser so erweiterte Android-SIP-Service erneuert im Hintergrund regelmäßig die SIP-Registrierung im KomBInoS-Verzeichnisdienst, ohne dass die App gestartet sein muss. Er liefert außerdem die Basis für den entsprechenden OSGi-Service in der KomBInoS-Dialogplattform. Hierzu mussten lediglich der intern verwendete, Android-spezifische Mechanismus zum Senden und Behandelns von Ereignissen OSGI-konform angepasst sowie alle Android-spezifischen API-Aufrufe, hauptsächlich zum Stromsparen und Überprüfen von Berechtigungen, entfernt werden. Bedenkt man, dass diese SIP-Bibliothek auf einer Vielzahl von Android-Geräten vorhanden ist und genutzt wird, so kann die SIP-Kommunikation zwischen KomBInoS-Plattform und Klienten als sehr robust erachtet werden.

Listing 9.7: AndroidManifest des KomBInoS-Client

```
1 <?xml version='1.0' encoding='utf-8'?>
2 <manifest package="de.dfki.iui.kombinos.client">
3   <uses-permission name="android.permission.INTERNET" />
4   <uses-permission name="android.permission.ACCESS_NETWORK_STATE" />
5   <uses-permission name="android.permission.USE_SIP" />
6   <uses-permission name="android.permission.READ_EXTERNAL_STORAGE" />
7   <uses-permission name="android.permission.WRITE_EXTERNAL_STORAGE" />
8   <uses-permission name="android.permission.RECORD_AUDIO" />
```

[10] siehe https://android.googlesource.com/platform/frameworks/opt/net/voip/ (letzter Zugriff: 04.11.2017)

```
9   <uses-permission name="android.permission.VIBRATE" />
10  <uses-permission name="android.permission.MODIFY_AUDIO_SETTINGS" />
11  <uses-permission name="android.permission.BLUETOOTH" />
12  <uses-permission name="android.permission.RECEIVE_BOOT_COMPLETED" />
13  <uses-permission name="android.permission.WAKE_LOCK" />
14  <uses-permission name="android.permission.READ_PHONE_STATE" />
15  <application icon="@drawable/icon_kombinos" label="@string/app_name">
16    <activity name=".KombinosClient" label="@string/app_name"
17      theme="@style/Theme.Black.NoTitleBar" launchMode="singleTop"
18      configChanges="orientation|keyboardHidden|keyboard|screenSize" >
19      <intent-filter>
20        <action name="android.intent.action.MAIN" />
21        <category name="android.intent.category.LAUNCHER" />
22      </intent-filter>
23      <meta-data name="alias" value="@string/alias_kombinos" />
24    </activity>
25    <!-- cut activity aliases for LWP, EWULEH, etc. -->
26    <receiver name=".receivers.Receiver">
27      <intent-filter>
28        <action name="android.intent.action.BOOT_COMPLETED" />
29        <action name="de.dfki.iui.kombinos.client.SETTINGS_CHANGED" />
30        <action name="de.dfki.iui.kombinos.client.CALL_STATE_CHANGED" />
31        <action name="de.dfki.iui.kombinos.client.PUSH_TO_TALK" />
32        <category name="android.intent.category.DEFAULT" />
33      </intent-filter>
34    </receiver>
35    <service name="de.dfki.iui.kombinos.client.sip.SipService" />
36  </application>
37 </manifest>
```

Um das spezifische Aussehen und die Handhabung unterschiedlicher mobiler Zielplattformen in HTML5 nachzubilden, wird das HTML5-Rahmenwerk KendoUI Mobile eingesetzt, welches inzwischen Bestandteil der quelloffenen Core-Distribution[11] des Herstellers Telerik ist. Im Vergleich zu anderen Rahmenwerken, z. B. Sencha ExtJS[12] oder jQueryMobile[13], erlaubt KendoUI die vollständig deklarative Beschreibung einer HTML5-Anwendung. Anwendungen auf Basis von KendoUI Mobile sind Single-Page-Webanwendungen, die sämtlichen HTML-Code in einer initial geladenen HTML-Datei enthalten. Nach dem Start der Anwendung müssen lediglich Anwendungsdaten zwischen Anwendung und Webserver bzw. Webdienst ausgetauscht werden. Dazu wird als MVVM-Rahmenwerk zur bidirektionalen Datenbindung zwischen Präsentation und Präsentationsmodell bewusst KnockoutJS[14] als leichtgewichtigere Alternative zum populären AngularJS-Rahmenwerk[15] eingesetzt. Dennoch können über sogenannte Re-

[11] siehe http://www.telerik.com/kendo-ui/open-source-core (letzter Zugriff: 04.11.2017)
[12] siehe https://www.sencha.com/products/extjs/ (letzter Zugriff: 04.11.2017)
[13] siehe https://jquerymobile.com/ (letzter Zugriff: 04.11.2017)
[14] siehe http://knockoutjs.com/ (letzter Zugriff: 04.11.2017)
[15] siehe http://angularjs.org/ (letzter Zugriff: 04.11.2017)

mote Views[16] HTML-Inhalte nachgeladen werden. Das Erscheinungsbild der mobilen Anwendung kann durch kaskadierende Stylesheets angepasst werden. Außerdem wird es durch den Einsatz des Internationalisierungs-Rahmenwerks i18next[17] möglich, die GUI in unterschiedlichen Sprachen zu lokalisieren. Insgesamt definieren diese Softwarebibliotheken das plattformspezifische Modell als Ziel der M2C-Transformation zur Generierung der grafischen UI. Anwendungsspezifische Ausprägungen der mobilen GUI werden im nächsten Kapitel vorgestellt.

9.7 Fazit

Die vorgestellte KomBInoS-Laufzeitumgebung zur mobilen multimodalen dialogischen Interaktion mit Smart Services implementiert eine offene Architektur unter Ausnutzung adäquater Kommunikationsstandards und -protokolle. Dadurch wird eine lose Kopplung der funktionalen Komponenten etabliert, welche eine Verteilung dieser in einer Cloud-Infrastruktur ermöglicht. Die Camunda BPM-Plattform im Back-End fungiert als Prozessausführungsmaschine und wurde um die Fähigkeit erweitert, prozessbezogene Ereignisse während der Ausführung per MQTT zu publizieren. Außerdem wurde die SiAM-Dialogplattform um funktionale Komponenten und Dienste zur Prozessinteraktion im Back-End via MQTT und REST ergänzt. Die prototypische Implementierung des Dialogmanagements wurde unter Einsatz des Statechart-basierten SiAM-Dialogmodells (Neßelrath 2016, S. 123ff.) zwar möglichst generisch umgesetzt. Der Einsatz flexiblerer Verfahren, wie in Kapitel 4.3.4 (S. 63) geschildert, kann sich jedoch vorteilhaft auf das Benutzererlebnis auswirken. Die Implementierung eines solchen Verfahrens wird im Ausblick (Kapitel 11.3, S. 264) wieder aufgegriffen. Aufgrund der wohldefinierten Schnittstellen der KomBInoS-Plattformkomponenten, wie in Abbildung 9.6 dargelegt, ist der Austausch einer konkreten Implementierung, etwa des Dialogmanagers, leicht möglich, nachdem die SiAM-Dialogplattform um komplexere Mechanismen zum Dialogmanagement ergänzt wurde. Benutzer können über leichtgewichtige mobile Klienten zu jeder Zeit mit für sie relevanten Prozessen in Kontakt treten, ebenso können sie über neue Entwicklungen in Kenntnis gesetzt werden. Hierzu erfolgt die Kommunikation zwischen KomBInoS-Dialogplattform und mobilem Klienten auf Basis von SIP und unter Zuhilfenahme eines zentralen Verzeichnisdienstes. Auf diese Weise kann auch die Dialogplattform einen Sitzungsaufbau zu einem bestimmten Agenten initiieren. Die KomBInoS-Laufzeitumgebung wurde wie auch die Werkbank mit Hilfe eines modellgetriebenen Ansatzes umgesetzt. Quantitative Kennzahlen zur Laufzeitumgebung befinden sich in Anhang A (S. 269). Insgesamt wird somit die in Kapitel 1.2.3 (S. 4) formulierte ingenieurtechnische Fragestellung (5) „*Wie kann man Smart Services im multimodalen Dialog auf mobilen Endgeräten nutzen?*“ umfassend beantwortet.

[16] siehe http://docs.telerik.com/kendo-ui/controls/hybrid/application#remote-views (letzter Zugriff: 04.11.2017)

[17] siehe http://i18next.com/ (letzter Zugriff: 04.11.2017)

KAPITEL 10

Anwendungen

Anhand von acht Forschungsprototypen, die im Rahmen der Arbeit in fünf Projekten entstanden sind, wird in diesem Kapitel die Einsatzfähigkeit der KomBInoS-Methodik in der Praxis demonstriert.

10.1 Einleitung

Die Evaluation einer integrierten Entwicklungs- und Kompositionsmethode für multimodale Dialogschnittstellen für Smart Services ist stark abhängig von einem technischen Rahmenwerk, welches die Methode in Software implementiert. Vor diesem Hintergrund werden in diesem Kapitel acht Anwendungen vorgestellt, die auf Basis der KomBInoS-Methodik in Teilen oder als Ganzes erstellt wurden. Im Fazit werden diese Anwendungen in Bezug auf ihre KomBInoS-Nutzung gegenübergestellt, sodass das Delta der Nutzung ersichtlich wird.

10.2 Mobiler TEXO-Client

Im Kontext des BMWi-geförderten THESEUS-Anwendungsfalls TEXO (Förderkennzeichen: 01MQ07012) realisiert der mobile TEXO-Client (Porta u. a. 2009a) eine mobile Anbindung an ein Enterprise Resource Planning-System als integriertes Unternehmensplanungs- und Steuerungssystem im Internet der Dienste. Die Anwendung deckt auf unternehmerischer Ebene die Geschäftsprozesse Einkauf von digitalen Diensten und Controlling ab. Im Dienstlebenszyklus (Kapitel 3.3, S. 33) werden die Phasen Dienstvermittlung und Dienstnutzung adressiert. Diese unterschiedlichen Facetten spiegeln sich im Layout der GUI wider und können vom Benutzer auf einem mobilen Endgerät im multimodalen Dialog bedient werden. Im unteren Teil der GUI befinden sich entsprechende Reiter zum Umschalten zwischen der unternehmensinternen Beschaffungsantragsliste

als Einstiegspunkt in die von einer Diensteplattform bereitgestellten Benutzerschnittstellen zur Dienstsuche und Dienstnutzung. Die mobile Anwendung unterstützt neben Sprache und Touch auch Gesten unter Zuhilfenahme eines anlernbaren Gestenerkenners als zusätzliche Eingabemodalität (Porta u. a. 2009b), etwa zur Implementierung einer Zurück- bzw. Verwerfen-Funktionalität durch Schütteln. Nach dem Einkauf und der Bereitstellung des Dienstes kann dieser unternehmensweit aus der mobilen Anwendung heraus genutzt werden.

Die Benutzerschnittstelle zur Dienstnutzung ist in Abbildung 10.1 dargestellt. Sie unterteilt sich (1) in eine Ansicht zur Visualisierung der mobilen Dienstnutzungsschnittstelle, (2) eine Ansicht zur Inspektion von Leistungsdaten, die während der Nutzung von der ausführenden Diensteplattform kontinuierlich aufgezeichnet werden und (nicht dargestellt) eine Ansicht zur Anzeige und zum Abgeben von Nutzerbewertungen.

Anwendung der Methodik

Die mobile UI zur Dienstvermittlung ist noch nativ in iOS implementiert. Der Anteil zur mobilen Dienstnutzung und -überwachung greift jedoch bereits auf einen in die native Anwendung eingebetteten Webbrowser zurück. Hierzu wurde die serverseitige Dialogplattform ODP (Löckelt u. a. 2014) technisch um die On-the-fly-Generierung von finalem HTML-Code auf Basis von deklarativen Beschreibungen grafischer Dienst-UIs erweitert. Dadurch können finale UIs, wie in Abbildung 10.1 (3) gezeigt, anhand eines modellgetriebenen Ansatzes über das Dialogsystem kontextbewusst generiert und präsentiert werden.

10.3 Mobiler Reiseführer

Der mobile Reiseführer (Sonntag u. a. 2010a) ermöglicht einem Reisenden die Umgebungssuche nach interessanten Orten. Die Anwendung setzt Ideen aus dem Bereich der lokationsbasierten Dienste, z. B. von Porta und Conrad (2008) und Porta (2008), prototypisch auf der vom TEXO-Client bereitgestellten Dialog-Infrastruktur neu um. Der folgende Dialog zwischen einem Benutzer und dem mobilen Reiseführer stellt eine typische Interaktionssequenz dar. Alle Sprachinteraktionen können auch durch Touch-Gesten auf der grafischen Benutzerschnittstelle durchgeführt werden.

(1) BENUTZER: Wo gibt es hier gute Restaurants? [*Die Position des Benutzers wird über die Lokalisierung des mobilen Endgeräts bestimmt.*].

(2) SYSTEM: Ich zeige Restaurants in der Nähe.

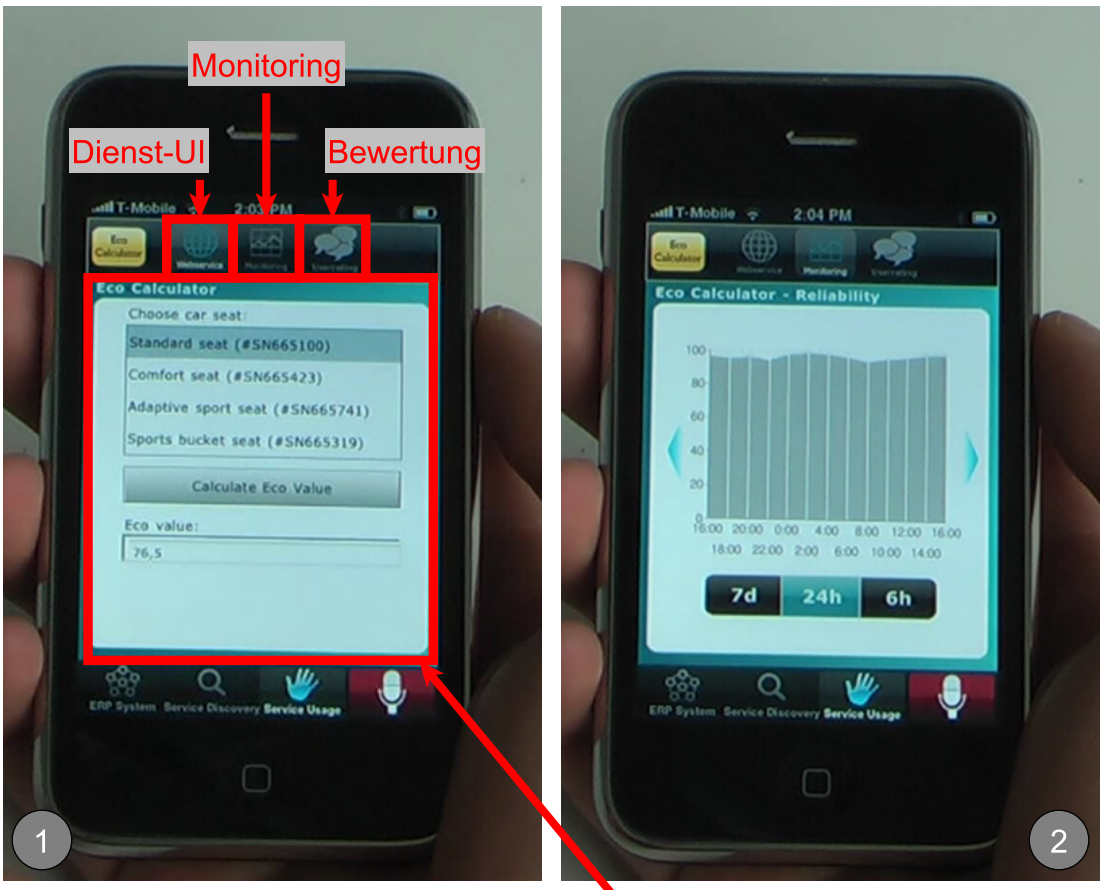
(3) BENUTZER: Gebe mir mehr Information zum zweiten Eintrag.

(4) SYSTEM: Details zum Restaurant „Fernsehturm“ werden angezeigt.

(5) BENUTZER: Was kann ich hier sonst noch besuchen?

(6) SYSTEM: Ich zeige interessante Orte in der Nähe.

(7) BENUTZER: Was kannst Du mir hierzu [↗] sagen? [*Benutzer berührt Listen-*




```
1  <canvas>
2      <window x="5" y="3" width="230" height="300" title="Eco Calculator">
3          <view>
4              <simplelayout spacing="10" axis="y"/>
5              <text text="Choose car seat:"/>
6              <textlistitem id="wsdl_UI_input" width="180">
7                  <text text="Standard seat (#SN665100)"></text>
8                  <text text="Comfort seat (#SN665423)"></text>
9              </textlistitem>
10             <button> Calculate Eco Value
11                 <handler name="onclick">
12                     % Spezifikation Dienstaufruf
13                 </handler>
14             </button>
15             <text text="Eco value:"/>
16             <textedit id="wsdl_UI_output" width="180"/>
17         </view>
18     </window>
19 </canvas>
```

Deklarative Beschreibung der Dienst-UI

3

Abbildung 10.1: Mobile Dienstnutzung im TEXO-Client

eintrag zum Roten Rathaus.].

(8) SYSTEM: Das Rote Rathaus ist ein Verwaltungsgebäude an der Rathausstraße...[*Zeigt Distanz, Wetter und Wikipedia-Zusammenfassung an*].

(9) BENUTZER: Wie komme ich dorthin?

(10) SYSTEM: Zeige Route.

Abbildung 10.2 bebildert die beschriebene Interaktionssequenz.

Abbildung 10.2: Grafische Benutzeroberfläche des mobilen Reiseführers (Sonntag u. a. 2010a, S. 174)

In (1) sucht der Benutzer nach Restaurants in der Umgebung. Anschließend wird in

(2) eine entsprechende Ergebnisliste nach Bewertung sortiert angezeigt. Die gefundenen Restaurants werden außerdem im Hintergrund auf der Karte platziert. Der Benutzer lässt sich in (3) Details zum Eintrag „Fernsehturm“ anzeigen und beschließt, dort zu speisen. Er möchte das Essen mit einem anschließenden Spaziergang abschließen. Also lässt er sich in (4) interessante Orte in der Nähe des Restaurants anzeigen. Besonders interessiert ihn das Rote Rathaus. In (5) wird weiterführende Information dazu und in (6) die Route dorthin angezeigt.

Anwendung der Methodik

Um die Konsistenz und die Vollständigkeit modalitätsspezifischer UIs zu sichern, wurde beim mobilen Reiseführer mit ODP erstmals ein durchgängiges modellgetriebenes Vorgehen bei der Entwicklung ausgehend von einer Aufgabenbeschreibung angewandt. Hierzu wurde die Eclipse-basierte ODP-Werkbank (Sonntag u. a. 2009b) mit der ISE-Werkbank (Scheithauer u. a. 2009) verschmolzen und um eigene Werkzeuge zur Aufgabenmodellierung und Modelltransformation (Porta 2010) ergänzt. Der mobile Client wurde zugunsten einer vollflächigen Browser-Komponente um native UI-Anteile bereinigt, sodass die mobile GUI als Single-Page-Webanwendung realisiert werden kann. In der Konsequenz konnte die GUI vollständig deklarativ beschrieben werden, einschließlich Steuerelemente wie etwa dem Push-To-Talk-Button. Damit einher geht eine vollständige Abbildung und Synchronisierung der GUI im Anzeigekontext der Dialogplattform. Aktualisierungen des Anzeigekontextes werden in Form von JSON-Dokumenten über eine JavaScript-API an die mobile GUI weitergegeben. Insgesamt wurde die mobile Client-Architektur unter Einsatz von Apache Cordova generalisiert, sodass der mobile Reiseführer plattformübergreifend auch auf der Android-Plattform genutzt werden kann.

10.4 Multimodale dialogische Schadenmeldung und Wetterwarnung

Ebenfalls im Rahmen von TEXO entstanden zwei Demonstratoren zur multimodalen dialogischen Schadenmeldung und Wetterwarnung. Die multimodale dialogischen Schadenmeldung (Sonntag u. a. 2010b, S. 64ff) betrachtet den in Abbildung 10.3 dargestellen unternehmensübergreifenden Geschäftsprozess der Schadenregulierung. Während der Prozess alle Beteiligten koordiniert und informiert, konzentriert sich die mobile Anwendung auf die Rolle des Versicherungsnehmers. Sie ermöglicht ihm die ortsunabhängige Durchführung aller ihn betreffenden Aufgaben beginnend bei der initialen Kontaktaufnahme im Versicherungsfall über die Durchführung einer Terminvereinbarung zur Abholung eines Mietwagens bis zur Qualitätssicherung der Regulierung im Rahmen von Kundenfeedback.

Abbildung 10.4 zeigt Bildschirmabzüge der mobilen Anwendung.

In (1) initiiert der Versicherungsnehmer eine Schadenmeldung. Anstatt in (2) die

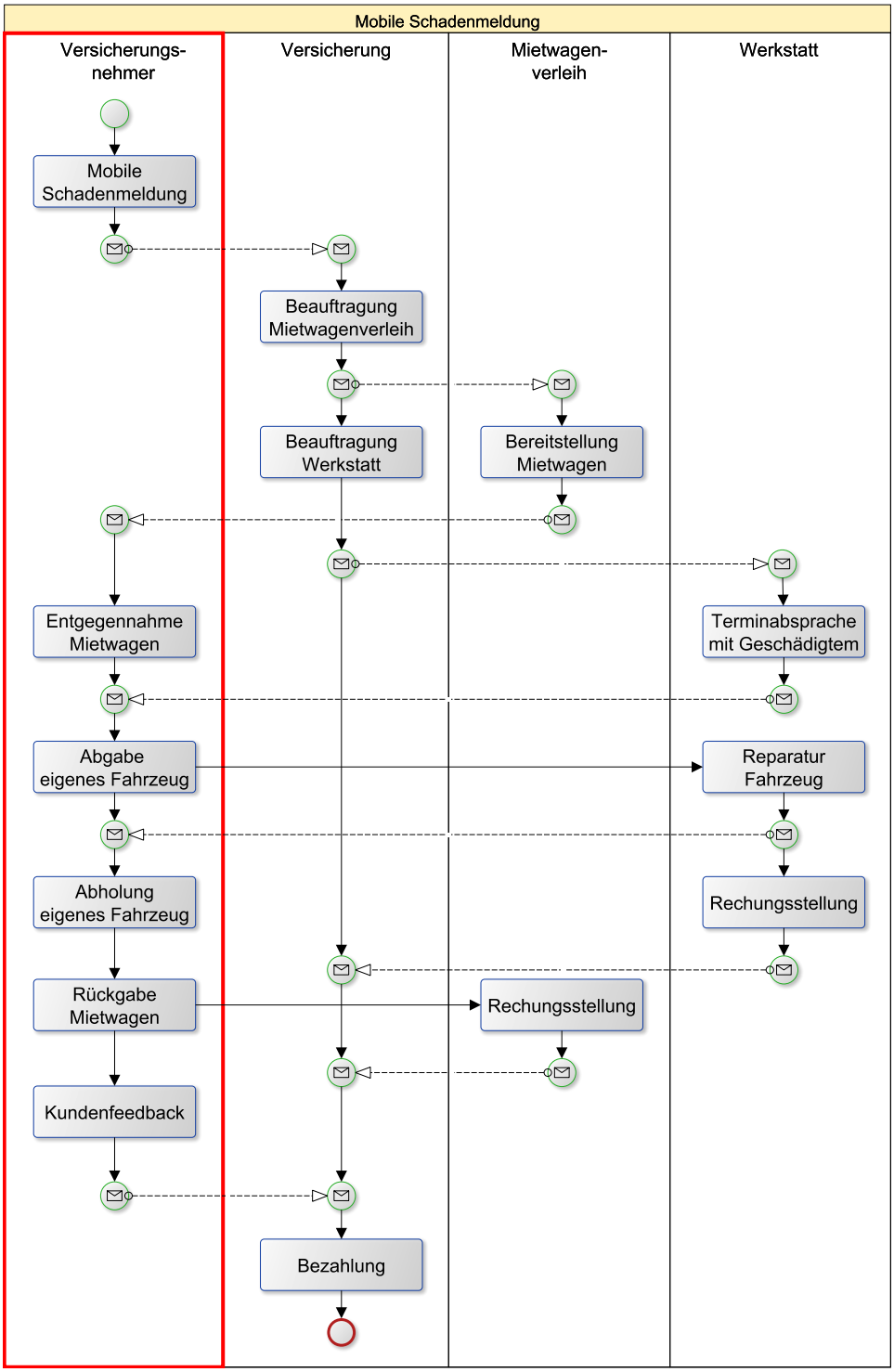


Abbildung 10.3: Prozess der Schadenmeldung

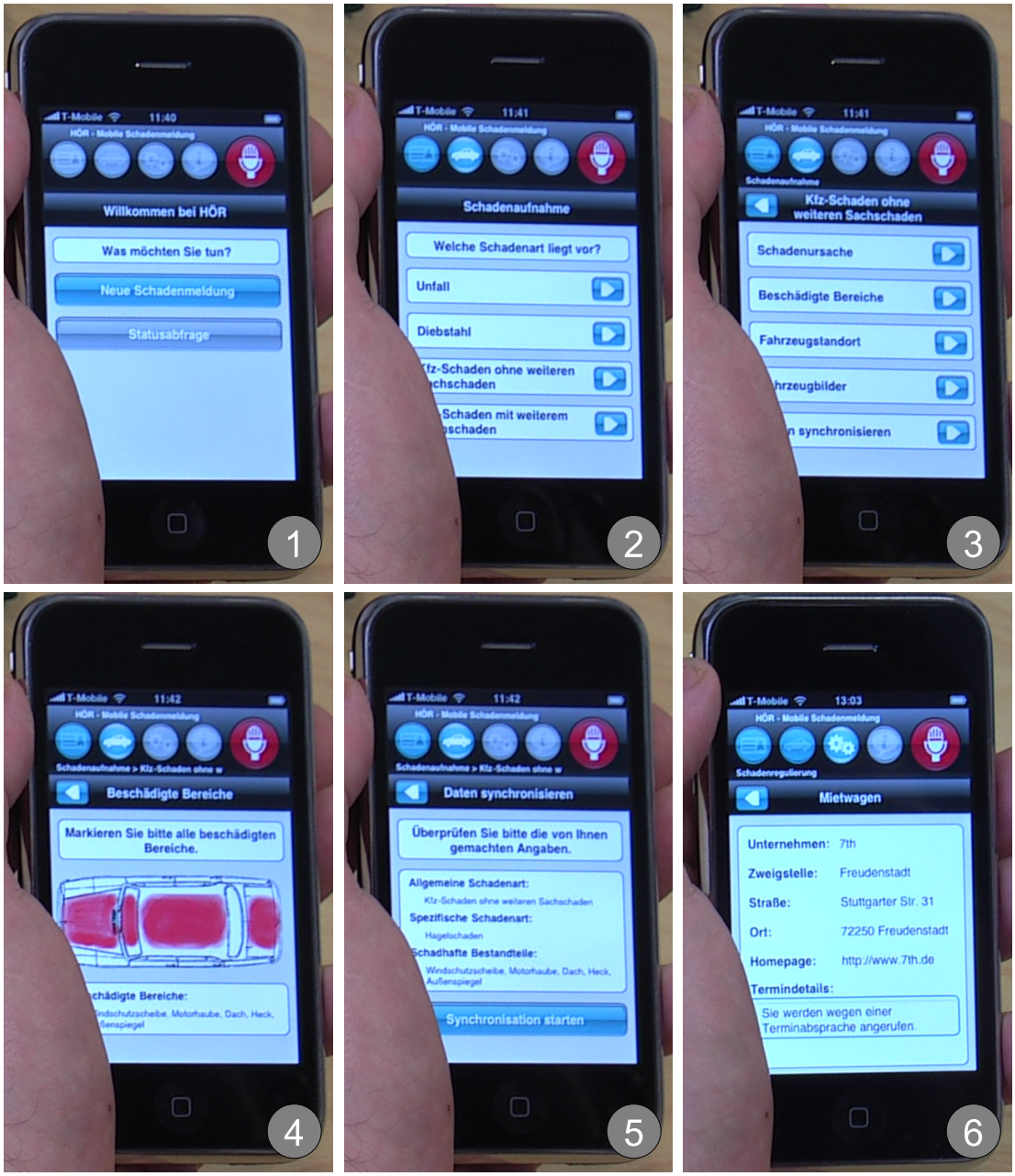


Abbildung 10.4: Mobile UI zur Schadenmeldung

Schadenart *Unfall* manuell zu selektieren, bietet die Versicherung ihrem Kunden als Alleinstellungsmerkmal eine multimodale Dialogschnittstelle zur Schadenaufnahme an. Es ergibt sich folgende Interaktionssequenz:

(1) BENUTZER: Mein Wagen hatte einen Hagelschaden. Überall sind Dellen im Lack und die Windschutzscheibe ist kaputt. [*Die Dialoganwendung bildet diese Aussage auf die entsprechende Schadenart ab und identifiziert auch die Frontscheibe als beschädigten Bereich.*]

(2) SYSTEM: Schadenart und zusätzliche Angaben erfasst. [*Die Dialogsystem wechselt zur Ansicht (3).*]

[*In (4) markiert der Geschädigte über eine Symbolgrafik weitere beschädigte Bereiche seines Fahrzeuges. Da nicht alle Details für eine Touchinteraktion geeignet sind, kann dies auch per Sprachinteraktion erfolgen.*]

(3) BENUTZER: Die Außenspiegel sind auch kaputt.

(4) SYSTEM: Die Liste der beschädigten Teile wurde ergänzt.

[*Der Geschädigte möchte zunächst keine weiteren Angaben machen und die Daten in (5) zur Versicherung übertragen.*]

(5) SYSTEM: Die Schadenmeldung wird nun übermittelt.

[*Es vergeht eine Weile, in der die Versicherung ein Mietwagenunternehmen mit der Bereitstellung eines Ersatzfahrzeuges für den Zeitraum der Reparatur beauftragt. Sobald der Mietwagen zur Verfügung steht, wird der Geschädigte in (6) über die mobile Anwendung darüber in Kenntnis gesetzt und um eine Terminvereinbarung zwecks Absprache gebeten.*]

(6) BENUTZER: Bitte erst nach 15h anrufen.

(7) SYSTEM: Termindetail hinzugefügt. [*Die zusätzliche Termininformation wird im Kundenmanagementsystem des Mietwagenunternehmens hinterlegt.*]

Zur Vermeidung künftiger Hagelschäden bietet die Versicherung im Rahmen einer Dienstinnovation zukünftig ihren Kunden exklusiv einen mobilen Wetterwarndienst an, der insbesondere Kraftfahrzeugführer vor schwer vorhersehbaren, lokal begrenzten und unmittelbar bevorstehenden Unwetterlagen warnt und auf nahegelegene Unterstellmöglichkeiten hinweist. Die mobile Benutzerschnittstelle des Wetterwarndienstes ist in Abbildung 10.5 dargestellt.

Im Falle eines bevorstehenden Unwetters in seiner Nähe wird der Kunde per Notifikation darauf hingewiesen. Es ergibt sich folgende Interaktionssequenz:

(1) SYSTEM: Nordwestlich Ihres Standorts hat sich eine lokale Gewitterzelle gebildet. Es ist mit Hagel zu rechnen.

(2) BENUTZER: Wo ist die nächste Unterstellmöglichkeit? [*Es werden Parkhäuser gesucht und in (2) angezeigt.*]

(3) SYSTEM: Fahren Sie am besten in das Karstadt-Parkhaus in der Kleine-

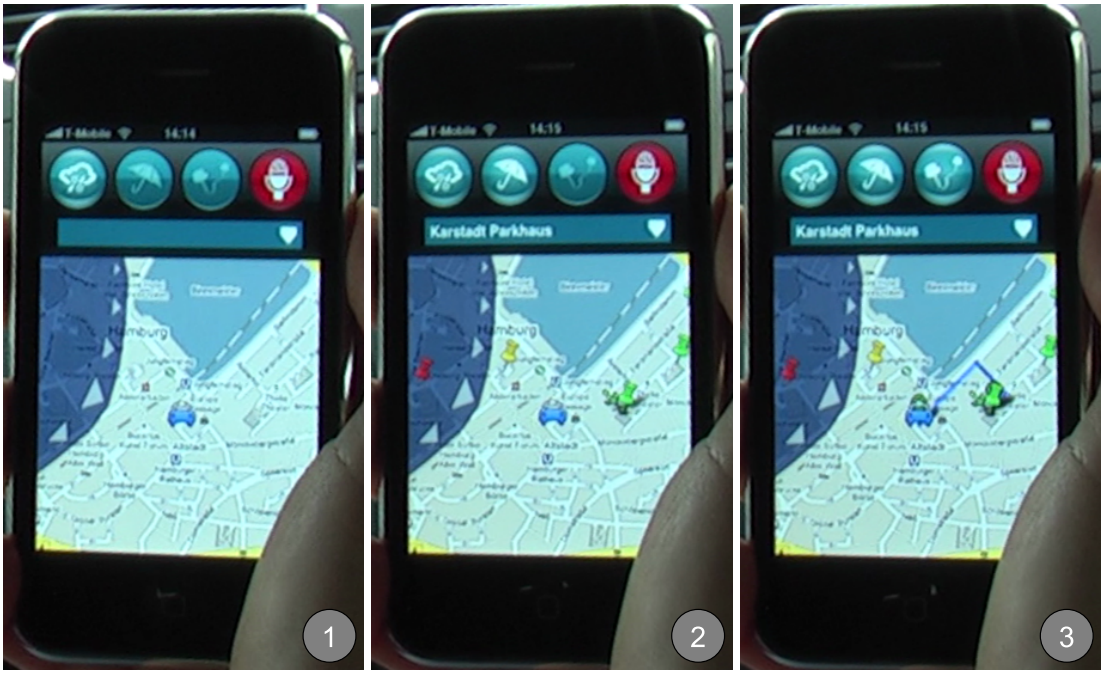


Abbildung 10.5: Mobile UI des Wetterwarndienstes

Rosen-Straße.

(4) BENUTZER: Wie komme ich dahin? [*Die Routenplanung berechnet die schnellste Route, welche in (3) angezeigt wird.*]

(5) SYSTEM: Route wird angezeigt.

Anwendung der Methodik

Die Demonstratoren wurden in einem vollständigen modellgetriebenen Entwicklungsverfahren (Neßelrath und Porta 2011) (Porta u. a. 2014a) ausgehend von einem formalen Geschäftsprozess erstellt. Hierbei kamen alle zu diesem Zweck in KomBInoS verfügbaren Werkzeuge zum Einsatz. Abbildung 10.6 zeigt beispielhaft die grafischen Aufgaben- und GUI-Editoren der KomBInoS-Werkbank.

Darin eingelassen ist die Ansicht der final gerenderten GUI des Wetterwarndienstes nach der Codegenerierung. UI-Fragmente für die Nutzung von Umkreissuche und Routenplanung konnten aufgrund der gemeinsamen Domänenmodellierung vom mobilen Reiseführer im Rahmen eines händisch komponierten Aufgabenmodells wiederverwendet werden. Die finale Zielplattform zur Nutzung der bislang vorgestellten KomBInoS-UIs bildet die wie beschrieben erweiterte ODP-Plattform. Die nachfolgend vorgestellten Demonstratoren nutzen als Zielplattform fortan die SiAM-Dialogplattform.

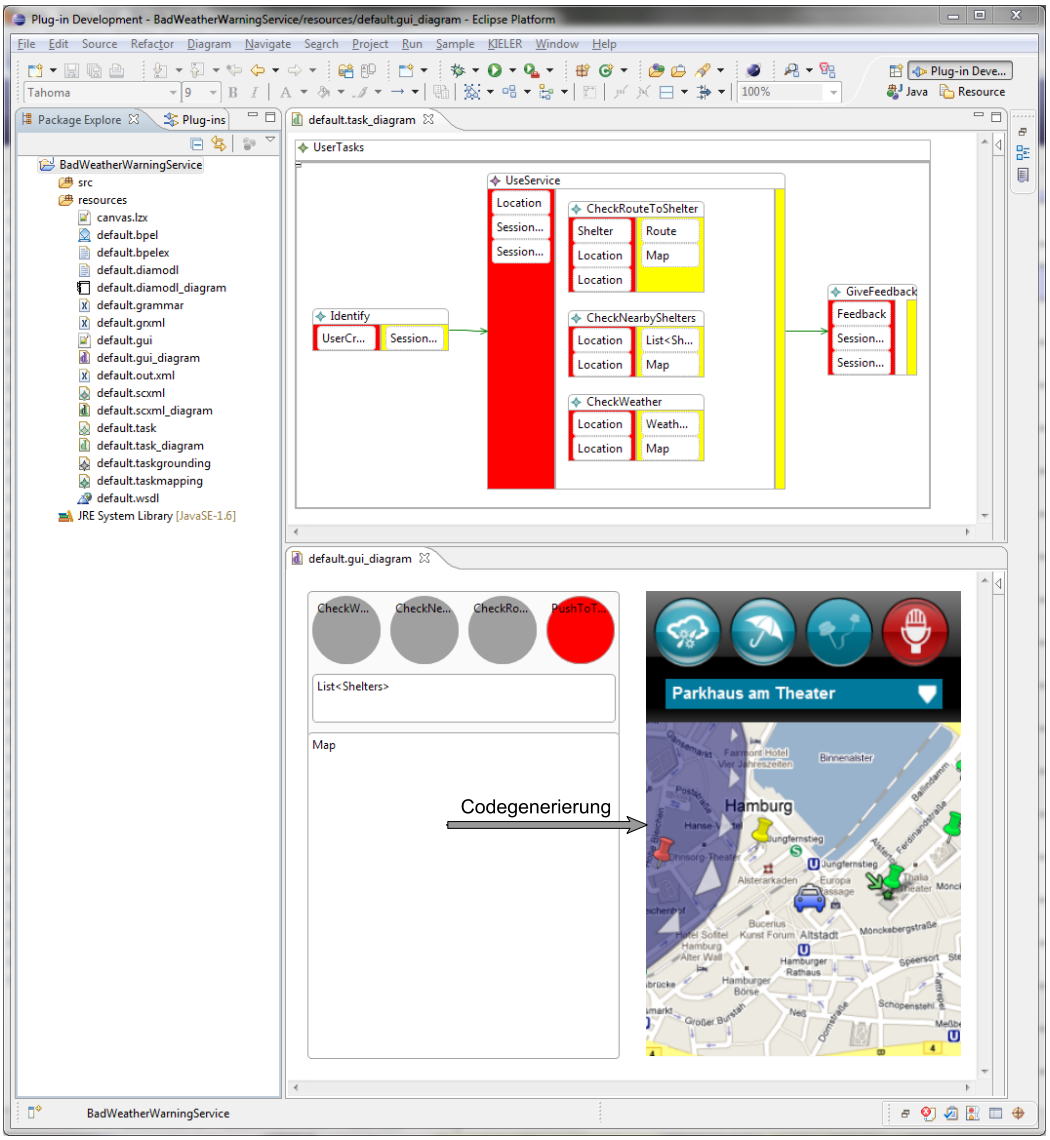


Abbildung 10.6: Entwicklung des Wetterwarndienstes

10.5 Multimodale dialogische Koordination von landwirtschaftlichen Ernteprozessen

Im Rahmen des BMBF-geförderten Software-Cluster-Verbundprojekts *Softwareinnovationen für das digitale Unternehmen* (SINNODIUM[1], Förderkennzeichen: 01IC12S01D) wurden prototypische Lösungen für die nächste Generation der Unternehmenssoftware geschaffen. Ein relevantes Anwendungsszenario beschäftigte sich mit smarter Produktion. Insbesondere die Landwirtschaft ist aufgrund des hohen Kostendrucks und des hohen Ausfallrisikos zu besonders effizienten Produktions- bzw. Ernteprozessen gezwungen. Die Effizienz dieser Prozesse hängt dabei in der modernen Landwirtschaft von einer komplexen Orchestrierung aller beteiligten Mitarbeiter und Landmaschinenfahrzeuge ab. Dennoch können äußere (z. B. Wetter, Änderung von Verkehrsbedingungen) oder innere Einflüsse (z. B. Schäden) diese sensible Struktur beeinträchtigen.

Vor diesem Hintergrund wurde mit dem Demonstrator *Landwirtschaftliche Produktion Multimodal* (LWP) (Porta u. a. 2014b; Tuncer u. a. 2014) ein cloud-basiertes System zur multimodalen dialogischen Koordination von landwirtschaftlichen Ernteprozessen implementiert. An einer Erntekampagne beteiligte Landmaschinenfahrer von Mähdreschern und Feldumschlagmaschinen werden so über Planänderungen im Ernteprozess aufgrund von gemeldeten Ereignissen oder verfügbaren Prozessdaten informiert. Weiterhin kann ein Fahrer Einzelheiten seines individuellen Auftrags im Dialog erfragen. Die mobile Anwendung erlaubt außerdem den Zugriff auf ein Anweisungs- und Nachweissystem, welches z. B. situationsabhängig benötigte Bedienhandbücher zur Verfügung stellt und durchgeführte Tätigkeiten gemäß einer gesetzlichen Nachweispflicht archiviert. Abbildung 10.7 zeigt ausschnittsweise die mobile Benutzerschnittstelle, die jedem Landmaschinenfahrer zur Verfügung steht.

Im vorliegenden Fall wird der Fahrer des Traktors mit der Kennziffer 08 in (1) während der Fahrt über ein Ereignis informiert, welches zu einer Umplanung seiner aktuellen Tätigkeiten geführt hat. Der Fahrer kann nun in (2) seinen neuen Auftrag im Dialog erkunden.

(1) SYSTEM: Achtung, Sie haben neue Anweisungen.

(2) BENUTZER: Was ist denn passiert?

(3) SYSTEM: Abfahrgespann 13 ist wegen eines Reifenschadens ausgefallen.

(4) BENUTZER: Was soll ich jetzt machen?

(5) SYSTEM: Fahren Sie zum Feld Mehlingen-Eckstraße. Anschließend laden Sie dort Maishäcksel auf.

(6) BENUTZER: Alles klar! Wie komme ich dahin? [*Es wird eine für das Gespann geeignete Route berechnet und in (3) angezeigt.*]

(7) SYSTEM: Die Route nach Mehlingen-Eckstraße wird auf der Karte ange-

[1] siehe http://www.software-cluster.com/de/forschung/projekte/verbundprojekte/sinnodium (letzter Zugriff: 04.11.2017)

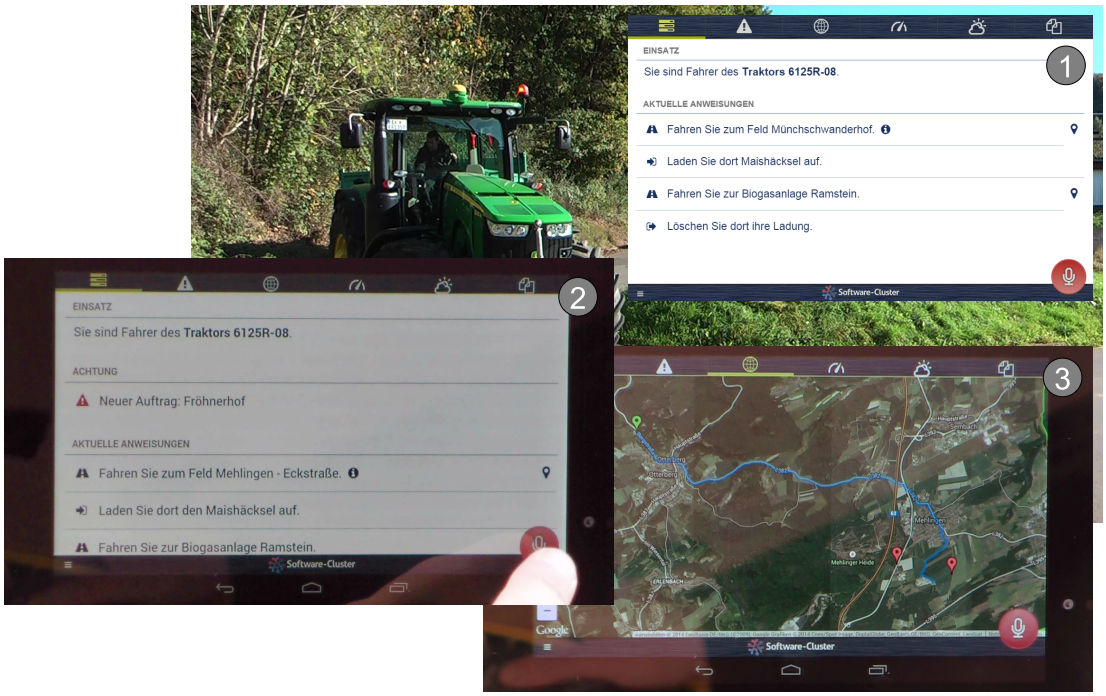


Abbildung 10.7: Mobile UI zur Koordination von landwirtschaftlichen Ernteprozessen

zeigt. Bitte beachten Sie die derzeit weiche Bodenbeschaffenheit des Feldes.

Anwendung der Methodik

Es wurde der vollständige modellgetriebene Entwicklungsprozess ausgehend von einer Prozessmodellierung in BPMN unter Verwendung aller assoziierten Werkzeuge eingesetzt. Hier liegt die Besonderheit darin, dass dieses Prozessmodell zwar die prototypische Sequenz der Tätigkeiten eines Landmaschinenfahres beschreibt, das Modell an sich aber aufgrund des hochdynamischen Umfeldes nicht von einer Prozessausführungsmaschine ausgeführt wird. Bei beibehaltenem Aufgabenmodell waren hierfür aufgrund des Technologiewechsels in der Zielplattform hin zu SiAM-dp umfangreiche Anpassungen in der Implementierung der Metamodell-Architektur und damit verknüpft den Modelltransformationen erforderlich. In diesem Zuge wurde SiAM-dp um die in Kapitel 9.4 (S. 216) beschriebenen Dialogkomponenten und Kommunikationsprotokolle SIP, MRCP, RTP und MQTT ergänzt.

10.6 Multimodale dialogische Unterstützung eines Marktleiters bei der Planung einer Sonderverkaufsfläche

Im Rahmen eines weiteren SINNODIUM-Anwendungsszenarios zu *emergenten Wissens- und Unterstützungsdiensten* wurde ein zweiter Demonstrator aus dem Bereich Smart Retail zur multimodalen dialogischen Unterstützung eines Marktleiters bei der Planung einer Sonderverkaufsfläche entwickelt. Für die Planung einer Aktions- oder Sonderverkaufsfläche erstellt ein Marktleiter oder eine andere sortimentsverantwortliche Person zu einem frühen Zeitpunkt ein Layout eines jeden Regals. Dabei werden in der Regel stationäre, tabellenbasierte Planungstools verwendet. Mit Hilfe der im Projekt entwickelten Planungsunterstützung unter Verwendung eines dreidimensionalen Abbilds des Marktes auf Basis von XML3D[2] können Produkte nun visuell verräumt, vertauscht und durch alternative Produkte der gleichen Kategorie ersetzt werden. Eine solche virtuelle Bestückung kann sowohl stationär als auch mobil auf der Verkaufsfläche erfolgen.

Abbildung 10.8 zeigt die mobile Benutzerschnittstelle, die eine gestenbasierte als auch eine erheblich komfortablere und schnellere sprachbasierte Navigation in der 3D-Welt ermöglicht.

Außerdem können komplexe Tätigkeiten zum Planen einer Sonderverkaufsfläche auf dem mobilen Endgerät durch multimodalen Dialog sehr einfach durchgeführt werden. So ergibt sich nach dem Einloggen in die Anwendung in (1) und der (gesten- oder sprachbasierten) Navigation zum Gemüseregal in (2) folgende Interaktionssequenz:

(1) BENUTZER: Gehe zum Müsli.

(2) SYSTEM: Ich gehe zum Müsli. [*Die Position wird durch eine animierte Kamerafahrt zum Müsliregal in (3) geändert.*]

(3) BENUTZER: Bitte mehr hiervon [↗]. [*Der Marktleiter zeigt auf ein Produkt.*]

(4) SYSTEM: Füge Müsli hinzu.

(5) BENUTZER: Bitte weniger Müsli der Marke XY.

(6) SYSTEM: Entferne Produkte der Marke XY.

(7) BENUTZER: Bitte vertausche diese [↗] beiden [↗] Produkte. [*Der Marktleiter zeigt nacheinander auf zwei Produkte.*]

(8) SYSTEM: Vertausche Produkte.

(9) BENUTZER: Bitte tausche dieses [↗] Produkt durch ein alternatives.

(10) SYSTEM: Produkt ausgetauscht.

[2] siehe http://xml3d.org/ (letzter Zugriff: 04.11.2017)

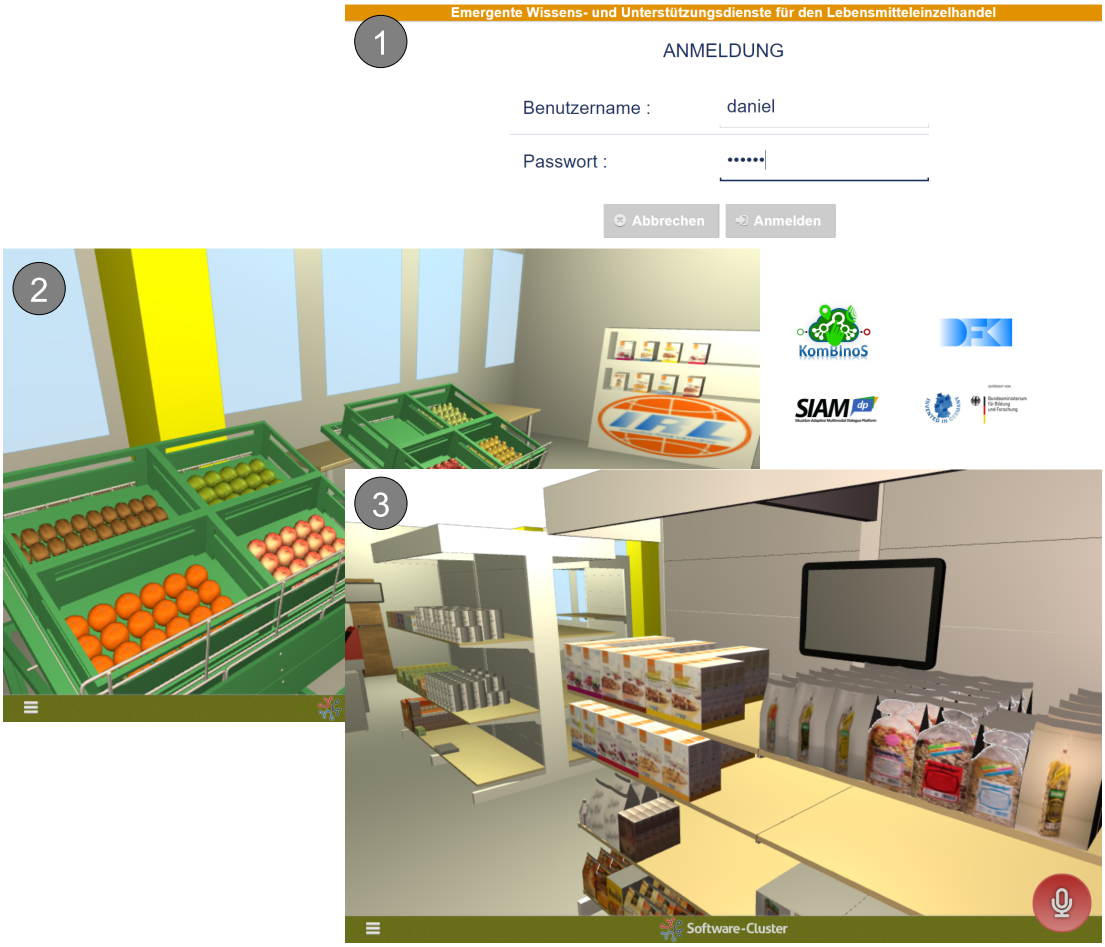


Abbildung 10.8: KomBInoS-UI zur Planung einer Sonderverkaufsfläche

Anwendung der Methodik

Die Planung einer Sonderverkaufsfläche ist eine Tätigkeit, die stark auf Erfahrungen und Präferenzen des Verantwortlichen beruht. Ein entsprechender Prozess wurde mit Hilfe von CMMN repräsentiert. Dieser ist in Abbildung 10.9 dargestellt und dient der vollständig angewandten modellgetriebenen Entwicklungsmethode als Startpunkt zur semiautomatischen Generierung der KomBInoS-UI.

Neben der erstmaligen Nutzung von CMMN in KomBInoS liegt eine weitere Besonderheit darin, dass die grafische UI im Wesentlichen aus einer XML3D-basierten dreidimensionalen Visualisierung des Marktes besteht. Hierfür musste das GUI-Metamodell um das Konzept 3DCanvas ergänzt werden. Die Auswahl dieses GUI-Konzepts im Rahmen der Generierung des GUI-Modells erfolgt auf Grundlage des TaskConcepts CyberPhysicalTask. Zur finalen Codegenerierung in HTML musste außerdem eine entsprechende Generierungsschablone erzeugt werden.

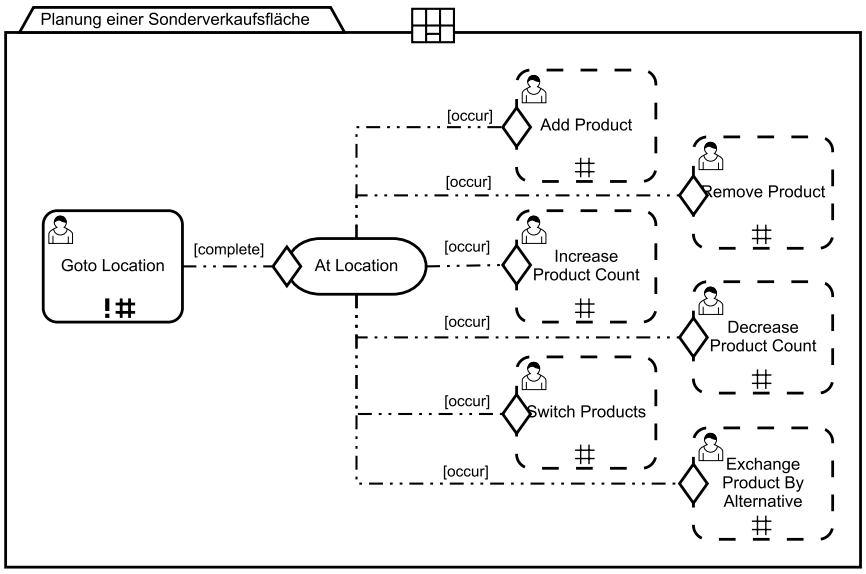


Abbildung 10.9: Planung einer Sonderverkaufsfläche als CMMN-Prozess

10.7 FactoryApps

Im Rahmen des BMBF-geförderten Industrie 4.0-Forschungsprojekts SmartF-IT[3] (Förderkennzeichen: 01IS13015A) wurden Konzepte zur Realisierung eines Factory-App-Stores für Prozessassistenzdienste entwickelt. Die darin enthaltenen Apps sollen sich kontextsensitiv an verschiedene Anwendungssituationen im Produktionsumfeld anpassen. Relevante Kontextfaktoren werden gemäß eines Benutzer-Rollen-Aufgaben-Fähigkeiten-Modells in einem Wissensgraphen beschrieben und beinhalten auch Fähigkeiten und Eigenschaften des Interaktionsgeräts sowie relevante Umweltfaktoren. Die Einsatzfähigkeit von SiAM-dp zum kontextsensitiven multimodalen Dialog in dieser Domäne wurde bereits von Neßelrath u. a. (2015) im Projekt Cypros unter Beweis gestellt. Zur Wahrung eines einheitlichen Corporate Designs sowie einheitlicher Bedienkonzepte wurden von Jafarian (2017) in SmartF-IT mit Hilfe von KomBInoS zwei verschiedene Anwendungen für die CPS-basierte Überwachung in Montagesystemen sowie für die Planung und Konfiguration der lokalen Materialbereitstellung entwickelt. Aufgrund der Planungstätigkeit lag der Fokus hierbei auf der Erstellung stationärer, großflächiger Anwendungen. Abbildung 10.10 zeigt die grafische Benutzerschnittstelle der Monitoring-Anwendung zur Abfrage von produktionsbezogenen Leistungskennzahlen.

Anwendung der Methodik

Beginnend jeweils von einer Prozessbeschreibung in CMMN, wurden grafische Benutzerschnittstellen für stationäre Anzeigegeräte implementiert. Hierzu wurde KendoUI

[3] siehe http://www.smartf-it-projekt.de/ (letzter Zugriff: 04.11.2017)

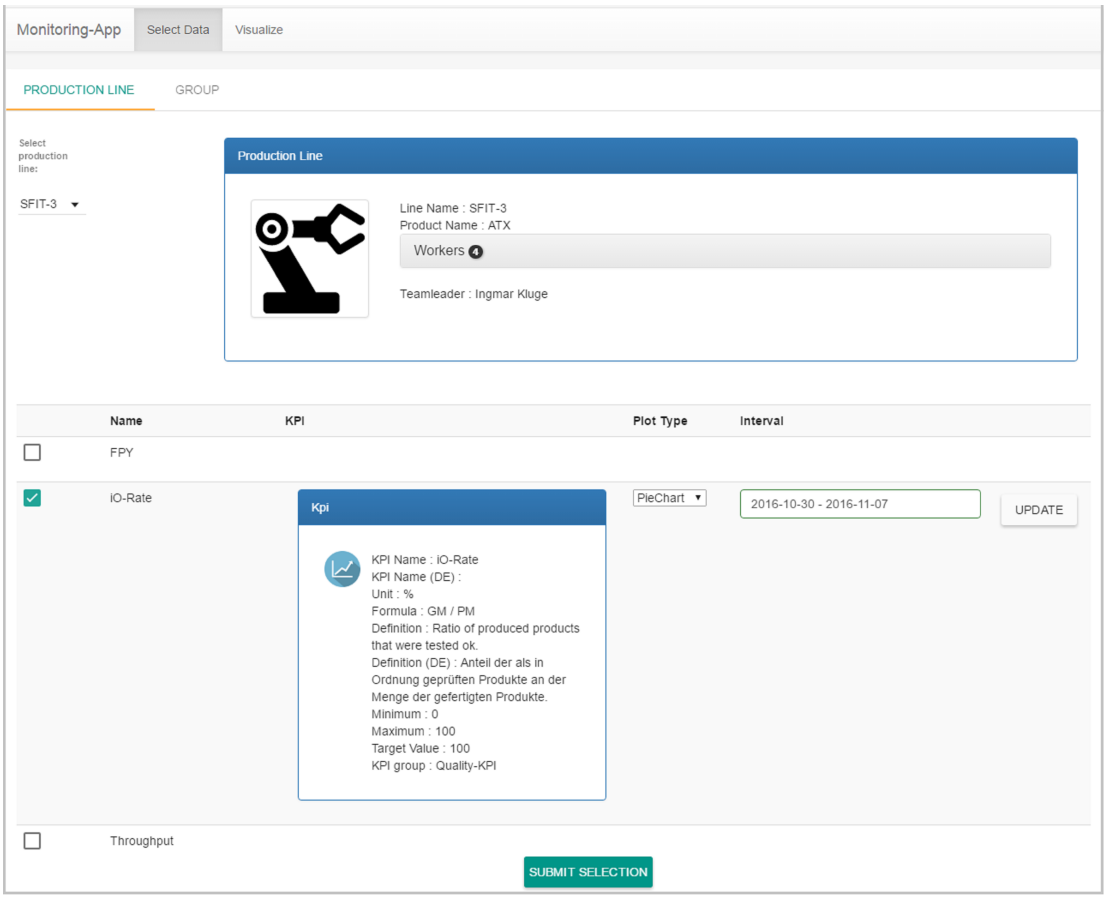


Abbildung 10.10: Grafische Benutzerschnittstelle zum Abfragen von Monitoringdaten in einer Industrie 4.0-Anwendung

mobile als Zielplattform für mobile GUIs gegen AngularJS ausgewechselt. In der Konsequenz wurde ein neuer Satz an M2C-Transformationen implementiert. Während der Entwicklung dieser Anwendungen hat sich bestätigt, dass sich die modellgetriebene semiautomatische Erstellung von Benutzerschnittstellen insbesondere für mobile UIs aufgrund der eingeschränkten Displaygröße eignet. Dies resultiert in überschaubareren grafischen UIs, deren Eingeschränktheit durch multimodalen Dialog konsistent und vollständig kompensiert werden kann. Sprachbasierte Interaktion stand hier nicht im Fokus, kann aber leicht ergänzt werden. Stattdessen wurde die Methode um einen Gruppierungsmechanismus erweitert, der es erlaubt, mehrere Aufgaben auf der ungleich größeren grafischen Bedienoberfläche stationärer Anzeigegeräte homogen abzubilden.

10.8 Hybr-IT-Demonstrator

Das BMBF-geförderte Projekt Hybr-IT[4] (Förderkennzeichen: 01IS16026A) erforscht die hybride und intelligente Mensch-Roboter-Kollaboration mit dem Ziel des Aufbaus und der Erprobung von hybriden Teams in wandlungsfähigen cyber-physischen Produktionssystemen. Hierbei ist eine ganzheitliche Betrachtung dieser Produktionssysteme einschließlich softwarebasierter Assistenzsysteme von Nöten.

Im Hybr-IT-Demonstrator wurde KomBInoS erstmals im Rahmen eines Montageprozesses zur Koordination von Werkern und kollaborativen Robotern eingesetzt. Der durchzuführende Montageprozess ist in Abbildung 10.11 dargestellt.

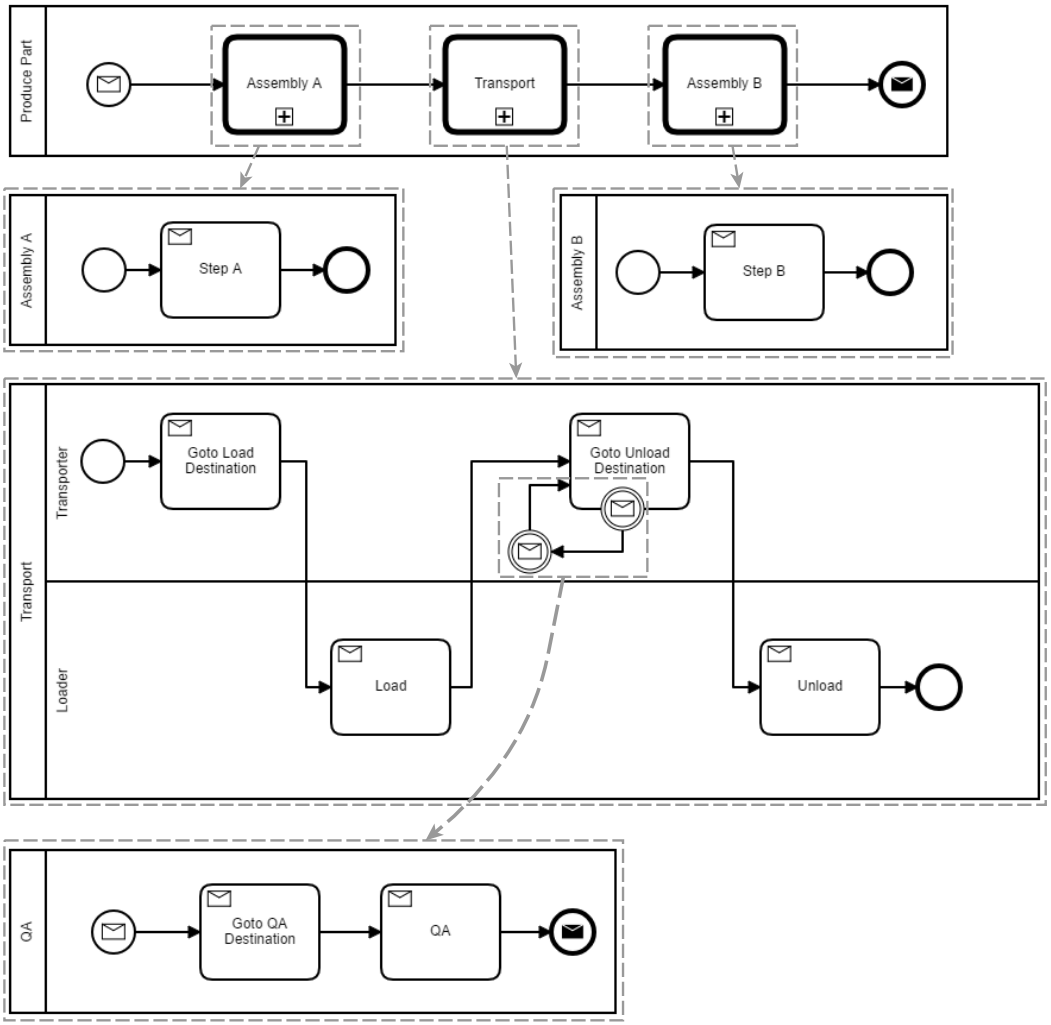


Abbildung 10.11: BPMN-basierter Montageprozess des Hybr-IT-Demonstrators

[4]siehe http://hybr-it-projekt.de/ (letzter Zugriff: 09.11.2017)

Nach einem ersten Montageschritt (*Step A*) ruft der menschliche Agent via Smartwatch einen Agenten zum Abtransport des bearbeiteten Werkstücks. Aufgrund der fähigkeitsbasierten Modellierung kann flexibel zur Laufzeit entschieden werden, welcher Agent den Transport durchführt. Im vorliegenden Fall geschieht dies schließlich durch eine autonom fahrende Roboterplattform, welche jedoch nicht über die notwendige Fähigkeit zum Beladen verfügt. Also wird als Ergebnis eines erneuten u. a. Fähigkeitenabgleichs dem Werker signalisiert, die Beladung zu übernehmen. Erhaltene Anweisungen müssen von Werkern quittiert werden. Die geschieht entweder durch Berühren des kapazitiven Displays oder, falls dies etwa durch das Tragen von Handschuhen nicht möglich ist, durch einen als Geste repräsentierten leichten Stoß unter Ausnutzung des Beschleunigungssensors analog zu Porta u. a. (2009b). Abbildung 10.12 zeigt diese Situation. Im Hintergrund ist die Management-GUI auf einem Tablet zu sehen. Darauf ist der aktuelle Prozessschritt (*Load*) hervorgehoben darstellt.

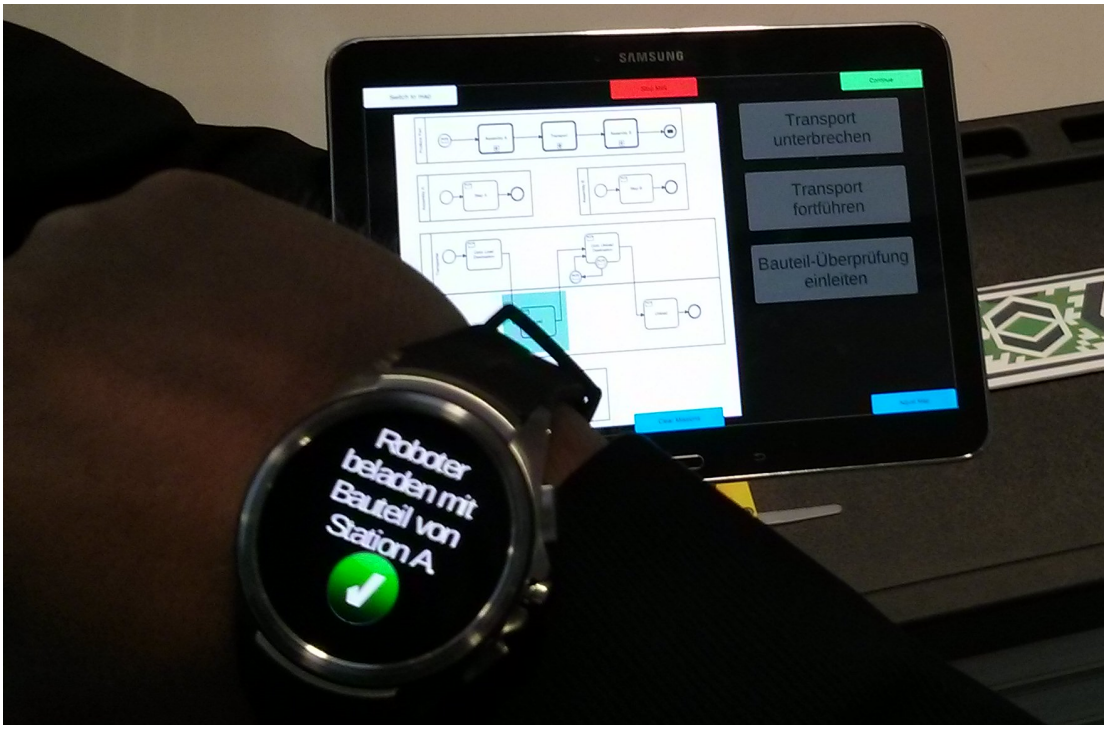


Abbildung 10.12: Smartwatch und Management-Tablet mit Montageprozess des Hybr-IT-Demonstrators

Während des Transports kann der Transportagent zur Qualitätssicherung an eine Qualitätssicherungsstation umgelenkt werden. Nach bestandener Prüfung wird der Transport zum eigentlichen Ziel fortgesetzt. Hier wartet ein weiterer Werker, der über die Verzögerung aufgrund der Qualitätssicherungsmaßnahmen informiert wurde, zur Fertigstellung der Montage (*Step B*).

Anwendung der Methodik

Mit Hilfe von Architekturprinzipien aus dem BMBF-geförderten Projekt BaSys 4.0[5] (Förderkennzeichen: 01IS16022E) und KomBInoS wurde der Hybr-IT-Demonstrator innerhalb kürzester Zeit umgesetzt. Hierbei wurde zuerst der in Abbildung 10.11 gezeigte ausführbare Montageprozess implementiert, auf dessen Grundlage im Rahmen einer modellgetriebenen Entwicklung unter Verwendung der entsprechenden KomBInoS-Werkzeuge schließlich die Werkerassistenz als browserbasierte GUI erstellt wurde, die wie im Falle des LWP-Demonstrators (Kapitel 10.5, S. 241) auf mehreren Smartwatches gleichzeitig angezeigt wird, jeweils mit anderer Benutzerkennung. Sprachinteraktion ist aufgrund fehlender Hardwarevoraussetzungen nicht möglich, wird aber in zukünftigen Weiterentwicklungen im Rahmen einer *Engineering- und Laufzeitumgebung für adaptive, multimodale Werkerassistenzdienste* (Kapitel 11.3, S. 265) berücksichtigt.

10.9 Nachbestellung von Waren

Im Rahmen des BMBF-geförderten Nachwuchsprogramms für IT-Führungskräfte *Software Campus*[6] (Industriepartner SAP SE, Förderkennzeichen: 01IS12050) wurde der Demonstrator zur mobilen Nachbestellung von Waren entwickelt. Thematisch ist er dem Anwendungsszenario des EWULEH-Demonstrators zuzuordnen und schlägt einen Bogen zurück zum mobilen TEXO-Client. Abbildung 10.13 zeigt die grafische Benutzerschnittstelle des Vorgesetzten zur Genehmigung einer Nachbestellung.

Zur Entscheidungsunterstützung sind entsprechende Funktionen analog zum TEXO-Client vorhanden. Die zu erledigenden Aufgaben des Benutzers in den von ihm wahrgenommenen Rollen wird über die Camunda-Integration der KomBInoS-Dialogplattform direkt aus der Prozessausführungsmaschine abgefragt und in der mobilen UI zur Anzeige gebracht. Auf die gleiche Weise werden Benutzereingaben über die Dialogplattform in den Prozess zurückgespielt.

Anwendung der Methodik

Der Fokus dieses Demonstrators lag auf einer vollständigen Abdeckung der KomBInoS-Methodik als Proof-of-Concept des modellgetriebenen Kompositionsansatzes. Dementsprechend wurden zuerst zwei Beispielprozesse samt KomBInoS-UIs zur Nachbestellung von Waren und zur Genehmigung eines Vorgangs umgesetzt. Diese wurden in einem Nachbestellprozess mit Genehmigung gemäß der in Kapitel 6.8.1 (S. 137) geschilderten Kompositionsmuster komponiert. Abbildung 10.14 zeigt die BPMN-Prozesse sowie die im jeweiligen Anwendungsmodell enthaltenen Aufgabenmodelle samt Kompositionsmodell.

[5] siehe http://www.basys40.de/ (letzter Zugriff: 04.11.2017)

[6] siehe http://www.softwarecampus.de/aktuelles/forschungsprojekte/projekt/kombinos-modellgetriebene-komposition-von-multimodalen-dialogischen-dienst-und-prozessbenutzungsschnittstellen-im-internet-der-dienste/ (letzter Zugriff: 04.11.2017)

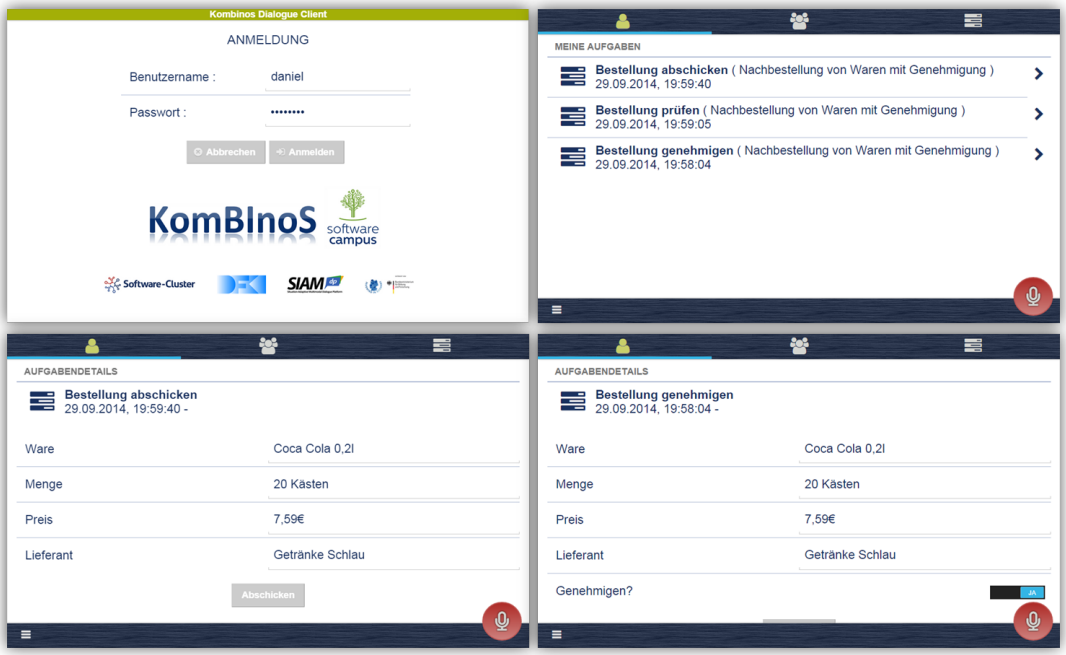


Abbildung 10.13: Grafische Benutzerschnittstelle des Software-Campus-Demonstrators

10.10 Fazit

In diesem Kapitel wurden acht Demonstratoren in den unterschiedlichsten fachlichen Anwendungsdomänen vorgestellt, welche alle zu einer kontinuierlichen Weiterentwicklung der KomBInoS-Methodik beigetragen haben. Mit KomBInoS wurden mobile Assistenzdienste im Tourismusbereich, in der Schadenregulierung und -vermeidung im Versicherungswesen, für Smart Farming und Smart Retail implementiert und u. a. im Rahmen der CeBIT 2011 und 2014 ausgestellt. Zudem wurden mobile und stationäre Assistenzdienste für die Industrie 4.0 entwickelt.

In Abbildung 10.15 werden die Demonstratoren nach dem Umfang des Einsatzes von KomBInoS-Bestandteilen verglichen.

Es wird deutlich, dass der Einsatz von KomBInoS, u. a. durch den modularen Aufbau der (Metamodell-) Architektur, sehr flexibel gestaltet werden kann. Dies führt zu den in Kapitel 2.4.3 (S. 22) geschilderten Vorteilen der modellgetriebenen Softwareentwicklung und wirkt deren Nachteile entgegen, sodass insgesamt der Nachweis der Anwendbarkeit der KomBInoS-Methodik erbracht ist.

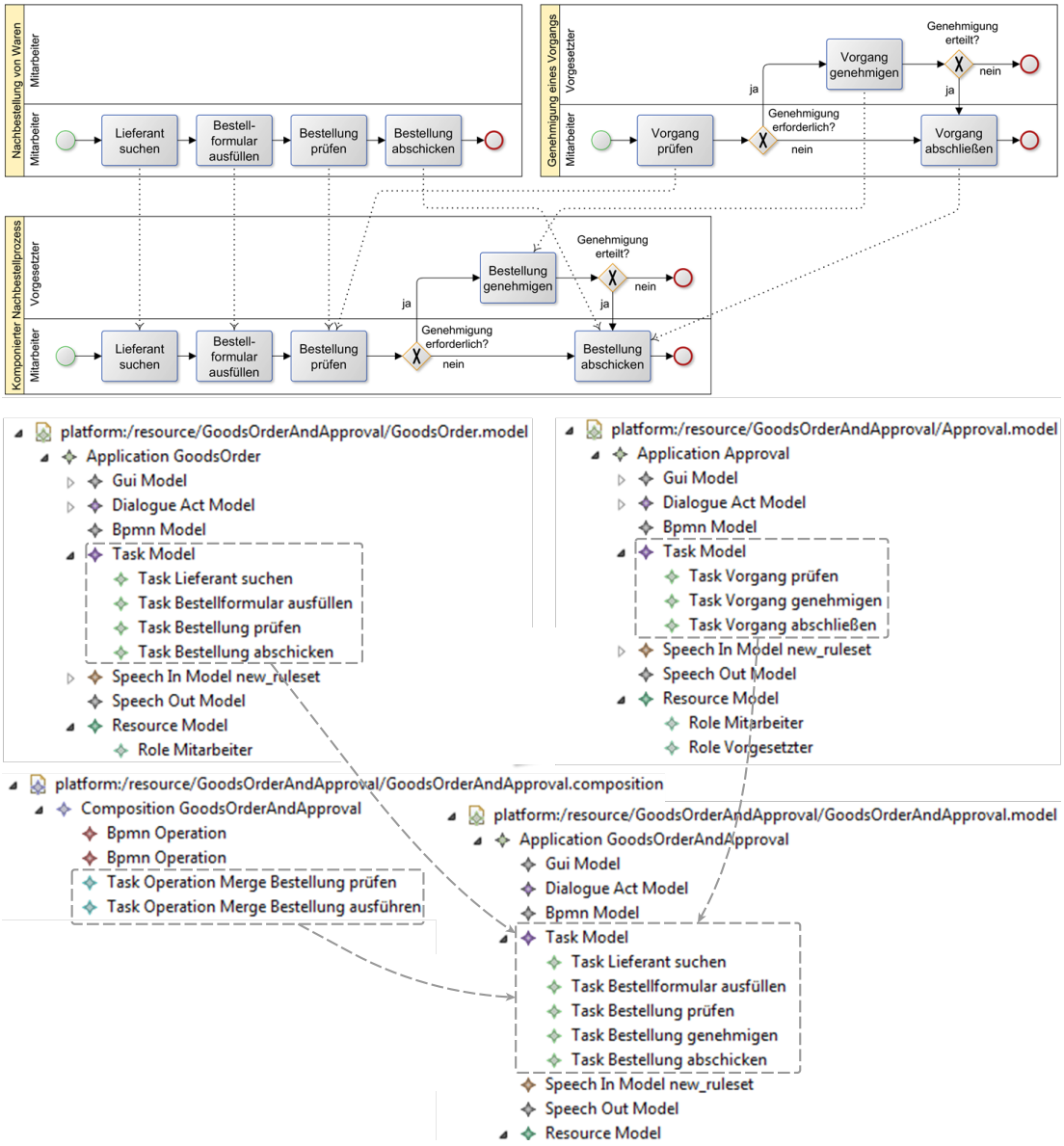


Abbildung 10.14: Modellgetriebene Komposition des Software-Campus-Demonstrators

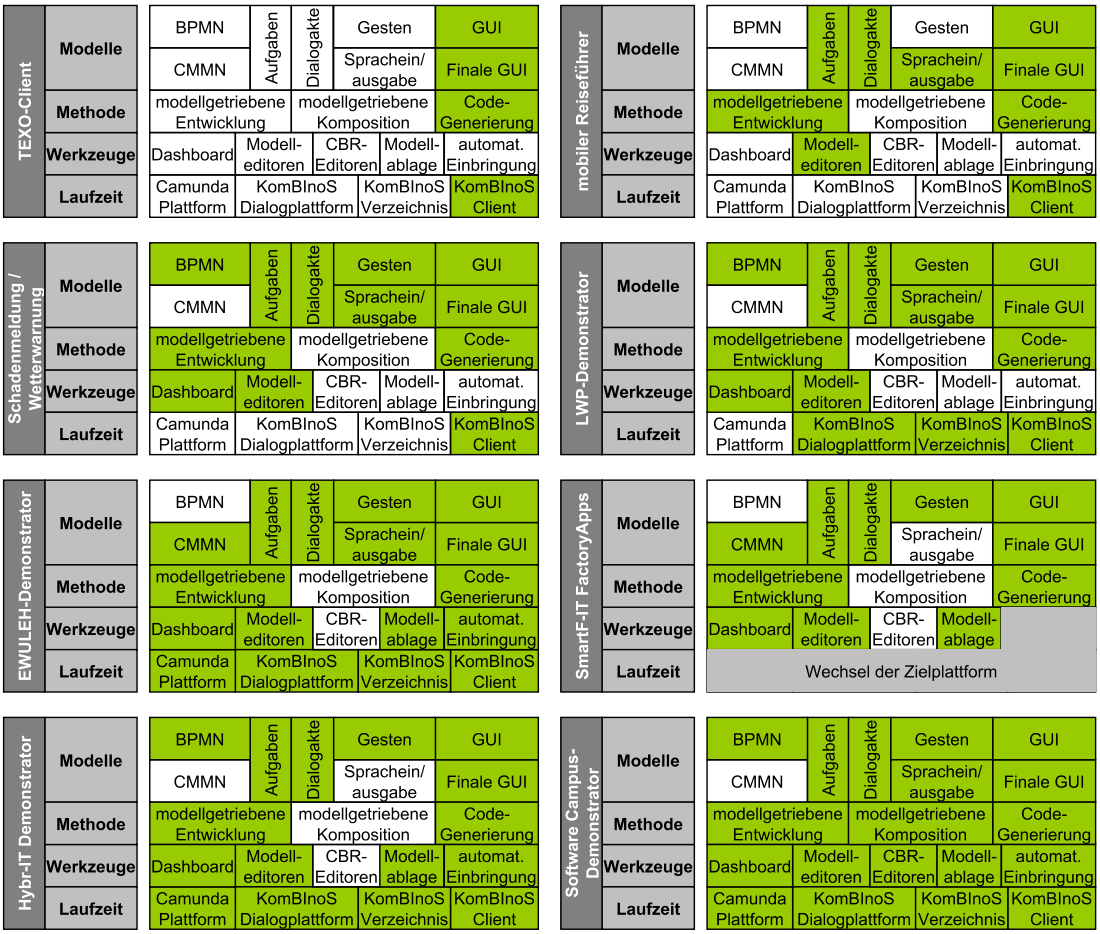


Abbildung 10.15: Featurematrix der KomBInoS-Demonstratoren

Teil IV

Diskussion

KAPITEL 11

Zusammenfassung und Ausblick

Das letzte Kapitel fasst die wesentlichen Beiträge und erreichten Ergebnisse der Arbeit zusammen und liefert Antworten auf die eingangs formulierten wissenschaftlichen und ingenieurtechnischen Fragestellungen. In einem Ausblick werden zukünftige Arbeiten und sinnvolle Ergänzungen dieser Arbeit diskutiert.

11.1 Zusammenfassung

Das Ziel der Arbeit war die Entwicklung eines ganzheitlichen Ansatzes zur effizienten Erstellung von multimodalen Dialogschnittstellen für Smart Services. Um dieses Ziel zu erreichen, wurde eine umfassende Methodik zur modellgetriebenen Erstellung von Benutzerschnittstellen gemäß Vanderdonckt (2005, S. 18) entwickelt. Die Methodik besteht aus:

(1) der in Kapitel 6 (S. 111) entwickelten **Metamodell-Architektur**, welche sowohl eine modellgetriebene Entwicklung als auch die Komposition von Benutzerschnittstellen erlaubt,

(2) dem in Kapitel 7 (S. 143) entwickelten **methodischen Vorgehen**, welches aus aufeinander abgestimmten Modelltransformationen, möglichen Kompositionsschritten und manuellen Entwicklungstätigkeiten besteht, sowie

(3) der in Kapitel 8 (S. 187) entwickelten **integrierten Werkzeugkette** auf Basis der Eclipse IDE als Implementierung der Methode.

Es wurde in Kapitel 9 (S. 211) außerdem eine cloud-fähige **Laufzeitumgebung zur mobilen Nutzung** der so erstellten Benutzerschnittstellen auf Basis der SiAM-Dialogplattform entwickelt. Als Proof-of-Concept wurden in Kapitel 10 (S. 231) acht **Beispielanwendungen und Demonstratoren** aus fünf Forschungsprojekten vorgestellt. Zusätzlich zur Smart Service Welt fand und findet KomBInoS auch Anwendung im Bereich der Industrie 4.0.

11.2 Beiträge

Die folgenden Abschnitte erläutern die wissenschaftlichen, ingenieurtechnischen und praktischen Beiträge der Arbeit sowie die Beiträge zur Smart Service Welt. Außerdem werden die im Rahmen der Arbeit entstandenen Veröffentlichungen aufgeführt.

11.2.1 Wissenschaftliche Beiträge

Die wissenschaftlichen Beiträge dieser Arbeit werden durch Beantwortung der in Kapitel 1.2.3 (S. 4) formulierten Forschungsfragen in Form von Kurzzusammenfassungen herausgestellt.

(1) **Modellgetriebene Entwicklung:** *Wie kann man multimodale Dialogschnittstellen für Smart Services schnell und mit geringem Programmieraufwand entwickeln? Wie sehen eine notwendige Metamodell-Architektur und entsprechende Modelltransformationen aus, die funktional vollständige und über Modalitätsgrenzen hinweg konsistente Benutzerschnittstellen erschaffen?*

Im Rahmen der Arbeit wurde in Kapitel 6 (S. 111) eine vollständige Metamodell-Architektur zur modellgetriebenen UI-Entwicklung konform zum Cameleon-Referenzrahmenwerk entworfen. Hierdurch wird eine klare fachliche Trennung zwischen unterschiedlichen Abstraktionsebenen erreicht. Weiterhin wurden in Kapitel 7.3 (S. 147) M2M-Transformationen spezifiziert, die einen semiautomatischen UI-Entwicklungsprozess ausgehend von formal beschriebenen Kollaborationprozessen umsetzen. Dadurch ist es möglich, vollständige und konsistente Benutzerschnittstellen in Bezug auf den zugrundeliegenden Prozess schnell und damit effizient zu erstellen. Der Schlüssel zur Auflösung starrer Modalitätsklassen liegt in der Verwendung von Dialogakten auf Ebene der abstrakten Benutzerschnittstelle. Nur so kann aufgrund der gewonnenen Flexibilität an dieser zentralen Position der Metamodell-Architektur und mit Hilfe entsprechender Verarbeitungskomponenten zur Laufzeit eine synergistische Multimodalität erzeugt werden. Dies stellt im Vergleich zu den vorgestellten verwandten Arbeiten einen wesentlichen Fortschritt dar. Die Ebene der konkreten UI-Modelle umfasst neben Modellen für die natürlichsprachliche Interaktion und die grafische Benutzerschnittstelle auch paralinguistische Ein- und Ausgabemodelle, etwa zur Gestensteuerung oder zur Steuerung eines virtuellen Charakters. Weitere Modelle können flexibel integriert werden. Die finale UI wird in einer M2C-Transformation technologisch auf moderne Webstandards und -Bibliotheken abgebildet. Dies gilt nicht nur für die grafische sondern gleichwohl auch für die natürlichsprachliche Benutzerschnittstelle in Form von Modellen zur Spracheingabe und -ausgabe. Letztere stammen aus der SiAM-Dialogplattform und wurden so erweitert, dass sie sich homogen in die Metamodell-Architektur einfügen.

(2) **Modellgetriebene Komposition:** *Wie kann man existierende multimodale Dialogschnittstellen von Smart Services wiederverwenden? Bis zu welchem Grad ist*

dies sinnvoll möglich? Wie können darüber hinaus auch Modellartefakte multimodaler Dialogschnittstellen wiederverwendet werden, deren zugehörige Dienste nicht Teil der Kompositionskette sind?

In Kapitel 6.8 (S. 137) wurde ein Kompositionsmodell auf Basis einer vorangegangenen Anforderungsanalyse entwickelt. Die Analyse hat drei Szenarien beschrieben und untersucht, in denen eine Komposition von Kollaborationsprozessen als Ausgangspunkt für eine anschließende UI-Komposition prinzipiell sinnvoll erscheint. Folglich enthält das Kompositionsmodell initial nur Information über die Komposition auf Prozessebene. Durch die Anwendung der semiautomatischen Erstellungsmethode, welche im nächsten Punkt eine gesonderte Behandlung erfährt, wird das Kompositionsmodell nun sukzessive aufgebaut. Dieses stellt somit einen nachvollziehbaren Verlauf der durchgeführten UI-Kompositionen auf Basis der ursächlichen Prozesskomposition dar. Die Modelltransformationen sind so gestaltet, dass sie konsequent zwischen neu erzeugten und komponierten UI-Anteilen unterscheiden und im Kompositionsmodell hinterlegen, welche Elemente auf welcher Basis wie komponiert wurden. Durch dieses Vorgehen können manuell erfolgte Nachbearbeitungen von wiederverwendeten UI-Artefakten in der Kompositionskette in die aktuell erstellte Benutzerschnittstelle übertragen werden. Mit jeder Konkretisierung der UI wächst tendenziell der Umfang der UI-Modelle. Durch Anwendung von fallbasiertem Schließen als überwachtem Lernansatz auf konkreten UI-Modellen, insbesondere Modellen für die Spracheingabe und -ausgabe, werden auch nicht-komponierte, vollständig neu entwickelte KomBInoS-UIs initial angereichert.

(3) **Integrierte Entwicklungs- und Kompositionsmethode:** *Wie sieht ein strukturiertes Vorgehen aus, welches sowohl Entwicklungs- als auch Kompositionstätigkeiten bei der Erstellung einer multimodalen Dialogschnittstelle für einen Smart Service integriert?*

Die modellgetriebene Entwicklung ist in der Regel ein semiautomatischer Prozess, bestehend aus automatischen Modelltransformationen und manuellen Verfeinerungen. Das in Kapitel 7 (S. 143) entwickelte Vorgehen vereint die modellgetriebene Entwicklung mit einem modellgetriebenen Kompositionsansatz in einer einzigen, integrierten Methode. Dadurch entsteht eine neue Komplexität bei der Erstellung einer Benutzerschnittstelle, die dem Entwickler dennoch nach Möglichkeit verborgen bleibt. Dies wird durch abgestufte Kompositions- oder Komplexitätsniveaus erreicht, die einen Rückschluss auf den Grad der notwendigen Intervention seitens des Entwicklers zulassen. So beschreibt die Methode einerseits, welche manuellen Tätigkeiten pro Modellebene bei der reinen Entwicklung durchzuführen sind, indem sie die zu erbringende Information auf Modellebene zur Ausführung der nächsten Modelltransformation definiert. Andererseits beschreibt sie, worauf ein Entwickler in welchem Detailgrad bei der gegebenenfalls angezeigten manuellen Nachbearbeitung einer Komposition zu achten hat. Durch diese klaren Modellschnittstellen wird schließlich eine Verteilung der Erstellung

auf mehrere Entwickler ermöglicht.

11.2.2 Ingenieurtechnische Beiträge

Wie im vorangegangenen Abschnitt werden an dieser Stelle die ingenieurtechnischen Beiträge dieser Arbeit als Antworten auf die entsprechenden Fragen in Kapitel 1.2.3 (S. 4) zusammengefasst.

(4) **Nahtlose Werkzeugunterstützung:** *Wie können Entwickler bei der Erstellung von multimodalen Dialogschnittstellen für Smart Services bei ihrer Aufgabe unterstützt werden?*

Die in Kapitel 8 (S. 187) beschriebene KomBInoS-Werkbank deckt den vielschichtigen Erstellungsprozess unter Zuhilfenahme existierender Eclipse-basierter Referenzimplementierungen unterschiedlichster MDA-Standards aus dem Umfeld des Eclipse Modeling Frameworks technisch vollständig und nahtlos ab. Methodisch wird der Entwickler sicher durch den komplexen Prozess geleitet. Dabei wird er effektiv bei der Anwendung der integrierten Erstellungsmethode unterstützt. Ein Projektwizard erzeugt initial das Grundgerüst einer KomBInoS-UI. Der Entwickler wird von einem Dashboard Schritt für Schritt durch den modellgetriebenen Erstellungsprozess geleitet. Hierbei kann er direkt aus dem Dashboard heraus Modelltransformationen anstoßen. Zur manuellen Vervollständigung der dadurch sukzessive erzeugten Teilmodelle stehen geeignete Editoren mit Kompositionsfunktionalität zur Verfügung. Diese sind in einer eigenen Perspektive angeordnet, um alle KomBInoS-relevanten Informationsaspekte auf einen Blick erfassen zu können. Der aktuelle Stand der Erstellung sowie noch fehlende Information sind ihm so jederzeit bekannt. Die Werkbank hat einerseits Zugriff auf eine zentrale Modellablage. Diese bietet Zugang zu den Modellen bereits erstellter KomBInoS-UIs und ermöglicht durch das Zusammenführen von UI-Modellen direkt auf Modellebene sowie durch Änderungsbenachrichtigungen zusätzlich die verteilte Erstellung einer UI. Andererseits hat die Werkbank direkten Zugriff auf eine Instanz der Laufzeitumgebung zum Inbetriebnehmen fertiger KomBInoS-UIs am Ende des Erstellungsprozesses.

(5) **Laufzeitumgebung:** *Wie kann man Smart Services im multimodalen Dialog auf mobilen Endgeräten nutzen?*

Die KomBInoS-Laufzeitumgebung (Kapitel 9, S. 211) zur (mobilen) multimodalen Dialoginteraktion mit Smart Services implementiert eine offene Architektur unter Ausnutzung relevanter Kommunikationsstandards und -protokolle. Dadurch wird eine lose Kopplung der funktionalen Komponenten etabliert, welche eine technische Verteilung der Komponenten in einer Cloud-Infrastruktur ermöglicht. Zu diesen Komponenten zählen (1) die Camunda BPM-Plattform im Back-End, welche Kollaborationsprozesse in BPMN- oder CMMN-Notation ausführen kann und durch eigene Erweiterungen relevante Prozessereignisse per MQTT publiziert, (2) die KomBInoS-Dialogplattform, einer erweiterten SiAM-Dialogplattform zur

Ausführung von KomBInoS-UIs, (3) der KomBInoS-Verzeichnisdienst mit Proxyfunktionalität und Integrationsmöglichkeiten für externe Dienste sowie (4) ein leichtgewichtiger mobiler KomBInoS-Client zur multimodalen Dialoginteraktion mit KomBInoS-UIs für Smart Services im Front-End. Die Laufzeitumgebung erlaubt eine Kontaktaufnahme der Dialogplattform mit Agenten, etwa im Rahmen länger laufender Prozesse. So können diese zu jeder Zeit mit für sie relevanten Prozessen in Kontakt treten und über neue Entwicklungen in Kenntnis gesetzt werden.

Insgesamt ergibt sich so auf Basis des in Kapitel 5.5 (S. 104) definierten Anforderungskatalogs der in Tabelle 5.2 (S. 107) dargestellte Vergleich der verwandten Arbeiten mit KomBInoS.

11.2.3 Praktische Beiträge

Bei der Entwicklung der Metamodell-Architektur wurden wiederverwendbare Referenzmodelle erzeugt, die nachfolgend unter Punkt (1) Erwähnung finden. Im Rahmen der Werkbankentwicklung wurden generische und wiederverwendbare Werkzeugmodule implementiert, die als praktische Beiträge (2) und (3) aufgeführt werden. Auch im Zuge der Entwicklung der Laufzeitumgebung wurden wiederverwendbare und domänenneutrale Laufzeitmodule implementiert, die die SiAM-Dialogplattform um sinnvolle Fähigkeiten ergänzen. Diese werden als praktische Beiträge (4) und (5) aufgeführt.

(1) **Ecore-basierte Referenzmodelle:**

Auf Basis der normativen XML-Schemata von CMMN zur Abbildung schwach strukturierter Prozesse (Kapitel 6.3.2, S. 115) sowie SSML zur deklarativen Beschreibung von Sprachsynthesevorschrifen (Kapitel 6.7.2, S. 131) wurden teilautomatisiert Ecore-basierte Metamodelle erstellt, deren Instanzen syntaktisch gemäß der Schemata dargestellt werden. Die Metamodelle können unabhängig von KomBInoS genutzt werden und Ausgangspunkt der Entwicklung Eclipse-basierter Werkzeugunterstützung sein, etwa in Form spezialisierter Editoren.

(2) **Modellbasiertes Transformationsdashboard zur methodischen Unterstützung modellgetriebener Softwareentwicklung:**

Im Rahmen der KomBInoS-Werkbank wurde ein konfigurier- und wiederverwendbares Rahmenwerk für jedwede Form von Dashboards zur modellgetriebenen Softwareentwicklung implementiert (Kapitel 8.4, S. 192). Dazu lässt sich die Darstellung und die Funktionalität eines Dashboards selbst auch durch ein Modell formulieren. Das Dashboard-Rahmenwerk ist in eigenständige Eclipse-Plugins aufgeteilt, die einen eigenen Erweiterungspunkt zur Deklaration von beliebigen Dashboards bereitstellen. Ein solches Dashboard stellt visuell den Erstellungsprozess nach und bietet somit eine methodische Unterstützung. Darüber hinaus stellt es aus Benutzersicht einen Integrationspunkt für eine durchgängige Werkzeugkette dar.

(3) **Technisches Rahmenwerk zum fallbasierten Schließen auf Ecore-basierten Metamodellen:**

Ebenfalls im Rahmen der KomBInoS-Werkbank wurde jCOLIBRI, ein Java-basiertes Rahmenwerk für CBR-Anwendungen, maßgeblich für fallbasiertes Schließen auf Ecore-basierten Metamodellen erweitert (Kapitel 8.8, S. 199). Dazu wurde mit Hilfe von EMF ein generisches Metamodell geschaffen, welches als Ausgangspunkt zur Modellierung domänenspezifischer Fallrepräsentation im Rahmen einer Fallbasis dient. Außerdem wurde eine Integrationsschicht entwickelt, die eine homogene Verbindung zwischen jCOLIBRI und der Modellwelt herstellt. Darauf aufbauend können schnell auf neue Domänen angepasste Editoren zur vollständigen Abdeckung des CBR-Zyklus entwickelt werden.

(4) **Ereignisbasierte Prozesssteuerung via MQTT:**

Die eingesetzte Camunda Prozessausführungsmaschine wurde um die Fähigkeit erweitert, für die Prozessausführung relevante Ereignisse mit Hilfe des standardisierten Kommunikationsprotokolls MQTT an interessierte Konsumenten zu übermitteln. Darüber hinaus wurde ein leichtgewichtiges Gateway implementiert, welches beliebige MQTT-Ereignisse innerhalb der OSGi-Serviceplattform als originäre OSGi-Ereignisse publiziert. Damit stehen solche Ereignisse allen OSGi-Komponenten zur Verfügung, ohne selbst über die Mittel zur Kommunikation via MQTT verfügen zu müssen. Die REST-basierte Camunda-API wurde außerdem in eine höherwertige Java-API innerhalb eines unabhängigen OSGi-Bündels gekapselt. Darauf aufbauend wurden KomBInoS-spezifische Komponenten zur Prozess- und Interaktionssteuerung konzipiert und implementiert, wie in Kapitel 9.4 (S. 220) geschildert.

(5) **Domänenneutrale Erweiterung der SiAM-Dialogplattform:**

Es wurden Bibliotheken und OSGi-basierte Dienste zur Kommunikation via SIP, MRCP und RTP implementiert und nahtlos in SiAM-dp integriert. Darauf abgestimmt wurde ein mobiler Client für die Android-Plattform auf Basis von Apache Cordova implementiert. Die Nutzung der Kommunikationsprotokolle und des Clients ist an keine bestimmte Domäne geknüpft, ermöglicht aber die gemischte Initiative bei länger laufenden, unterbrochenen Interaktionen. Darüber hinaus erlauben diese Erweiterungen eine Nutzung von SiAM-dp über Internettelefonie.

Auch konzeptionell kann SiAM-dp von der in dieser Arbeit vorgestellten Erstellungsmethode stark profitieren. Einerseits wurden SiAM-Metamodelle sinnvoll erweitert, um Modellierungslücken im Rahmen eines modellgetriebenen Vorgehens zu schließen. Andererseits enthält das KomBInoS-Domänenmodell Repräsentationen übertragbarer Domänenkonzepte auf Basis von Standard-Ontologien. Diese in SiAM-dp zurückgespielt können als Ausgangspunkt z. B. für wiederverwendbare, domänenübergreifende Vorlagen für die Spracherkennung und -synthese dienen.

(6) **Prototypen und Demonstratoren in fachlichen Anwendungsdomänen:**

Im Rahmen der Arbeit sind acht Prototypen und Demonstratoren in den unter-

schiedlichsten fachlichen Anwendungsdomänen entstanden. So wurden mit Hilfe von KomBInoS mobile Assistenzdienste im Tourismusbereich, in der Schadenregulierung und -vermeidung im Versicherungswesen, für Smart Farming und Smart Retail implementiert und einer breiten Öffentlichkeit, u. a. im Rahmen der CeBIT 2011 und 2014, präsentiert. Außerdem wurde KomBInoS im Bereich Industrie 4.0 zur Erstellung von hallenbodennahen Assistenzdiensten sowie im Rahmen der Mensch-Roboter-Kollaboration eingesetzt.

11.2.4 Beiträge zur Smart Service Welt

Die Beiträge der vorliegenden Arbeit zur Smart Service Welt lassen sich den folgenden zur Realisierung von software-definierten Plattformen notwendigen Technologiebausteinen zuordnen.

- **Case-based Reasoning für automatisierte Dienstauswahl** (Acatech 2014, S. 163f.). Die Smart Service Welt sieht konkreten Forschungsbedarf bei der „*Analyse von verschiedenen CBR-Methoden für die automatisierte Diensteauswahl auf Software-definierten Plattformen*„ sowie die Notwendigkeit zur „*Entwicklung von domänenspezifischen und domänenübergreifenden Fall-Datenbanken*“. Durch die in KomBInoS realisierten Erweiterungen der jCOLIBRI-Plattform hinsichtlich fallbasiertem Schließen auf Ecore-basierten Metamodellen können diese Fallbasen in formale Modelle gefasst und beschleunigt entwickelt und angepasst werden, um auf dieser Grundlage eine strukturierte Analyse von CBR-Methoden durchzuführen.
- **Serviceorchestrierung** (Acatech 2014, S. 164f.). Die Orchestrierbarkeit von Diensten und Dienstleistungen zur Wertschöpfung ist von zentraler Bedeutung für die Smart Service Welt. Folglich werden „*Infrastruktur und Werkzeuge zur Unterstützung des Anbieters beim Erstellen einer Orchestrierung auf allen Ebenen der digitalen und physischen Dienstleistung, von der abstrakten Prozessebene bis zum Menschen als Erbringer einer physischen Dienstleistung*“ als konkreter Forschungsbedarf formuliert. KomBInoS betrachtet die Komposition von intelligenten Benutzerschnittstellen für Smart Services. Die ganzheitliche KomBInoS-Methodik stellt in diesem Kontext sowohl Infrastruktur, Werkzeuge als auch eine methodische Unterstützung zur Erstellung von (komponierten) Benutzerschnittstellen sowie deren Nutzung zur Verfügung.
- **Alerting Management und Kollaboration** (Acatech 2014, S. 188). Im Falle eines eintretenden Ereignisses ist es entscheidend, den richtigen Agenten über den passenden Kanal in möglichst kurzer Zeit darüber in Kenntnis zu setzen. Die KomBInoS-Laufzeitumgebung ist hierzu in der Lage und kann somit als generischer Technologiebaustein für den Bereich *Alerting Management und Kollaboration* angesehen werden. Insgesamt leistet KomBInoS einen wichtigen Beitrag zum formulierten konkreten Forschungsbedarf „*HMI-Entwicklung für die barrierefreie Bedienbarkeit*“.

11.2.5 Publikationen

Die wissenschaftlichen und praktischen Beiträge dieser Arbeit wurden im Rahmen diverser Veröffentlichungen publiziert.

Buchkapitel

- D. Porta, M. Deru, S. Bergweiler, G. Herzog und P. Poller: *Building Multimodal Dialog User Interfaces in the Context of the Internet of Services.* In Wolfgang Wahlster, Hans-Joachim Grallert, Stefan Wess, Hermann Friedrich, und Thomas Widenka (Hrsg.), Towards the Internet of Services: The THESEUS Research Program, 145-162, Springer, Berlin, 2014.
- D. Sonntag und D. Porta: *Intelligent Semantic Mediation, Knowledge Acquisition and User Interaction.* In Wolfgang Wahlster, Hans-Joachim Grallert, Stefan Wess, Hermann Friedrich, und Thomas Widenka (Hrsg.), Towards the Internet of Services: The THESEUS Research Program, 179–189, Springer, Berlin, 2014.

Konferenzbeiträge

- D. Porta: *A Novel, Community-enabled Mobile Information System for Hikers.* Proceedings of the 2nd International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM-08), 438-444, IEEE Computer Society Press, 2008.
- D. Porta und J. Conrad: *UBIGIouS: A Ubiquitous, Mixed-Reality Geographic Information System.* Proceedings of the 13th International Conference on Intelligent User Interfaces (IUI), 393–396, ACM, New York, NY, USA, 2008.
- D. Porta, D. Sonntag und R. Neßelrath: *New Business to Business Interaction: Shake your iPhone and speak to it.* Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI), ACM, New York, NY, USA, 2009.
- S. Bergweiler, M. Deru und D. Porta: *Integrating a multitouch kiosk system with mobile devices and multimodal interaction.* ACM International Conference on Interactive Tabletops and Surfaces (ITS), 245–246, ACM, New York, NY, USA, 2010.
- D. Sonntag, D. Porta und J. Setz: *HTTP/REST-based Meta Web Services in Mobile Application Frameworks.* Proceedings of the 4nd International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM-10), 170-175, IARIA/XPS (Xpert Publishing Services), 2010.
- D. Porta, Z. Tuncer, M. Wirth und M. Hellenschmidt: *Multimodal Task Assignment and Introspection in Distributed Agricultural Harvesting Processes.* Proceedings of the Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM-14), Rome, Italy, 227-232, 2014.
- Z. Tuncer, D. Porta, M. Wirth und M. Hellenschmidt: *Multimodal Coordination of an Agricultural Harvesting Process.* Proceedings of the 72th International

Conference Agricultural Engineering (LAND.TECHNIK.2014), 2014.

Workshopbeiträge

- D. Porta, D. Sonntag und R. Neßelrath: *A Multimodal Mobile B2B Dialogue Interface on the iPhone.* Proceedings of the 4th Workshop on Speech in Mobile and Pervasive Environments (SiMPE '09) in conjunction with MobileHCI '09, ACM, 2009.
- D. Porta: *Towards Model-driven Development of Mobile Multimodal User Interfaces for Services.* In Klaus-Peter Fähnrich, und Bogdan Franczyk (Hrsg.), Informatik 2010: Service Science - Neue Perspektiven für die Informatik, Beiträge der 40. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Band 1, (175)1:497-502, GI, 2010.
- R. Nesselrath und D. Porta. *Rapid Development of Multimodal Dialogue Applications with Semantic Models.* Proceedings of the 7th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems (KRPDS-11), 37-47, 2011.

Technischer Bericht

- D. Sonntag, C. Weihrauch, O. Jacobs, und D. Porta. *THESEUS CTC-WP4 Usability Guidelines for Use Case Applications.* DFKI Technical Report THESEUS, (1):96, 2010.

Patentanmeldung

- I. Bierwas, D. Dengler, D. Porta, R. Nesselrath, und S. Germesin. *Intelligent automated online transaction system for automated interaction with online transaction web sites.* EP Patent App. EP20130000558, 2014.

Betreute Abschlussarbeiten

- J. Eckhard. *Intuitive Visualization of Rich Structured Data on the iPhone Platform.* Bachelorarbeit, Universität des Saarlandes, 2009.
- J. Schottler. *Ein Framework zur multimodalen Interaktion mit Service-Mashups auf Basis modellbasierter UI-Transformationen.* Masterarbeit, Fachhochschule Trier, 2009.
- E. Denerz. *Statechart Eclipse: Ein grafischer Editor zur Entwicklung semantischer Statecharts.* Bachelorarbeit, Universität des Saarlandes, 2010.
- J. Offenberg. *IMS^2: Semantic Statechart Processes for Dialogue Managing.* Bachelorarbeit, Universität des Saarlandes, 2010.
- C. Jäckel. *Semi-automatische Komposition von Dienstbenutzerschnittstellen auf mehreren Abstraktionsebenen.* Bachelorarbeit, Universität des Saarlandes, 2013.
- E. Denerz. *A Formal Approach for Modelling and Implementing Agent-based and Privacy-aware Dialogue Management.* Masterarbeit, Universität des Saarlandes, 2013.
- P. Mennig. *Enriching Speech Recognition Grammars using Semantic Data – in*

the Context of Semantic Web. Masterarbeit, Universität des Saarlandes, 2013.

- A. Speicher. *Unterstützung der Erzeugung von UI-Modellen durch fallbasiertes Schließen.* Bachelorarbeit, Universität des Saarlandes, 2015.
- N. Jafarian. *Generating Industrie 4.0 Assistant Web Applications from CMMN: A Model-driven Approach.* Masterarbeit, Universität des Saarlandes, 2017.

11.3 Zukünftige Arbeiten und Ausblick

Wie bereits eingangs zusammengefasst, bestand das Ziel der Arbeit in der Entwicklung eines ganzheitlichen Ansatzes zur effizienten und konsistenten Erstellung von multimodalen Dialogschnittstellen für Smart Services. Um dieses Ziel zu erreichen, wurde eine umfassende Methodik zur modellgetriebenen Entwicklung und Komposition solcher Benutzerschnittstellen konzipiert und implementiert. Hierbei wurde die Arbeit thematisch durch die soeben beantworteten Forschungsfragen geleitet und anhand dieser vertieft. Zu guter Letzt soll die folgende Aufzählung weitere interessante Fragestellungen aufwerfen, die im Rahmen dieser Arbeit keine Berücksichtigung fanden, aber in zukünftigen Arbeiten kurz-, mittel- und langfristig dennoch auf unterschiedlichen Gebieten lohnenswerte Forschungsgegenstände sowie Erweiterungen und Anwendungen dieser Arbeit darstellen.

- **Regelmaschine für SiAM-Abbildungsregeln:**
  Wie in Kapitel 7.3.7 (S. 176) erläutert, erfolgt die Selektion von Ecore-basierten Abbildungsregeln in SiAM-dp nach der Strategie *First in First Served*. Mit einer verstärkten Nutzung solcher Regeln müssen intelligentere Algorithmen zur Selektion, z. B. Rete (Forgy 1982), und Konfliktresolution, wie etwa von Pfleger (2007, S. 155ff.) für erweiterte getypte Merkmalsstrukturen umgesetzt, implementiert werden. Dies kann kurzfristig erfolgen.
- **Überschreibschutz manueller Modellanpassungen bei wiederholter Modelltransformation:**
  Werden an einem Modell, welches durch eine Modelltransformation erzeugt wurde, manuelle Anpassungen vorgenommen, so werden diese bei einer erneuten Anwendung der Transformation verworfen, weil das Zielmodell auf Basis der Quellmodelle auf Dateiebene vollständig neu aufgebaut wird. Hier ist, wie in Kapitel 3 (S. 146) geschildert, kurz- bis mittelfristig ein Verfahren notwendig, welches nachträgliche manuelle Anpassungen an einem Modell identifizieren und diese anschließend in ein neu generiertes Modell übertragen kann.
- **Dialogmanagement:**
  Von Löckelt (2008) wurden einschlägige, auch hier anwendbare Vorarbeiten zu sogenannten Dialogspielen geleistet. Das notwendige Dialogmanagement wurde für Demonstratoren prototypisch unter Einsatz des von SiAM-dp bereitgestellten graphbasierten SiAM-Dialogmodells (Neßelrath 2016, S. 123ff.) möglichst gene-

risch umgesetzt. Der Einsatz flexiblerer Verfahren, wie in Kapitel 4.3.4 (S. 63) geschildert, kann sich jedoch vorteilhaft auf das Benutzererlebnis auswirken. Die Umsetzung eines entsprechenden Ansatzes zum Schablonenfüllen steht aktuell auf der Forschungs- und Entwicklungsagenda[1] der SiAM-Dialogplattform. Aufgrund der klar definierten Schnittstellen der KomBInoS-Plattformkomponenten ist der Austausch einer konkreten Implementierung, insbesondere des Dialogmanagers, leicht und kurzfristig möglich, nachdem die SiAM-Dialogplattform um komplexere Mechanismen zum Dialogmanagement ergänzt wurde. Anschließend bietet sich mittelfristig eine hervorragende technologische Grundlage zum empirischen Erforschen geeigneter Dialogmanagementstrategien für eine kontextsensitive multimodale Dialoginteraktion mit Smart Services.

- **Intelligente Chat- und Actionbots zur Automatisierung der Sachbearbeitung:**

  Im Rahmen einer Studie hat Oracle (2016, S. 5) herausgefunden, dass 80% der 800 befragten Unternehmen bis 2020 planen, Chatbots in der Kundenbetreuung einzusetzen. Solche Chatbots können in der Regel monomodale Text- oder Spracheingabe verarbeiten und eine entsprechende Ausgabe erzeugen. Kann ein Chatbot über die reine Fragenbeantwortung hinaus auch aktiv Prozesse anstoßen und vorantreiben, so spricht man von Actionbots. Sie ermöglichen einem Kunden im Kontext einer angebotenen Dienstleistung eine zeitunabhängige Selbstbedienung ohne Kontakt zu einem menschlichen Sachbearbeiter. Hierbei handelt es sich nicht um ein gänzlich neues aber ganz sicher um ein starkes Trendthema mit günstigen wirtschaftlichen Erfolgsaussichten. Aufgrund der Prozessorientiertheit kann KomBInoS hier mittelfristig sowohl methodisch als auch technologisch zur Implementierung von Chat- und Actionbots eingesetzt werden, ohne dass vorab Anpassungen an der Domänenarchitektur (Kapitel 2.4.1, S. 20) notwendig sind. Durch die Erreichbarkeit eines solchen Bots per Telefon und die natürlichsprachliche Umsetzung entsteht dennoch der Eindruck einer menschlichen Konversation. Aufgrund der verteilten Architektur der Laufzeitumgebung ist ebenso die Einbettung in digitale Assistenten, wie etwa Google Siri, Amazon Alexa, Microsoft Cortana oder Google Home, denkbar.
- **Engineering- und Laufzeitumgebung für adaptive, multimodale Werkerassistenzdienste:**

  Wie in Kapitel 10.7 (S. 245) gezeigt, wurde der hier vorgestellte Ansatz bereits erfolgreich im Industrie 4.0-Kontext im Rahmen des Projekts SmartF-IT zur Entwicklung sogenannten FactoryApps eingesetzt. Im Projekt BaSys 4.0[2] steht derzeit die Entwicklung einer offenen Softwareplattform zum Betrieb hoch wandelbarer Produktionsanlagen im Fokus. Im Bereich Montage impliziert eine solche Wandelbarkeit hinsichtlich der Variantenvielfalt, Betriebsmittel und Montageprozesse in Kombination mit einer Losgröße-1-Fertigung die Notwendigkeit einer ebenso gro-

[1] siehe http://madmacs.dfki.de/?page_id=436 (letzter Zugriff: 04.11.2017)
[2] siehe http://www.basys40.de/ (letzter Zugriff: 04.11.2017)

ßen und schnellen Wandelbarkeit von Werkerassistenzdiensten zur Montageanleitung und -überwachung. KomBInoS bietet hier eine ausgezeichnete Grundlage, um mittel- bis langfristig einen wandelbaren, multimodalen Werkerassistenzdienst zu entwickeln, der im Rahmen einer Anlagenrekonfiguration lediglich eine minimale Engineeringphase zur Er- und Bereitstellung von neuen Inhalten benötigt. Stehen solche Inhalte unter Verwendung des Konzepts der Verwaltungsschale (Plattform Industrie 4.0 2016) immer vorab zur Verfügung, so kann die Engineeringphase unter verstärkter Einbeziehung von nicht überwachten Kompositionsansätzen zur Laufzeit praktisch entfallen. Über eine Prozesssimulation im Rahmen der virtuellen Inbetriebnahme der gewandelten Gesamtanlage kann sowohl die inhaltliche als auch die ergonomische Qualität der Werkerassistenz dennoch sichergestellt werden. Solche Aspekte der Selbstadaptivität vor dem Hintergrund der Wandelbarkeit eines cyber-physischen Produktionssystems wurden im interaktiven (Montage-) Handbuch (Pfleger u. a. 2015) im Kontext des Projekts Cypros[3] nicht adressiert. Der Einsatz von KomBInoS in diesem Anwendungsgebiet erfordert vorab eine eingehende Analyse der Domäne zur Anpassung der Domänenarchitektur. Erste Arbeiten hierzu haben bereits begonnen.

[3] siehe http://www.projekt-cypros.de/ (letzter Zugriff: 04.11.2017)

Teil V

Anhang

ANHANG A

Überblick über die Implementierung des KomBInoS-Rahmenwerks

An dieser Stelle wird zentral ein Einblick über die Modularisierung von KomBInoS sowie ein quantitativer Überblick über die Modellierung und Implementierung gegeben. Tabelle A.1 schlüsselt entsprechende Metriken für die drei Gruppen *KomBInoS Modelle* (Kapitel 6, S. 111), *KomBInoS Werkbank* (Kapitel 8, S. 187) und *KomBInoS Laufzeit* (Kapitel 9, S. 211) auf. Zur automatisierten Erhebung dieser quantitativen Aussagen über die Codebasis wurde das Eclipse Metrics Plugin[1] verwendet. Die teilautomatisiert erstellten Metamodelle von CMMN und SSML werden in der Betrachtung der KomBInoS Modelle ebenso vernachlässigt wie die genutzten Metamodelle von SiAM-dp und BPMN. Abbildung A.1 zeigt die Bestandteile der Gruppen in Form von Eclipse-Projekten.

Tabelle A.1: Code- und Modellmetriken von KomBInoS

	Java-			Modell-	
Anzahl	Schnittstellen	Klassen	Codezeilen	Konzepte	Attribute
KomBInoS Modelle	317	749	113335	201	267
KomBInoS Werkbank	81	273	12059	92	104
KomBInoS Laufzeit	66	158	21515	42	204
Summe	464	1180	146909	335	575

[1] siehe https://sourceforge.net/projects/metrics/ (letzter Zugriff: 08.11.2017)

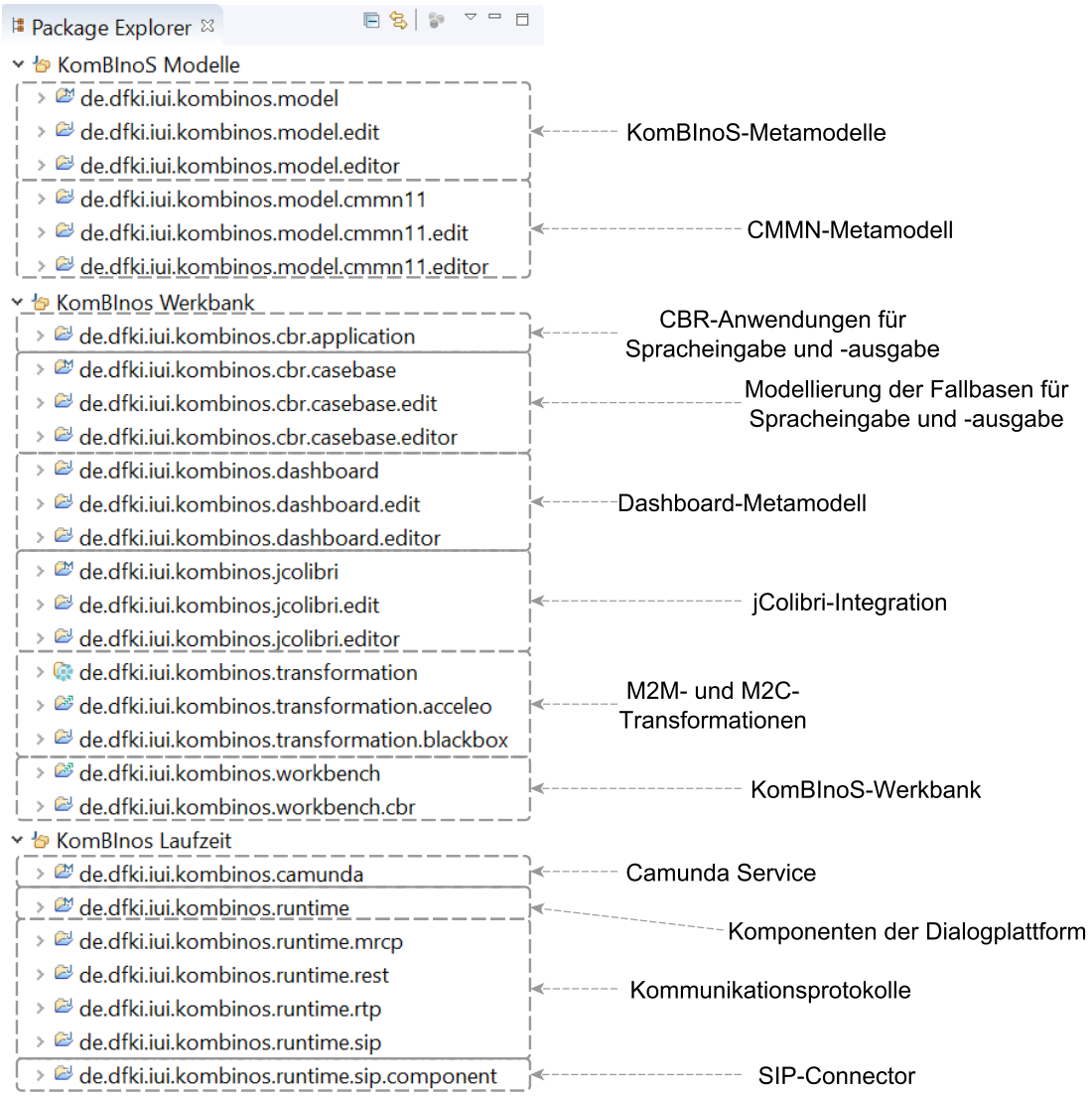


Abbildung A.1: Überblick über die Implementierung des KomBInoS-Rahmenwerks

ANHANG B

Unifikation und Overlay auf Ecore-basierten Metamodellen

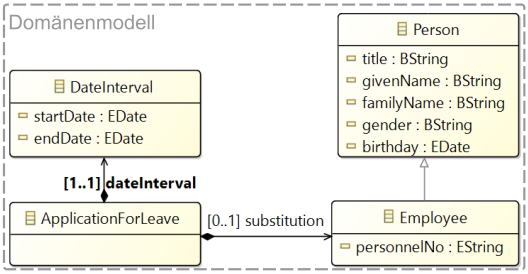
An dieser Stelle wird ein kurzer Überblick über die Unifikation und Overlay auf Ecore-basierten Metamodellen anhand von Pflegers (2007, S. 127ff.) und Neßelraths (2016, S. 89ff.) Erläuterungen gegeben. Auf Basis des Ecore Meta-Metamodells wurde in SiAM-dp eine API implementiert, die gebräuchliche Algorithmen erweiterter getypter Merkmalstrukturen für die Wissensverarbeitung in multimodalen Dialogsystemen, wie sie im Rahmen von ODP verwendet wurden, im Rahmen des Eclipse Modeling Framework für Ecore-basierte Metamodelle und deren Instanzen verfügbar macht.

Gegeben sei eine Ecore-basierte Vererbungshierarchie in Form eines Metamodells. Dann beschreibt die **Unifikation** eine kommutative Operation, die für zwei Instanzobjekte die größte untere Schranke dieser Objekte als spezifischstes Objekt zurückliefert, welches die Information beider Eingabeobjekte subsumiert. Die Unifikation kann fehlschlagen, wenn keine größte untere Schranke existiert, etwa bei gegensätzlichen Informationen oder falls kein gemeinsames Elternkonzept existiert, welches die Information beider Instanzen aufnehmen kann. Abbildung B.1 zeigt unter Punkt (1) ein Beispiel.

Zum instanzbasierten Mustervergleich steht mit der **eingeschränkten Unifikation** eine nichtkommutative Variante der Unifikation zur Verfügung, welche prüft, ob ein Instanzobjekt mindestens die Information enthält, die ein anderes, als Muster fungierendes Instanzobjekt enthält. Die Rückgabe dieser Operation ist ein Wahrheitswert. Abbildung B.1 zeigt unter Punkt (2) ein Beispiel. Da diese Art der Musterspezifikation sehr eingeschränkt ist, wurde in SiAM-dp mit dem Pattern-Metamodell (Kapitel C, S. 273) ein ausdrucksstärkerer Mechanismus entwickelt.

Overlay (Alexandersson und Becker 2001, 2003) beschreibt ebenfalls eine nichtkommutative Operation, welche jedoch nicht fehlschlägt. Die Operation dient zum Anreichern einer als *Covering* bezeichneten Instanz mit Information einer zweiten, nicht

notwendigerweise unifizierbaren Instanz genannt *Background*. Overlay kann im übertragenen Sinne als Prägeoperation aufgefasst werden, bei der das Covering-Objekt als Matrize fungiert. Overlay verfügt über einen Scoring-Mechanismus, der ein semantisches Kompatibilitäts- oder Ähnlichkeitsmaß zwischen Covering und Background berechnet. So kann bei zwei Overlay-Anwendungen mit unterschiedlichem Covering aber gleichem Background bestimmt werden, welche Anwendung besser war. Dies dient u. a. zur Diskursauflösung von Referenzen und wird in KomBInoS auch im Rahmen des fallbasierten Schließens (Kapitel 8.8, S. 199) eingesetzt. Abbildung B.1 zeigt unter Punkt (3) ein Beispiel.



$$
\begin{aligned}
&(1)\ \text{unify}\left(\begin{bmatrix}\textsc{Employee}\\ \text{givenName} & \textit{Max}\\ \text{familiyName} & \textit{Mustermann}\\ \text{personellNo} & \textit{0815}\end{bmatrix},\ \begin{bmatrix}\textsc{Employee}\\ \text{givenName} & \textit{Max}\\ \text{familiyName} & \textit{Mustermann}\\ \text{birthday} & \textit{01.02.1976}\end{bmatrix}\right)\\
&\qquad = \begin{bmatrix}\textsc{Employee}\\ \text{givenName} & \textit{Max}\\ \text{familiyName} & \textit{Mustermann}\\ \text{personellNo} & \textit{0815}\\ \text{birthday} & \textit{01.02.1976}\end{bmatrix}\\
&(2)\ \text{runify}\left(\begin{bmatrix}\textsc{Employee}\\ \text{givenName} & \textit{Max}\\ \text{familiyName} & \textit{Mustermann}\\ \text{personellNo} & \textit{0815}\end{bmatrix},\ \begin{bmatrix}\textsc{Employee}\\ \text{givenName} & \textit{Max}\\ \text{familiyName} & \textit{Mustermann}\\ \text{birthday} & \textit{01.02.1976}\end{bmatrix}\right) = \bot\\
&(3)\ \text{overlay}\left(\begin{bmatrix}\textsc{Employee}\\ \text{givenName} & \textit{Max}\\ \text{familiyName} & \textit{Mustermann}\\ \text{personellNo} & \textit{0815}\end{bmatrix},\ \begin{bmatrix}\textsc{Person}\\ \text{givenName} & \textit{Erika}\\ \text{familiyName} & \textit{Musterfrau}\\ \text{birthday} & \textit{01.02.1976}\end{bmatrix}\right)\\
&\qquad = \begin{bmatrix}\textsc{Employee}\\ \text{givenName} & \textit{Max}\\ \text{familiyName} & \textit{Mustermann}\\ \text{personellNo} & \textit{0815}\\ \text{birthday} & \textit{01.02.1976}\end{bmatrix}
\end{aligned}
$$

Abbildung B.1: Beispiel - (1) Unifikation, (2) eingeschränkte Unifikation und (3) Overlay angewendet auf zwei Instanzobjekte

ANHANG C

Das Pattern-Metamodell

Das Pattern-Metamodell der SiAM-Dialogplattform erlaubt die Spezifikation feingranularer Muster von Ecore-basierten Domänenobjekten. Das Metamodell wurde initial im Rahmen von KomBInoS durch Denerz (2013) entwickelt und anschließend in SiAM-dp weiterentwickelt. In dieser weiterentwickelten Version kommt das Metamodell in Verbindung mit den darauf operierenden Algorithmen zum Mustervergleich unverändert in KomBInoS zum Einsatz, insbesondere zur Spezifikation von nicht aufgelösten semantischen Inhalten sowie für den Bedingungsteil von Abbildungsregeln im Rahmen der in Kapitel 7.3 (S. 147) beschriebenen Modelltransformationen. Der Vollständigkeit halber wird das Metamodell hier auf Basis von Neßelrath (2016, S. 96ff.) beschrieben.

Das Pattern-Metamodell ist in Abbildung C.1 dargestellt. Das Muster eines komplexen Domänenobjekts (PObject) referenziert das entsprechende Domänenkonzept eines Ecore-basierten Metamodells (EClass). Weiterhin enthält das Muster eine Menge von Slots (PSlot), welche Strukturelemente (EStructuralFeature) des Domänenkonzepts in Form von Attributen und Referenzen näher beschreibt. Dies geschieht durch Angabe eines Wertebereichs. Bezieht sich das Strukturelement des Slots auf eine Referenz, so wird der gültige Wertebereich rekursiv durch ein weiteres komplexes Muster (PObject) beschrieben. Referenziert der Slot ein Attribut des Domänenkonzepts, so kann entweder verlangt werden, dass der Wert des Attributs nicht definiert, also leer, ist (PEmptySlot). Alternativ kann der Wertebereich durch eine Menge von Restriktionen (PRestrictions) feingranular eingeschränkt und logisch miteinander verknüpft werden (PQuantorEnum, PRestrictionsEnum). In Abhängigkeit vom Datentyp des Attributs des Domänenkonzepts können dedizierte Vergleichsoperatoren für den jeweiligen Datentyp in einer spezialisierten Variante der abstrakten Restriktion (PRestriction) zur Anwendung kommen. Der Wertebereich eines Attributs vom Typ String (PStringRestriction) lässt sich etwa durch Angabe einer zu findenden Zeichenkette (*value*) in Kombination mit einer üblichen Vergleichsfunktion (z. B. PStringRestrictionEnum.*CONTAINS*) einschränken.

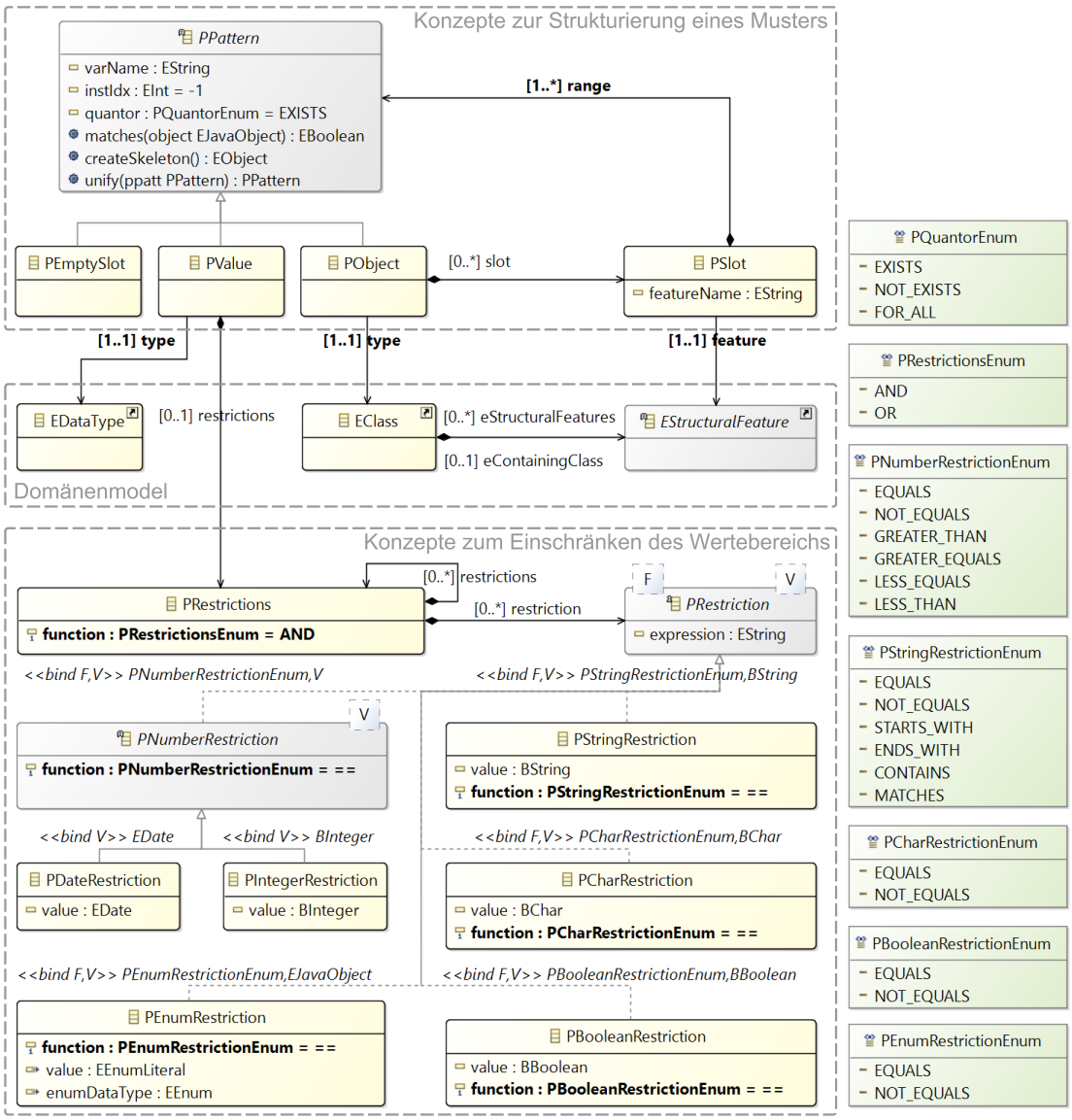


Abbildung C.1: Das SiAM-Metamodell zur Musterspezifikation

Der Abgleich eines Musters mit einem instanziierten Domänenobjekt berücksichtigt die taxonomische Ordnung der verwendeten Domänenkonzepte und prüft rekursiv, ob alle durch Quantoren und Restriktionen ausgedrückten Bedingungen aller Teilmuster durch die im Fokus stehende Instanz erfüllt werden. Die Funktionalität zum Musterabgleich steht durch die Methode *matches()* allen Mustern (PPattern) zur Verfügung. Ebenso existieren Methoden zum unifikationsbasierten Verschmelzen von Mustern (*unify()*) sowie zum Erzeugen einer gültigen Instanz auf Basis eines Musters (*createSkeleton()*).

Abbildung C.2 illustriert ein Beispiel auf Grundlage des in Abbildung 7.4 gezeigten Domänenmodells.

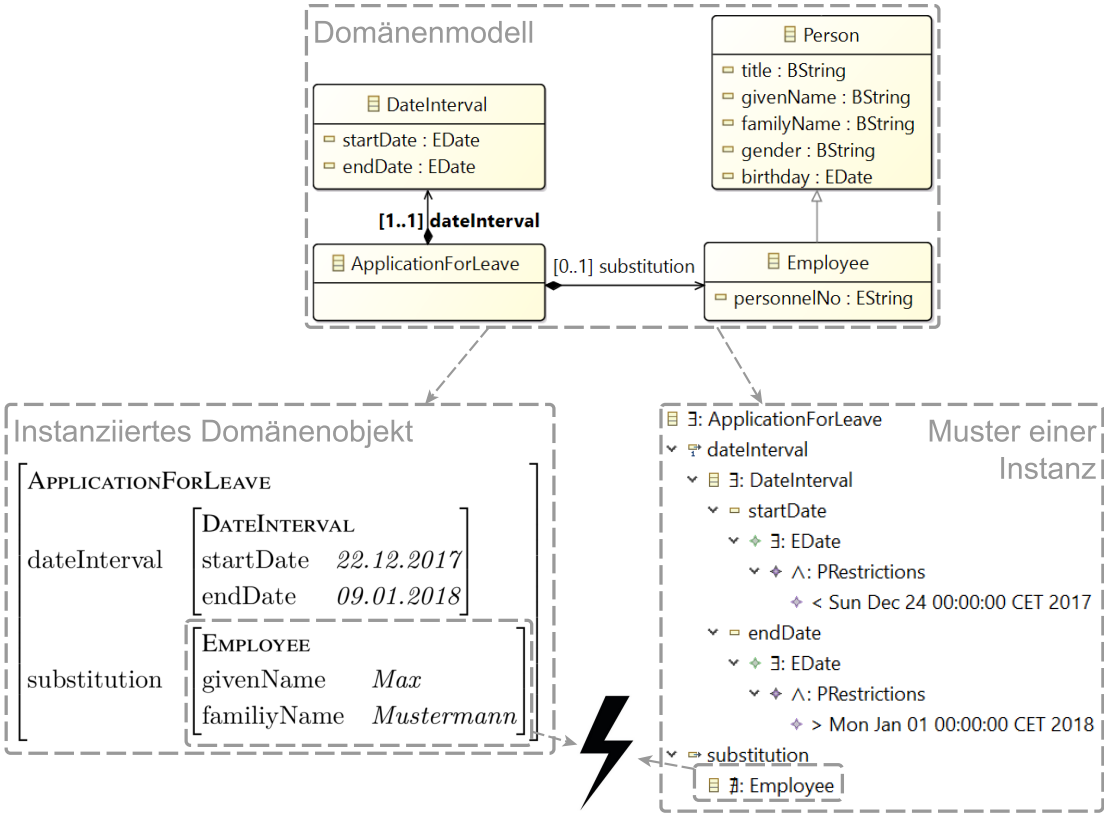


Abbildung C.2: Beispiel - Musterabgleich

Links dargestellt ist eine Instanz des Domänenkonzepts Urlaubsantrag (ApplicationForLeave). Der gewünschte Zeitraum erstreckt sich vom 22.12.2017 bis zum 09.01.2018. Als Ansprechpartner soll Max Mustermann fungieren. Rechts dargestellt ist ein Muster des selben Domänenkonzepts. Dieses Muster formuliert eine Anfrage an Instanzen, die umgangssprachlich folgendermaßen geäußert werden kann: *gibt mir alle Urlaubsanträge, die sich mindestens über Weihnachten und Neujahr erstecken und in denen kein (∄) Ansprechpartner benannt wurde.* Während die erste Bedingung (der Zeitraum) noch auf die links gezeigte Instanz zutrifft, scheitert der Musterabgleich an der zweiten Bedingung, dem vorhandenen Ansprechpartner.

Akronyme

API	Programmierschnittstelle (Application Programming Interface)
AUI	abstraktes UI
BPMN	Business Process Model and Notation
CBR	fallbasiertes Schließen (Case-Based Reasoning)
CDI	Contexts and Dependency Injection
CMMN	Case Management Model and Notation
CRF	Cameleon Referenzrahmenwerk (Cameleon Reference Framework)
CUI	konkretes UI
DISL	Dialog and Interface Specification Language
EMF	Eclipse Modeling Framework
EPK	ereignisgesteuerte Prozesskette
GUI	grafische Benutzerschnittstelle
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	integrierte Entwicklungsumgebung (Integrated Development Environment)
ISE	Integrated Service Engineering
JSON	JavaScript Object Notation
M2C	Modell-zu-Code (model-to-code)
M2M	Modell-zu-Modell (model-to-model)
MDA	Modellgetriebene Architektur (Model-driven Architecture)
MDSE	modellgetriebene Softwareentwicklung (Model-driven Software Engineering)
MOF	Meta Object Facility

MQTT	Message Queue Telemetry Transport
MRCP	Media Resource Control Protocol
MVC	Model-View-Controller
MVVM	Model-View-ViewModel
OCL	Object Constrained Language
ODP	Ontologiebasierte Dialogplattform
OSGi	Open Service Gateway Initiative
OWL	Web Ontology Language
PoSR	Potsdam Services Registry
QVT	Query View Transformation
RDF	Resource Description Framework
REST	Representational State Transfer
RTP	Realtime Transport Protocol
SDP	Session Description Protocol
SiAM-dp	Situationsadaptive Multimodale Dialogplattform
SIP	Session Initiation Protocol
SOA	dienstorientierte Architektur (Service-oriented Architecture)
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SRGS	Speech Recognition Grammar Specification
SSML	Speech Synthesis Markup Language
UDP	User Datagram Protocol
UI	Benutzerschnittstelle (User Interface)
UIML	User Interface Markup Language
UML	Unified Modeling Language
URC	Universal Remote Console
URI	Uniform Resource Identifier
USDL	Unified Service Description Language
UsiXML	User Interface Extensible Markup Language
W3C	World Wide Web Consortium
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Service Description Language
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

Literaturverzeichnis

(van der Aalst und ter Hofstede 2012) Aalst, W. van der ; Hofstede, A. ter: Workflow patterns put into context. In: **Software & Systems Modeling** 11 (2012), Nr. 3, S. 319–323

(Aamodt und Plaza 1994) Aamodt, A. ; Plaza, E.: Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. In: **AI Communications** 7 (1994), März, Nr. 1, S. 39–59

(Abrams und Schulte 2008) Abrams, C. ; Schulte, W. R.: Service-Oriented Architecture Overview and Guide to SOA Research / Gartner, January 2008 (G00154463). – Forschungsbericht

(AbuJarour u. a. 2009) AbuJarour, M. ; Craculeac, M. ; Menge, F. ; Vogel, T. ; Schwarz, J. F.: Posr: A Comprehensive System for Aggregating and Using Web Services. In: **Proceedings of the 2009 Congress on Services - I**. Washington, DC, USA : IEEE Computer Society, 2009, S. 139–146

(Acatech 2014) Acatech ; Kagermann, H. ; Riemensperger, F. ; Hoke, D. ; Helbig, J. ; Stocksmeier, D. ; Wahlster, W. ; Scheer, A.-W. ; Schweer, D. (Eds.): **Smart Service Welt: Umsetzungsempfehlungen für das Zukunftsprojekt Internetbasierte Dienste für die Wirtschaft**. März 2014

(Acatech 2015) Acatech ; Kagermann, H. ; Riemensperger, F. ; Hoke, D. ; Schuh, G. ; Scheer, A.-W. ; Spath, D. ; Leukert, B. ; Wahlster, W. ; Rohleder, B. ; Schweer, D. (Eds.): **Smart Service Welt: Umsetzungsempfehlungen für das Zukunftsprojekt Internetbasierte Dienste für die Wirtschaft - Abschlussbericht Langversion**. März 2015

(Alexandersson und Becker 2001) Alexandersson, J. ; Becker, T.: Overlay as the Basic Operation for Discourse Processing in a Multimodal Dialogue System. In: **Proceedings of the IJCAI-01 Workshop on Knowledge and Reasoning in Practical Dialogue Systems**, 2001, S. 1–7

(Alexandersson und Becker 2003) Alexandersson, J. ; Becker, T.: The Formal Foundations Underlying Overlay. In: **Proceedings of the Fifth International**

Workshop on Computational Semantics (IWCS-5). Tilburg, The Netherlands, 2003, S. 1–14

(Alexandersson u. a. 2011) Alexandersson, J. ; Bund, J. ; Carrasco, E. ; Epelde, G. ; Klíma, M. ; Urdaneta, E. ; Vanderheiden, G. ; Zimmermann, G. ; Zinnikus, I.: openURC: Standardisation towards "User Interfaces for Everyone, Everywhere, on Anything". In: Wichert, R. ; Eberhardt, B. (Eds.): **Ambient Assisted Living: 4. AAL-Kongress 2011 Berlin, Germany**. Berlin : Springer, 2011, S. 117–125

(Alexandersson u. a. 1998) Alexandersson, J. ; Buschbeck-Wolf, B. ; Fujinami, T. ; Kipp, M. ; Koch, S. ; Maier, E. ; Reithinger, N. ; Schmitz, B. ; Siegel, M.: Dialogue Acts in VERBMOBIL-2 : Second Edition. In: **Verbmobil Report** 1998 (1998), Nr. 226

(Artstein u. a. 2011) Artstein, R. ; Rushforth, M. ; Gandhe, S. ; Traum, D. ; Donigian, A.: Limits of Simple Dialogue Acts for Tactical Questioning Dialogues. In: **Proceedings of 7th IJCAI workshop on Knowledge and Reasoning in Practical Dialogue Systems**, 2011, S. 1–7

(aruba Networks 2014) aruba Networks: **ARE YOU READY FOR GenMobile? - How a new group is changing the way we work, live and communicate**. Januar 2014. – URL http://community.arubanetworks.com/t5/Technology-Blog/GenMobile-Has-Entered-the-Workforce/ba-p/134323. – (letzter Zugriff: 04.11.2017)

(Atkinson und Kühne 2002) Atkinson, C. ; Kühne, T.: Rearchitecting the UML infrastructure. In: **ACM Transactions on Modeling and Computer Simulation (TOMACS)** 12 (2002), October, S. 290–321

(Atkinson und Kühne 2003) Atkinson, C. ; Kühne, T.: Model-Driven Development: A Metamodeling Foundation. In: **IEEE Software** 20 (2003), September, Nr. 5, S. 36–41

(Austin 1962) Austin, J. L.: **How to do things with words**. Oxford University Press, 1962 (William James Lectures)

(Axelsson u. a. 2001) Axelsson, J. ; Cross, C. ; Lie, H. W. ; McCobb, G. ; Raman, T. V. ; Wilson, L.: XHTML+Voice Profile 1.0 - W3C Note 21 December 2001 / World Wide Web Consortium (W3C), Dezember 2001. – Forschungsbericht

(Babitski u. a. 2011) Babitski, G. ; Bergweiler, S. ; Grebner, O. ; Oberle, D. ; Paulheim, H. ; Probst, F.: SoKNOS – Using Semantic Technologies in Disaster Management Software. In: Antoniou, G. ; Grobelnik, M. ; Simperl, E. ; Parsia, B. ; Plexousakis, D. ; De Leenheer, P. ; Pan, J. (Eds.): **The Semanic Web: Research and Applications** Bd. 6644. Springer Berlin Heidelberg, 2011, S. 183–197

(Balbo u. a. 2004) Balbo, S. ; Ozkan, N. ; Paris, C.: Choosing the Right Task Modelling Notation: A Taxonomy. In: **The Handbook of Task Analysis for Human-Computer Interaction**. Lawrence Erlbaum Associates, 2004, S. 445–465

(Barnett u. a. 2015) Barnett, J. ; Akolkar, R. ; Auburn, R. ; Bodell, M. ; Burnett, D. C. ; Carter, J. ; McGlashan, S. ; Lager, T. ; Helbing, M. ; Hosn, R. ; Raman, T. ; Reifenrath, K. ; Rosenthal, N. ; Roxendal, J.: **State Chart XML (SCXML): State Machine Notation for Control Abstraction - W3C Proposed Recommendation 30 April 2015**. April 2015

(Barros und Oberle 2012) Barros, A. ; Oberle, D.: **Handbook of Service Description: USDL and Its Methods**. New York : Springer, 2012

(Bean 2009) Bean, J.: **SOA and Web Services Interface Design: Principles, Techniques, and Standards**. San Francisco, Calif.; Oxford : Morgan Kaufmann; Elsevier Science, 2009

(Becker u. a. 2014) Becker, T. ; Burghart, C. ; Nazemi, K. ; Ndjiki-Nya, P. ; Riegel, T. ; Schäfer, R. ; Sporer, T. ; Tresp, V. ; Wissmann, J.: Core Technologies for the Internet of Services. In: (Wahlster u. a. 2014), S. 59–88

(Behring u. a. 2007) Behring, A. ; Heinrich, M. ; Winkler, M. ; Dargie, W.: Werkzeugunterstützte Modellierung multimodaler, adaptiver Benutzerschnittstellen. In: **i-com - Zeitschrift für interaktive und kooperative Medien** 6 (2007), Dezember, Nr. 3, S. 31–36

(Bergweiler 2014) Bergweiler, S.: Interactive Service Composition and Query. In: (Wahlster u. a. 2014), S. 163–178

(Beuvens und Vanderdonckt 2012) Beuvens, F. ; Vanderdonckt, J.: UsiGesture: An environment for integrating pen-based interaction in user interface development. In: **Sixth International Conference on Research Challenges in Information Science (RCIS)**, IEEE, Mai 2012, S. 1–12

(Beuvens und Vanderdonckt 2014) Beuvens, F. ; Vanderdonckt, J.: UsiGesture: Test and Evaluation of an Environment for Integrating Gestures in User Interfaces. In: **Revista Română de Interacţiune Om-Calculator** 7 (2014), Nr. 2, S. 139–160

(Bierwas u. a. 2014) Bierwas, I. ; Dengler, D. ; Porta, D. ; Nesselrath, R. ; Germesin, S.: **Intelligent automated online transaction system for automated interaction with online transaction web sites**. EP Patent App. EP20130000558. August 2014

(Block u. a. 2004) Block, H. U. ; Caspari, R. ; Schachtl, S.: Callable Manuals - Access to Product Documentation via Voice (Anrufbare Bedienungsanleitungen -

Zugang zu Produktdokumentation über Sprache). In: **it - Information Technology** 46 (2004), Nr. 6, S. 299–305

(BMWi 2016) BMWi: **Digitale Strategie 2025**. 2016. – URL https://www.bmwi.de/Redaktion/DE/Publikationen/Digitale-Welt/digitale-strategie-2025.html. – (letzter Zugriff: 04.11.2017)

(Bohus und Rudnicky 2009) Bohus, D. ; Rudnicky, A. I.: The RavenClaw dialog management framework: Architecture and systems. In: **Comput. Speech Lang.** 23 (2009), Juli, Nr. 3, S. 332–361

(Booth u. a. 2004) Booth, D. ; Haas, H. ; Brown, A.: Web Services Glossary - W3C Working Group Note 11 February 2004 / World Wide Web Consortium (W3C), February 2004. – Forschungsbericht

(Boyer 2009) Boyer, J. M.: **XForms 1.1 - W3C Recommendation 20 October 2009**. October 2009

(Brennan 1998) Brennan, S. E.: The Grounding Problem in Conversations With and Through Computers. In: Fussell, S. R. ; Kreuz, R. J. (Eds.): **Social and Cognitive Approaches to Interpersonal Communication**. Hillsdale, NJ : Lawrence Erlbaum, 1998, S. 201–225

(Brickley und Guha 2014) Brickley, D. ; Guha, R.: **RDF Schema 1.1 - W3C Recommendation 25 February 2014**. Februar 2014

(Britz u. a. 2016) Britz, J. ; Alexandersson, J. ; Stephan, W.: **UCH Goes EAL4—The Foundation of an Eco System for Ambient Assisted Living: ISO/IEC 15408 Common Criteria Based Implementation of the ISO/IEC 24752 Universal Control Hub Middleware**. S. 83–96. In: Wichert, R. ; Klausing, H. (Eds.): **Ambient Assisted Living: 8. AAL-Kongress 2015,Frankfurt/M, April 29-30. April, 2015**. Cham : Springer International Publishing, 2016

(Brown u. a. 2005) Brown, A. ; Conallen, J. ; Tropeano, D.: Introduction: Models, Modeling, and Model-Driven Architecture (MDA). In: Beydeda, S. ; Book, M. ; Gruhn, V. (Eds.): **Model-Driven Software Development**. Berlin/Heidelberg : Springer-Verlag, 2005, Kap. 1, S. 1–16

(Bui 2006) Bui, T. H.: Multimodal Dialogue Management - State of the Art / Centre for Telematics and Information Technology, University of Twente. Enschede, 2006 (TR-CTIT-06-01). – Forschungsbericht

(Bußmann 2002) Bussmann, H.: **Lexikon der Sprachwissenschaft**. Dritte. Stuttgart : Alfred Kröner Verlag, 2002

(Bundesregierung 2014) Bundesregierung: **Die neue Hichtech-Strategie - Innovationen für Deutschland**. 2014. – URL https://www.hightech-strategie.de/. – (letzter Zugriff: 04.11.2017)

(Bunt 1994) Bunt, H.: Context and Dialogue Control. In: **THINK Quarterly** 3 (1994)

(Bunt 2011) Bunt, H.: The semantics of dialogue acts. In: **Proceedings of the Ninth International Conference on Computational Semantics**. Stroudsburg, PA, USA : Association for Computational Linguistics, 2011 (IWCS '11), S. 1–13

(Bunt u. a. 2010) Bunt, H. ; Alexandersson, J. ; Carletta, J. ; Choe, J.-W. ; Fang, A. C. ; Hasida, K. ; Lee, K. ; Petukhova, V. ; Popescu-Belis, A. ; Romary, L. ; Soria, C. ; Traum, D.: Towards an ISO standard for dialogue act annotation. In: Calzolari, N. ; Choukri, K. ; Maegaard, B. ; Mariani, J. ; Odijk, J. ; Piperidis, S. ; Rosner, M. ; Tapias, D. (Eds.): **Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)**. Valletta, Malta : European Language Resources Association (ELRA), Mai 2010

(Burnett und Shanmugham 2012) Burnett, D. ; Shanmugham, S.: **Media Resource Control Protocol Version 2 (MRCPv2)**. November 2012

(Burnett u. a. 2004) Burnett, D. C. ; Walker, M. R. ; Hunt, A.: **Speech Synthesis Markup Language (SSML) Version 1.0 - W3C Recommendation 7 September 2004**. September 2004

(Calvary u. a. 2003) Calvary, G. ; Coutaz, J. ; Thevenin, D. ; Limbourg, Q. ; Bouillon, L. ; Vanderdonckt, J.: A Unifying Reference Framework for multi-target user interfaces. In: **Interacting with Computers** 15 (2003), Juni, Nr. 3, S. 289–308

(Campbell u. a. 2002) Campbell, B. ; Rosenberg, J. ; Schulzrinne, H. ; Huitema, C. ; Gurle, D.: **Session Initiation Protocol (SIP) Extension for Instant Messaging**. December 2002

(Cardoso u. a. 2009) Cardoso, J. ; Voigt, K. ; Winkler, M.: Service Engineering for the Internet of Services. In: Aalst, W. ; Mylopoulos, J. ; Rosemann, M. ; Shaw, M. J. ; Szyperski, C. ; Filipe, J. ; Cordeiro, J. (Eds.): **Enterprise Information Systems** Bd. 19. Berlin/Heidelberg : Springer-Verlag, 2009, Kap. 2, S. 15–27

(Carpenter 1992) Carpenter, B.: **Cambridge Tracts in Theoretical Computer Science**. Bd. 32: **The Logic of Typed Feature Structures**. Cambridge, MA : Cambridge University Press, Oktober 1992

(CEA 2008) CEA: **CEA-2018 (ANSI)**. März 2008

(Chinnici u. a. 2007) Chinnici, R. ; Moreau, J.-J. ; Ryman, A. ; Weerawarana, S.: **Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language**. June 2007

(Clark 1996) Clark, H. H.: **Using Language**. Cambridge, UK : Cambridge University Press, 1996

(Clark und Schaefer 1989) Clark, H. H. ; Schaefer, E. F.: Contributing to Discourse. In: **Cognitive Science** 13 (1989), Nr. 2, S. 259–294

(Claus und Schwill 1993) Claus, V. ; Schwill, A. ; Engesser, H. (Eds.): **Duden Informatik: ein Sachlexikon für Studium und Praxis**. Mannheim : Dudenverlag, 1993

(Constantine 2003) Constantine, L.: Canonical Abstract Prototypes for Abstract Visual and Interaction Design. In: Jorge, J. ; Jardim Nunes, N. ; Cunha, J. Falcão e (Eds.): **Interactive Systems. Design, Specification, and Verification** Bd. 2844. Berlin/Heidelberg : Springer-Verlag, 2003, S. 1–15

(Coutaz u. a. 1995) Coutaz, J. ; Nigay, L. ; Salber, D. ; Blandford, A. ; May, J. ; Young, R. M.: Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. In: **InterAct** Bd. 95, 1995, S. 115–120

(Cyganiak u. a. 2014) Cyganiak, R. ; Wood, D. ; Lanthaler, M.: **RDF 1.1 Concepts and Abstract Syntax**. Februar 2014

(Daconta u. a. 2003) Daconta, M. C. ; Obrst, L. J. ; Smith, K. T.: **The Semantic Web: A Guide to the Future of XML, Web Services and Knowledge Management**. Indianapolis, IN : Wiley, 2003

(Daniel u. a. 2009) Daniel, F. ; Casati, F. ; Benatallah, B. ; Shan, M.-C.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: **Proceedings of the 28th International Conference on Conceptual Modeling**. Berlin, Heidelberg : Springer-Verlag, 2009 (ER '09), S. 428–443

(Daniel und Matera 2014) Daniel, F. ; Matera, M.: **Mashups: Concepts, Models and Architectures**. Heidelberg : Springer, 2014 (Data-Centric Systems and Applications)

(Daniel u. a. 2007) Daniel, F. ; Matera, M. ; Yu, J. ; Benatallah, B. ; Saint-Paul, R. ; Casati, F.: Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. In: **Internet Computing, IEEE** 11 (2007), Nr. 3, S. 59 – 66

(Delgado und Araki 2005) Delgado, R. L. C. ; Araki, M.: **Spoken, Multilingual and Multimodal Dialogue Systems: Development and Assessment**. Chichester, England; Hoboken, NJ : John Wiley, 2005

(Denerz 2013) Denerz, E.: **A Formal Approach for Modelling and Implementing Agent-based and Privacy-aware Dialogue Management**, Universität des Saarlandes, Masterarbeit, 2013

(Dijkman und Gorp 2011) Dijkman, R. ; Gorp, P.: BPMN 2.0 Execution Semantics Formalized as Graph Rewrite Rules. In: Mendling, J. ; Weidlich, M. ; Weske, M. (Eds.): **Business Process Modeling Notation** Bd. 67. Springer Berlin Heidelberg, 2011, S. 16–30

(Do 2012) Do, H. H.: **Schema Matching and Mapping-based Data Integration: Architecture, Approaches and Evaluation**. Saarbrücken : AV Akademikerverlag, 2012

(Dufft u. a. 2010) Dufft, N. ; Schleife, K. ; Bertschek, I. ; Vanberg, M. ; Böhmann, T. ; Schmitt, A. K. ; Barnreiter, M.: Das wirtschaftliche Potenzial des Internet der Dienste. (2010), November

(Dumas u. a. 2009) Dumas, B. ; Lalanne, D. ; Oviatt, S.: Multimodal Interfaces: A Survey of Principles, Models and Frameworks. In: Lalanne, D. ; Kohlas, J. (Eds.): **Human Machine Interaction** Bd. 5440. Springer Berlin Heidelberg, 2009, S. 3–26

(Endres-Niggemeyer 2013) Endres-Niggemeyer, B.: **Semantic Mashups - Intelligent Reuse of Web Resources**. Berlin Heidelberg : Springer-Verlag, 2013

(Ertl 2009) Ertl, D.: Semi-automatic multimodal user interface generation. In: **Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems**. New York, NY, USA : ACM, 2009 (EICS '09), S. 321–324

(Euzenat und Shvaiko 2013) Euzenat, J. ; Shvaiko, P.: **Ontology Matching**. Berlin, Heidelberg : Springer Berlin Heidelberg : Imprint: Springer, 2013

(Feldmann u. a. 2009) Feldmann, M. ; Nestler, T. ; Muthmann, K. ; Jugel, U. ; Hübsch, G. ; Schill, A.: Overview of an End-user Enabled Model-driven Development Approach for Interactive Applications Based on Annotated Services. In: **Proceedings of the 4th Workshop on Emerging Web Services Technology**. New York, NY, USA : ACM, 2009 (WEWST '09), S. 19–28

(Fensel u. a. 2003) Fensel, D. ; Hendler, J. A. ; Lieberman, H. ; Wahlster, W.: **Spinning the Semantic Web**. Cambridge, MA : MIT Press, 2003

(Fettke 2009) Fettke, P.: Ansätze der Informationsmodellierung und ihre betriebswirtschaftliche Bedeutung: Eine Untersuchung der Modellierungspraxis in Deutschland. In: **Schmalenbachs Zeitschrift für betriebswirtschaftliche Forschung** 61 (2009), August, Nr. 5, S. 550–580

(Fielding 2000) Fielding, R. T.: **REST: Architectural Styles and the Design of Network-based Software Architectures**, University of California, Irvine, Doctoral dissertation, 2000

(Forgy 1982) Forgy, C. L.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In: **Artificial Intelligence** 19 (1982), September, Nr. 1, S. 17–37

(Foster 2002) Foster, M. E.: State of the art review: Multimodal fission. In: **COMIC project Deliverable** 6 (2002), Nr. 09

(Freund und Rücker 2016) Freund, J. ; Rücker, B.: **Praxishandbuch BPMN 2.0 - Mit Einführung in CMMN und DMN**. Fünfte. München; Wien : Hanser, 2016

(Frey 2015) Frey, J.: **ASaP - Integrationsplattform für Smart Services in Intelligenten Umgebungen**, Universität des Saarlandes, Dissertation, 2015

(Frey u. a. 2010) Frey, J. ; Stahl, C. ; Röfer, T. ; Krieg-Brückner, B. ; Alexandersson, J.: The DFKI Competence Center for Ambient Assisted Living. In: Ruyter, B. de ; Wichert, R. ; Keyson, D. ; Markopoulos, P. ; Streitz, N. ; Divitini, M. ; Georgantas, N. ; Mana Gomez, A. (Eds.): **Ambient Intelligence** Bd. 6439. Springer Berlin Heidelberg, 2010, S. 310–314

(Gabler Wirtschaftslexikon 2017) Gabler Wirtschaftslexikon: **Stichwort: Mash-Up**. September 2017. – URL http://wirtschaftslexikon.gabler.de/Archiv/81588/mash-up-v8.html. – (letzter Zugriff: 04.11.2017)

(Gamma u. a. 1994) Gamma, E. ; Helm, R. ; Johnson, R. ; Vlissides, J.: **Design Patterns: Elements of Reusable Object-Oriented Software**. Reading, Mass. : Addison-Wesley, 1994

(Gandhe u. a. 2011) Gandhe, S. ; Rushforth, M. ; Aggarwal, P. ; Traum, D. R.: Evaluation of an Integrated Authoring Tool for Building Advanced Question-Answering Characters. In: **INTERSPEECH'11**, 2011, S. 1289–1292

(Gandhe u. a. 2009) Gandhe, S. ; Whitman, N. ; Traum, D. ; Artstein, R.: An Integrated Authoring Tool for Tactical Questioning Dialogue Systems. In: **Proceedings of the 6th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems**, Association for the Advancement of Artificial Intelligence (AAAI), 2009

(Gangemi u. a. 2002) Gangemi, A. ; Guarino, N. ; Masolo, C. ; Oltramari, A. ; Schneider, L.: **Sweetening Ontologies with DOLCE**. S. 166–181. In: Gómez-Pérez, A. ; Benjamins, V. R. (Eds.): **Knowledge Engineering and Knowledge**

Management: Ontologies and the Semantic Web: 13th International Conference, EKAW 2002 Sigüenza, Spain, October 1–4, 2002 Proceedings. Berlin, Heidelberg : Springer, 2002

(Garrett 2010) Garrett, J. J.: **The elements of user experience : user-centered design for the Web and beyond**. Berkeley, CA : New Riders, 2010

(Gasevic u. a. 2009) Gasevic, D. ; Djuric, D. ; Devedzic, V. ; Gasevic, D.: **Model driven engineering and ontology development**. Dordrecht; New York : Springer, 2009

(Grice 1975) Grice, H. P.: Logic and Conversation. In: Cole, P. ; Morgan, J. L. (Eds.): **Syntax and Semantics: Vol. 3: Speech Acts**. New York : Academic Press, 1975, S. 41–58

(Gronback 2009) Gronback, R. C.: **Eclipse Modeling Project : A Domain-Specific Language (DSL) Toolkit**. Erste. Upper Saddle River, N.J. : Addison-Wesley, März 2009

(Grosz und Sidner 1986) Grosz, B. J. ; Sidner, C. L.: Attention, Intentions, and the Structure of Discourse. In: **Computational Linguistics** 12 (1986), Juli, Nr. 3, S. 175–204

(Hadley und Sandoz 2009) Hadley, M. ; Sandoz, P.: **JAX-RS: The Java API for RESTful Web Services**. Java Specification Request (JSR) 311. September 2009

(Hahn 2017) Hahn, V.: **A Dialogue Management Framework for Dialogue Adaption based on User Backchannels**, Universität des Saarlandes, Masterarbeit, Oktober 2017

(Hamann u. a. 2008) Hamann, T. ; Hübsch, G. ; Springer, T.: A Model-Driven Approach for Developing Adaptive Software Systems. In: Meier, R. ; Terzis, S. (Eds.): **Distributed Applications and Interoperable Systems** Bd. 5053. Springer Berlin Heidelberg, 2008, S. 196–209

(Handley u. a. 2006) Handley, M. ; Jacobson, V. ; Perkins, C.: **SDP: Session Description Protocol**. July 2006

(Harris und Seaborne 2013) Harris, S. ; Seaborne, A.: **SPARQL 1.1 Query Language**. März 2013

(Heckmann u. a. 2007) Heckmann, D. ; Schwarzkopf, E. ; Mori, J. ; Dengler, D. ; Kröner, A.: The User Model and Context Ontology GUMO Revisited for Future Web 2.0 Extensions. In: Bouquet, P. ; Euzenat, J. ; Ghidini, C. ; McGuinness, D. L. ; Serafini, L. ; Shvaiko, P. ; Wache, H. (Eds.): **C&O:RR** Bd. 298, CEUR-WS.org, 2007

(Heinrich u. a. 2007) Heinrich, M. ; Winkler, M. ; Steidelmüller, H. ; Zabelt, M. ; Behring, A. ; Neumerkel, R. ; Strunk, A.: MDA applied: a task-model driven tool chain for multimodal applications. In: **Proceedings of the 6th international conference on Task models and diagrams for user interface design**. Berlin/Heidelberg : Springer-Verlag, 2007 (TAMODIA'07), S. 15–27

(Heller und Allgaier 2010) Heller, M. ; Allgaier, M.: Model-based Service Integration for Extensible Enterprise Systems with Adaptation Patterns. In: Marca, D. ; Shishkov, B. ; Sinderen, M. van (Eds.): **Proceedings of the 2010 International Conference on e-Business (ICE-B)**, SciTEPress, 2010, S. 163–168

(Heller u. a. 2012) Heller, M. ; Schmeling, B. ; Heinzl, S. ; Leidig, T. ; Duddy, K. ; Sandfuchs, T. ; Klein, A. ; Allgaier, M.: **Enabling USDL by Tools**. S. 385–414. In: Barros, A. ; Oberle, D. (Eds.): **Handbook of Service Description: USDL and Its Methods**. Boston, MA : Springer US, 2012

(Hindelang 2010) Hindelang, G.: **Einführung in die Sprechakttheorie: Sprechakte, Äusserungsformen, Sprechaktsequenzen**. Fünfte. Berlin; New York : De Gruyter, 2010

(Hitzler u. a. 2008) Hitzler, P. ; Krötzsch, M. ; Rudolph, S. ; Sure, Y.: **Semantic Web Grundlagen**. Berlin/Heidelberg : Springer-Verlag, 2008 (eXamen.press)

(Horrocks und Patel-Schneider 2004) Horrocks, I. ; Patel-Schneider, P.: Reducing OWL Entailment to Description Logic Satisfiability. In: **Journal of Web Semantics** 1 (2004), Oktober, Nr. 4, S. 345–357

(Hunt und McGlashan 2004) Hunt, A. ; McGlashan, S.: **Speech Recognition Grammar Specification Version 1.0 - W3C Recommendation 16 March 2004**. März 2004

(IDC 2017) IDC: **Worldwide Wearables Market to Nearly Double by 2021, According to IDC**. Juni 2017. – URL https://www.idc.com/getdoc.jsp?containerId=prUS42818517. – (letzter Zugriff: 04.11.2017)

(Ings u. a. 2010) Ings, D. ; Clément, L. ; König, D. ; Mehta, V. ; Mueller, R. ; Rangaswamy, R. ; Rowley, M. ; Trickovic, I.: **WS-BPEL Extension for People (BPEL4People) Specification Version 1.1**. OASIS Committee Specification. August 2010

(Ings u. a. 2012) Ings, D. ; Clément, L. ; König, D. ; Mehta, V. ; Mueller, R. ; Rangaswamy, R. ; Rowley, M. ; Trickovic, I.: **Web Services Human Task (WS-HumanTask) Specification Version 1.1**. OASIS Committee Specification Draft 12 / Public Review Draft 05. July 2012

(ISO 2012) ISO: **ISO 24617-2:2012: Language resource management – Semantic annotation framework (SemAF) – Part 2: Dialogue acts**. 2012

(ISO/IEC 2014) ISO/IEC: **ISO/IEC 24752-1:2014: Information technology – User interfaces – Universal remote console – Part 1: General framework**. Dezember 2014

(Jafarian 2017) Jafarian, N.: **Generating Industrie 4.0 Assistant Web Applications from CMMN: A Model-driven Approach**, Universität des Saarlandes, Masterarbeit, Januar 2017

(Janzen u. a. 2011) Janzen, S. ; Blomqvist, E. ; Filler, A. ; Gönül, S. ; Kowatsch, T. ; Adamou, A. ; Germesin, S. ; Romanelli, M. ; Presutti, V. ; Cimen, C. ; Maass, W. ; Postaci, S. ; Alpay, E. ; Namli, T. ; Erturkmen, G. B. L.: IKS Deliverable - D4.1 Report: AmI Case - Design and Implementation (Public) / IKS Project, September 2011. – Deliverable

(Janzen u. a. 2010) Janzen, S. ; Kowatsch, T. ; Maass, W. ; Filler, A.: **Lecture Notes in Computer Science**. Bd. 6497: **Linkage of Heterogeneous Knowledge Resources within In-Store Dialogue Interaction**. S. 145–160. In: Patel-Schneider, P. F. ; Pan, Y. ; Hitzler, P. ; Mika, P. ; Zhang, L. ; Pan, J. Z. ; Horrocks, I. ; Glimm, B. (Eds.): **The Semantic Web – ISWC 2010: 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part II** Bd. 6497. Berlin, Heidelberg : Springer, 2010

(Janzen u. a. 2016) Janzen, S. ; Maass, W. ; Kowatsch, T.: Finding the Middle Ground - A Model for Planning Satisficing Answers. In: **Proceedings of 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)**, The Association for Computer Linguistics, August 2016, S. 547–557

(Jekat u. a. 1995) Jekat, S. ; Klein, A. ; Maier, E. ; Maleck, I. ; Mast, M. ; Quantz, J. J.: Dialogue Acts in VERBMOBIL. In: **Verbmobil Report** 1998 (1995), Nr. 65

(Jockisch und Rosendahl 2009) Jockisch, M. ; Rosendahl, J.: Klassifikation von Modellen. In: Bandow, G. ; Holzmüller, H. H. (Eds.): **"Das ist gar kein Modell!"**. Wiesbaden : Gabler, 2009, Kap. 2, S. 23–52

(Johnston 2009) Johnston, M.: Building multimodal applications with EMMA. In: **Proceedings of the 2009 international conference on Multimodal interfaces**. New York, NY, USA : ACM, 2009 (ICMI-MLMI '09), S. 47–54

(Johnston u. a. 2009) Johnston, M. ; Baggia, P. ; Burnett, D. C. ; Carter, J. ; Dahl, D. A. ; McCobb, G. ; Raggett, D.: **EMMA: Extensible MultiModal Annotation Markup Language - W3C Recommendation 10 February 2009**. February 2009

(Jurafsky und Martin 2009) Jurafsky, D. ; Martin, J. H.: **Speech and Language Processing: An Introduction to Natural Language Processing,**

Computational Linguistics and Speech Recognition. Upper Saddle River : Prentice Hall, Pearson Education International, 2009

(Kaptelinin und Nardi 2006) Kaptelinin, V. ; Nardi, B. A.: **Acting with Technology: Activity Theory and Interaction Design**. Cambridge, Mass. : MIT Press, 2006

(Kett u. a. 2014) Kett, H. ; Winkler, M. ; Kadner, K.: Integrated Service Engineering (ISE). In: (Wahlster u. a. 2014), S. 425–437

(Klusch 2008) Klusch, M.: Semantic Web Service Coordination. In: Schumacher, M. ; Helin, H. ; Schuldt, H. (Eds.): **CASCOM: Intelligent Service Coordination in the Semantic Web**. Basel : Birkhäuser, 2008 (Whitestein Series in Software Agent Technologies and Autonomic Computing), S. 59–104

(Kopp u. a. 2009) Kopp, O. ; Martin, D. ; Wutke, D. ; Leymann, F.: The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. In: **Enterprise Modelling and Information Systems Architectures** 4 (2009), June, Nr. 1, S. 3–13

(Kuhlmann u. a. 2014) Kuhlmann, F. ; Hannemann, J. ; Traub, M. ; Böhme, C. ; Zillner, S. ; Cavallaro, A. ; Seifert, S. ; Decker, B. ; Traphöner, R. ; Kayser, S. ; Lindemann, U. ; Prasse, S. ; Marczinski, G. ; Grützner, R. ; Fasse, A. ; Oberle, D.: The THESEUS Use Cases. In: (Wahlster u. a. 2014), S. 259–287

(Kühne 2005) Kühne, T.: What is a Model? In: Bezivin, J. ; Heckel, R. (Eds.): **Language Engineering for Model-Driven Software Development**. Dagstuhl, Germany : Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005 (Dagstuhl Seminar Proceedings 04101)

(Kurczynski 2008) Kurczynski, K.: Prozessmodellierung im Wettbewerb: EPK vs. BPMN. In: **is report** 12 (2008), Nr. 6, S. 30–35

(Lalanne u. a. 2009) Lalanne, D. ; Nigay, L. ; Palanque, P. A. ; Robinson, P. ; Vanderdonckt, J. ; Ladry, J.-F.: Fusion engines for multimodal input: a survey. In: Crowley, J. L. ; Ivanov, Y. ; Wren, C. R. ; Gatica-Perez, D. ; Johnston, M. ; Stiefelhagen, R. (Eds.): **ICMI**, ACM, 2009, S. 153–160

(Lane u. a. 2010) Lane, H. ; Schneider, M. ; Michael, S. ; Albrechtsen, J. ; Meissner, C.: Virtual Humans with Secrets: Learning to Detect Verbal Cues to Deception. In: Aleven, V. ; Kay, J. ; Mostow, J. (Eds.): **Intelligent Tutoring Systems** Bd. 6095. Springer Berlin Heidelberg, 2010, S. 144–154

(Larsson und Traum 2000) Larsson, S. ; Traum, D. R.: Information state and dialogue management in the TRINDI dialogue move engine toolkit. In: **Natural Language Engineering** 6 (2000), September, S. 323–340

(Löckelt 2008) Löckelt, M.: **A Flexible and Reusable Framework for Dialogue and Action Management in Multi-Party Discourse**, Saarland University, Dissertation, 2008

(Löckelt u. a. 2014) Löckelt, M. ; Deru, M. ; Schulz, C. H. ; Bergweiler, S. ; Becker, T. ; Reithinger, N.: A Unified Approach for Semantic-Based Multimodal Interaction. In: (Wahlster u. a. 2014), S. 131–144

(Leont'ev 1978) Leont'ev, A. N.: **Activity, consciousness, and personality**. Englewood Cliffs, N.J. : Prentice-Hall, 1978

(Leontiev 1974) Leontiev, A. N.: The Problem of Activity in Psychology. In: **Soviet Psychology** 13 (1974), Nr. 2, S. 4–33

(Li u. a. 2011) Li, H. ; Xu, F. ; Uszkoreit, H.: TechWatchTool: Innovation and Trend Monitoring. In: Angelova, G. ; Bontcheva, K. ; Mitkov, R. ; Nicolov, N. (Eds.): **Proceedings of the International Conference on Recent Advances in Natural Language Processing 2011 (RANLP 2011)**, RANLP 2011 Organising Committee, 2011, S. 660–665

(Limbourg u. a. 2004) Limbourg, Q. ; Vanderdonckt, J. ; Michotte, B. ; Bouillon, L. ; Florins, M.: USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. In: **ICWE Workshops**, 2004, S. 325–338

(Limbourg u. a. 2005) Limbourg, Q. ; Vanderdonckt, J. ; Michotte, B. ; Bouillon, L. ; López-Jaquero, V.: USIXML: A Language Supporting Multi-path Development of User Interfaces. In: Bastide, R. ; Palanque, P. ; Roth, J. (Eds.): **Engineering Human Computer Interaction and Interactive Systems** Bd. 3425. Berlin/Heidelberg : Springer-Verlag, 2005, Kap. 12, S. 200–220

(Löckelt 2006) Löckelt, M.: **Plan-Based Dialogue Management for Multiple Cooperating Applications**. S. 301–316. In: Wahlster, W. (Eds.): **SmartKom: Foundations of Multimodal Dialogue Systems**. Berlin, Heidelberg : Springer, 2006

(Maaß u. a. 2017) Maass, W. ; Shcherbatyi, I. ; Marquardt, S. ; Kritzner, A. ; Moser, B.: Real-Time Smart Farming Services. In: **Proceedings of the 75th International Conference on Agricultural Engineering (LAND.TECHNIK AgEng 2017)** Verein Deutscher Ingenieure (VDI) (Veranst.), November 2017

(Maass und Filler 2007) Maass, W. ; Filler, A.: Tip 'n Tell: Product-Centered Mobile Reasoning Support for Tangible Shopping. In: Nixon, L. J. B. ; Cuel, R. ; Francisco, D. de ; Simperl, E. ; Tempich, C. (Eds.): **Proceedings of the Workshop on Making Semantics Work For Business (MSWFB 2007), part of 1st European Semantic Technology Conference, Vienna, Austria**, Juni 2007, S. 12–17

(Maass u. a. 2011) Maass, W. ; Kowatsch, T. ; Janzen, S. ; Varshney, U.: A Natural Language Technology-enhanced Mobile Sales Assistant for In-store Shopping Situations. In: Tuunainen, V. K. ; Rossi, M. ; Nandhakumar, J. (Eds.): **19th European Conf. on Information Systems (ECIS 2011)**, Association for Information Systems, 2011

(Maass und Varshney 2012) Maass, W. ; Varshney, U.: Design and Evaluation of Ubiquitous Information Systems and Use in Healthcare. In: **Decision Support Systems** 54 (2012), August, Nr. 1, S. 597–609

(Mann und Thompson 1988) Mann, W. C. ; Thompson, S. A.: Rhetorical structure theory: Toward a functional theory of text organization. In: **Text - Interdisciplinary Journal for the Study of Discourse** 8 (1988), Nr. 3, S. 243–281

(Maybury und Wahlster 1998) Maybury, M. T. ; Wahlster, W.: An Introduction to Intelligent User Interfaces. In: **Readings in Intelligent User Interfaces**. 1998, S. 1–13

(McGlashan u. a. 2004) McGlashan, S. ; Burnett, D. C. ; Carter, J. ; Danielsen, P. ; Ferrans, J. ; Hunt, A. ; Lucas, B. ; Porter, B. ; Rehor, K. G. ; Tryphonas, S.: **Voice Extensible Markup Language (VoiceXML) Version 2.0 - W3C Recommendation 16 March 2004**. March 2004

(Meggle 1979) Meggle, G.: **Handlung, Kommunikation, Bedeutung**. Frankfurt am Main : Suhrkamp, 1979

(Meixner 2010) Meixner, G.: Model-based Useware Engineering. In: **Workshop on Future Standards for Model-Based User Interfaces** W3C (Veranst.), Mai 2010, S. 1–2

(Meixner u. a. 2011) Meixner, G. ; Seissler, M. ; Breiner, K.: Model-Driven Useware Engineering. In: Hussmann, H. ; Meixner, G. ; Zuehlke, D. (Eds.): **Model-Driven Development of Advanced User Interfaces** Bd. 340. Berlin : Springer, 2011, S. 1–26

(Mell und Grance 2011) Mell, P. ; Grance, T.: The NIST Definition of Cloud Computing / National Institute of Standards and Technology (NIST). Gaithersburg, MD, September 2011 (800-145). – Forschungsbericht

(Melzer und Eberhard 2008) Melzer, I. ; Eberhard, S.: **Service-orientierte Architekturen mit Web Services : Konzepte - Standards - Praxis**. Heidelberg : Spektrum Akad. Verl., 2008

(Metzger 2005) Metzger, A.: A Systematic Look at Model Transformations. In: Beydeda, S. ; Book, M. ; Gruhn, V. (Eds.): **Model-Driven Software Development**. Berlin/Heidelberg : Springer-Verlag, 2005, Kap. 2, S. 19–33

(Miller und Mukerji 2003) Miller, J. ; Mukerji, J.: MDA Guide Version 1.0.1. Needham, Massachusetts, June 2003 (omg/2003-06-01). – Forschungsbericht

(Mitra und Lafon 2007) Mitra, N. ; Lafon, Y.: **SOAP Version 1.2 Part 0: Primer (Second Edition)**. W3C Recommendation. April 2007

(Morbini u. a. 2013) Morbini, F. ; Audhkhasi, K. ; Sagae, K. ; Artstein, R. ; Can, D. ; Georgiou, P. ; Narayanan, S. ; Leuski, A. ; Traum, D.: Which ASR should I choose for my dialogue system? In: **Proceedings of the SIGDIAL 2013 Conference**. Metz, France : Association for Computational Linguistics, August 2013, S. 394–403

(Mori u. a. 2004) Mori, G. ; Paterno, F. ; Santoro, C.: Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. In: **IEEE Trans. Softw. Eng.** 30 (2004), August, S. 507–520

(zur Muehlen und Recker 2008) Muehlen, M. zur ; Recker, J.: How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In: Bellahsène, Z. ; Léonard, M. (Eds.): **Advanced Information Systems Engineering** Bd. 5074. Springer Berlin Heidelberg, 2008, S. 465–479

(Mueller u. a. 2004) Mueller, W. ; Schaefer, R. ; Bleul, S.: Interactive multimodal user interfaces for mobile devices. In: **Proceedings of the 37th Annual Hawaii International Conference on System Sciences**, January 2004

(Neßelrath 2016) Nesselrath, R.: **SiAM-dp: An open development platform for massively multimodal dialogue systems in cyber-physical environments**. Saarbrücken, Universität des Saarlandes, Dissertation, 2016

(Neßelrath und Feld 2014) Nesselrath, R. ; Feld, M.: SiAM-dp: A Platform for the Model-Based Development of Context-Aware Multimodal Dialogue Applications. In: **IE'14: Proceedings of the 10th International Conference on Intelligent Environments**. Shanghai, China : IEEE, July 2014

(Neßelrath u. a. 2015) Nesselrath, R. ; Becker, T. ; Reiplinger, M. ; Schwartz, T.: **SiAM-dp, eine multimodale Dialogplattform im Industriekontext**. S. 277 – 288. In: Reinhart, G. ; Scholz-Reiter, B. ; Wahlster, W. ; Wittenstein, M. ; Zühlke, D. (Eds.): **Intelligente Vernetzung in der Fabrik - Industrie 4.0 Umsetzungsbeispiele für die Praxis**. Stuttgart : Fraunhofer Verlag, September 2015

(Neßelrath und Porta 2011) Nesselrath, R. ; Porta, D.: Rapid Development of Multimodal Dialogue Applications with Semantic Models. In: **Proceedings of the 7th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems (KRPDS-11)**, July 2011

(NESSI 2014) NESSI: **Software Engineering: Key Enabler for Innovation**. Juli 2014. – URL http://www.nessi-europe.com/Files/Private/NESSI_SE_WhitePaper-FINAL.pdf. – (letzter Zugriff: 04.11.2017)

(Nestler u. a. 2010) Nestler, T. ; Dannecker, L. ; Pursche, A.: User-Centric Composition of Service Front-Ends at the Presentation Layer. In: Dan, A. ; Gittler, F. ; Toumani, F. (Eds.): **Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops** Bd. 6275. Springer Berlin Heidelberg, 2010, S. 520–529

(Nestler u. a. 2009) Nestler, T. ; Feldmann, M. ; Preussner, A. ; Schill, A.: Service Composition at the Presentation Layer using Web Service Annotations. In: **Proceedings of the First International Workshop on Lightweight Integration on the Web (ComposableWeb)**, 2009

(Nestler u. a. 2011) Nestler, T. ; Namoun, A. ; Schill, A.: End-user development of service-based interactive web applications at the presentation layer. In: **Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems**. New York, NY, USA : ACM, 2011 (EICS '11), S. 197–206

(Nigay und Coutaz 1993) Nigay, L. ; Coutaz, J.: A Design Space for Multimodal Systems: Concurrent Processing and Data Fusion. In: **Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems**. New York, NY, USA : ACM, 1993 (CHI '93), S. 172–178

(Niles und Pease 2001) Niles, I. ; Pease, A.: Towards a Standard Upper Ontology. In: **Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001**. New York, NY, USA : ACM, 2001 (FOIS '01), S. 2–9

(Norman 2002) Norman, D. A.: **The Design of Everyday Things**. New York : Basic Books, 2002

(OASIS 2006) OASIS: **Reference Model for Service Oriented Architecture 1.0**. August 2006

(OASIS 2007) OASIS: **Web Services Business Process Execution Language Version 2.0**. April 2007

(Oberle 2014) Oberle, D.: A Unified Description Language for the Internet of Services. In: (Wahlster u. a. 2014), S. 439–449

(Oestereich 2009) Oestereich, B.: **Analyse und Design mit der UML 2.3: Objektorientierte Softwareentwicklung**. Neunte. München : Oldenbourg, 2009

(OMG 2008) OMG: **MOF Model to Text Transformation Language, v1.0**. Januar 2008

(OMG 2011a) OMG: **Business Process Model and Notation (BPMN), Version 2.0**. January 2011

(OMG 2011b) OMG: **Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1**. January 2011

(OMG 2011c) OMG: **OMG MOF 2 XMI Mapping Specification, Version 2.4.1**. August 2011

(OMG 2011d) OMG: **OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1**. August 2011

(OMG 2011e) OMG: **OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1**. August 2011

(OMG 2012) OMG: **OMG Object Constraint Language (OCL), Version 2.3.1**. January 2012

(OMG 2013a) OMG: **Business Process Model and Notation (BPMN), Version 2.0.2**. Dezember 2013

(OMG 2013b) OMG: **OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1**. June 2013

(OMG 2014a) OMG: **Case Management Model and Notation, Version 1.0**. May 2014

(OMG 2014b) OMG: **Interface Definition Language, Version 3.5**. März 2014

(OMG 2016) OMG: **Case Management Model and Notation, Version 1.1**. Januar 2016

(Oracle 2016) Oracle: **Can Virtual Experiences Replace Reality? The future role for humans in delivering customer experience.** 2016. – URL https://go.oracle.com/LP=43079?elqCampaignId=79575. – (letzter Zugriff: 04.11.2017)

(Oviatt 2012) Oviatt, S.: Multimodal Interfaces. In: Jacko, J. A. (Eds.): **Human Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications, Third Edition**. Boca Raton, FL, USA : CRC Press, Inc., 2012, Kap. Multimodal interfaces, S. 405–430

(Oviatt und Cohen 2015) Oviatt, S. ; Cohen, P. R.: **The Paradigm Shift to Multimodality in Contemporary Computer Interfaces**. San Rafael, CA : Morgan & Claypool Publishers, 2015 (Synthesis Lectures on Human-Centered Informatics)

(Paternò u. a. 2011a) Paternò, F. ; Santoro, C. ; Spano, L.: A Design Space for User Interface Composition. In: Hussmann, H. ; Meixner, G. ; Zuehlke, D. (Eds.): **Model-Driven Development of Advanced User Interfaces** Bd. 340. Springer Berlin Heidelberg, 2011, S. 43–65

(Paternò u. a. 2011b) Paternò, F. ; Santoro, C. ; Spano, L. D.: Engineering the authoring of usable service front ends. In: **Journal of Systems and Software** 84 (2011), Nr. 10, S. 1806 – 1822

(Paternò und Giammarino 2006) Paternò, F. ; Giammarino, F.: Authoring interfaces with combined use of graphics and voice for both stationary and mobile devices. In: **Proceedings of the working conference on Advanced visual interfaces**. New York, NY, USA : ACM, 2006 (AVI '06), S. 329–335

(Paternò u. a. 1997) Paternò, F. ; Mancini, C. ; Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: **Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction**. London, UK, UK : Chapman & Hall, Ltd., 1997 (INTERACT '97), S. 362–369

(Paternò u. a. 2008) Paternò, F. ; Santoro, C. ; Mantyjarvi, J. ; Mori, G. ; Sansone, S.: Authoring pervasive multimodal user interfaces. In: **Int. J. Web Eng. Technol.** 4 (2008), Mai, S. 235–261

(Paternò u. a. 2009) Paternò, F. ; Santoro, C. ; Spano, L. D.: MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. In: **ACM Trans. Comput.-Hum. Interact.** 16 (2009), November, Nr. 4, S. 1–30

(Paulheim und Erdogan 2010) Paulheim, H. ; Erdogan, A.: Seamless Integration of Heterogeneous UI Components. In: **Proceedings of the 2Nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems**. New York, NY, USA : ACM, 2010 (EICS '10), S. 303–308

(Paulheim und Probst 2010) Paulheim, H. ; Probst, F.: Application Integration on the User Interface Level: An Ontology-Based Approach. In: **Data Knowl. Eng.** 69 (2010), Nr. 11, S. 1103–1116

(Paulheim und Probst 2013) Paulheim, H. ; Probst, F.: UI^{2}Ont—A Formal Ontology on User Interfaces and Interactions. In: Hussein, T. ; Paulheim, H. ; Lukosch, S. ; Ziegler, J. ; Calvary, G. (Eds.): **Semantic Models for Adaptive Interactive Systems**. London : Springer, 2013, S. 1–24

(Pautasso u. a. 2008) Pautasso, C. ; Zimmermann, O. ; Leymann, F.: Restful web services vs. "big" web services: making the right architectural decision. In: **Proceedings of the 17th international conference on World Wide Web**. New York, NY, USA : ACM, 2008 (WWW '08), S. 805–814

(Pavel und Euzenat 2013) Pavel, S. ; Euzenat, J.: Ontology Matching: State of the Art and Future Challenges. In: **IEEE Transactions on Knowledge and Data Engineering** 25 (2013), Januar, Nr. 1, S. 158–176

(Pedrinaci u. a. 2014) Pedrinaci, C. ; Cardoso, J. ; Leidig, T.: Linked USDL: A Vocabulary for Web-Scale Service Trading. In: Presutti, V. ; d'Amato, C. ; Gandon, F. ; d'Aquin, M. ; Staab, S. ; Tordai, A. (Eds.): **The Semantic Web: Trends and Challenges** Bd. 8465. Springer International Publishing, 2014, S. 68–82

(Pfleger 2007) Pfleger, N.: **Context-based Multimodal Interpretation: An Integrated Approach to Multimodal Fusion and Discourse Processing**. Postfach 151141, 66041 Saarbrücken, Saarland University, Dissertation, November 2007

(Pfleger u. a. 2015) Pfleger, N. ; Becker, T. ; Neisens, D. S. ; Schwartz, T.: **Eine Entwicklungs- und Ablaufumgebung für das interaktive Handbuch für Cyber-Physische Produktionssysteme**. S. 337–349. In: Reinhart, G. ; Scholz-Reiter, B. ; Wahlster, W. ; Wittenstein, M. ; Zühlke, D. (Eds.): **Intelligente Vernetzung in der Fabrik - Industrie 4.0 Umsetzungsbeispiele für die Praxis**. Stuttgart : Fraunhofer Verlag, September 2015

(Pietschmann 2012) Pietschmann, S.: **Modellgetriebene Entwicklung adaptiver, komponentenbasierter Mashup-Anwendungen**, Technischen Universität Dresden, Dissertation, Januar 2012

(Pietschmann u. a. 2010a) Pietschmann, S. ; Nestler, T. ; Daniel, F.: Application composition at the presentation layer: alternatives and open issues. In: **Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services**. New York, NY, USA : ACM, 2010 (iiWAS '10), S. 461–468

(Pietschmann u. a. 2010b) Pietschmann, S. ; Tietz, V. ; Reimann, J. ; Liebing, C. ; Pohle, M. ; Meissner, K.: A metamodel for context-aware component-based mashup applications. In: **Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services**. New York, NY, USA : ACM, 2010 (iiWAS '10), S. 413–420

(Pietschmann u. a. 2009) Pietschmann, S. ; Voigt, M. ; Rümpel, A. ; Meissner, K.: CRUISe: Composition of Rich User Interface Services. In: Gaedke, M. ; Grossniklaus, M. ; Díaz, O. (Eds.): **Web Engineering** Bd. 5648. Springer Berlin Heidelberg, 2009, S. 473–476

(Plattform Industrie 4.0 2016) Plattform Industrie 4.0: **Struktur der Verwaltungsschale**. April 2016. – URL http://www.zvei.org/Verband/Publikationen/Seiten/Struktur-der-Verwaltungsschale.aspx. – (letzter Zugriff: 04.11.2017)

(Polgar 2012) Polgar, J.: **Using WSRP 2.0 with JSR 168 and 286 Portlets**. S. 37–49. In: Adamson, G. ; Polgar, J. (Eds.): **Enhancing Enterprise and Service-Oriented Architectures with Advanced Web Portal Technologies**. Hershey, PA, USA : IGI Global, 2012

(Porta 2008) Porta, D.: A Novel, Community-enabled Mobile Information System for Hikers. In: **Proceedings of the 2nd International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM-08)**, IEEE Computer Society Press, 2008, S. 438–444

(Porta 2010) Porta, D.: Towards Model-driven Development of Mobile Multimodal User Interfaces for Services. In: Fähnrich, K.-P. ; Franczyk, B. (Eds.): **Informatik 2010: Service Science - Neue Perspektiven für die Informatik, Beiträge der 40. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Band 1** Bd. 175, GI, September 2010, S. 497–502

(Porta und Conrad 2008) Porta, D. ; Conrad, J.: UBIGIouS: a ubiquitous, mixed-reality geographic information system. In: **Proceedings of the 13th international conference on intelligent user interfaces**. New York, NY, USA : ACM, 2008 (IUI '08), S. 393–396

(Porta u. a. 2014a) Porta, D. ; Deru, M. ; Bergweiler, S. ; Herzog, G. ; Poller, P.: Building Multimodal Dialog User Interfaces in the Context of the Internet of Services. In: (Wahlster u. a. 2014), S. 145–162

(Porta u. a. 2009a) Porta, D. ; Sonntag, D. ; Nesselrath, R.: A Multimodal Mobile B2B Dialogue Interface on the iPhone. In: **Proceedings of the 4th Workshop on Speech in Mobile and Pervasive Environments (SiMPE '09) in conjunction with MobileHCI '09**, 2009

(Porta u. a. 2009b) Porta, D. ; Sonntag, D. ; Nesselrath, R.: New business to business interaction: shake your iPhone and speak to it. In: **Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services**. New York, NY, USA : ACM, 2009 (MobileHCI '09), S. 59:1–59:2

(Porta u. a. 2014b) Porta, D. ; Tuncer, Z. ; Wirth, M. ; Hellenschmidt, M.: Multimodal Task Assignment and Introspection in Distributed Agricultural Harvesting Processes. In: **Proceedings of the Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2014), Rome, Italy**, IARIA/XPS (Xpert Publishing Services), August 2014, S. 227–232

(Rahmani u. a. 2010) Rahmani, T. ; Oberle, D. ; Dahms, M.: An Adjustable Transformation from OWL to Ecore. In: Petriu, D. ; Rouquette, N. ; Haugen,

y. (Eds.): **Model Driven Engineering Languages and Systems** Bd. 6395. Springer Berlin / Heidelberg, 2010, S. 243–257

(Recio-García u. a. 2014) Recio-García, J. A. ; González-Calero, P. A. ; Díaz-Agudo, B.: Jcolibri2: A Framework for Building Case-based Reasoning Systems. In: **Sci. Comput. Program.** 79 (2014), Januar, S. 126–145

(Reithinger u. a. 2006) Reithinger, N. ; Gebhard, P. ; Löckelt, M. ; Ndiaye, A. ; Pfleger, N. ; Klesen, M.: VirtualHuman: Dialogic and Affective Interaction with Virtual Characters. In: **Proceedings of the 8th International Conference on Multimodal Interfaces**. New York, NY, USA : ACM, 2006 (ICMI '06), S. 51–58

(Reithinger und Sonntag 2005) Reithinger, N. ; Sonntag, D.: An integration framework for a mobile multimodal dialogue system accessing the semantic web. In: **INTERSPEECH**, ISCA, 2005, S. 841–844

(Riedl u. a. 2009) Riedl, C. ; May, N. ; Finzen, J. ; Stathel, S. ; Kaufman, V. ; Krcmar, H.: An Idea Ontology for Innovation Management. In: **International Journal on Semantic Web and Information Systems (IJSWIS)** 5 (2009), Nr. 4, S. 1–18

(Rosenberg u. a. 2002) Rosenberg, J. ; Schulzrinne, H. ; Camarillo, G. ; Johnston, A. ; Peterson, J. ; Sparks, R. ; Handley, M. ; Schooler, E.: **SIP: Session Initiation Protocol**. June 2002

(Rump 1999) Rump, F. J.: **Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozessketten: Formalisierung, Analyse und Ausführung von EPKs**. Stuttgart : Teubner, 1999

(Russell u. a. 2006) Russell, N. ; Aalst, W. van der ; Hofstede, A. ter: Exception Handling Patterns in Process-Aware Information Systems / BPM Center, 2006 (BPM-06-04). – Forschungsbericht

(Russell u. a. 2005a) Russell, N. ; Aalst, W. M. van der ; Hofstede, A. H. ter ; Edmond, D.: Workflow Resource Patterns: Identification, Representation and Tool Support. In: Pastor, O. ; Cunha, J. Falcão e (Eds.): **Advanced Information Systems Engineering** Bd. 3520. Springer Berlin Heidelberg, 2005, S. 216–232

(Russell u. a. 2005b) Russell, N. ; Hofstede, A. H. ter ; Edmond, D. ; Aalst, W. M. van der: Workflow Data Patterns: Identification, Representation and Tool Support. In: Delcambre, L. ; Kop, C. ; Mayr, H. ; Mylopoulos, J. ; Pastor, O. (Eds.): **Conceptual Modeling – ER 2005** Bd. 3716. Springer Berlin Heidelberg, 2005, S. 353–368

(Russell und Norvig 2003) Russell, S. J. ; Norvig, P.: **Artificial Intelligence: A Modern Approach**. Zweite. Upper Saddle River, New Jersey, USA : Prentice Hall, 2003

(Sacks u. a. 1974) Sacks, H. ; Schegloff, E. ; Jefferson, G.: A simplest systematics for the organization of turn-taking for conversation. In: **Language** 50 (1974), December, Nr. 4, Part 1, S. 696–735

(Scheer 2002) Scheer, A.-W.: **ARIS - vom Geschäftsprozess zum Anwendungssystem**. Berlin [u.a.] : Springer, 2002

(Schegloff und Sacks 1973) Schegloff, E. A. ; Sacks, H.: Opening Up Closings. In: **Semiotica** 8 (1973), Nr. 4, S. 289–327

(Schehl u. a. 2008) Schehl, J. ; Pfalzgraf, A. ; Pfleger, N. ; Steigner, J.: The BabbleTunes System: Talk to Your iPod! In: **Proceedings of the 10th international conference on Multimodal interfaces**. New York, NY, USA : ACM, 2008 (ICMI '08), S. 77–80

(Scheithauer u. a. 2009) Scheithauer, G. ; Voigt, K. ; Bicer, V. ; Heinrich, M. ; Strunk, A. ; Winkler, M.: Integrated service engineering workbench: service engineering for digital ecosystems. In: **Proceedings of the International Conference on Management of Emergent Digital EcoSystems**. New York, NY, USA : ACM, 2009 (MEDES '09)

(Schulzrinne u. a. 2003) Schulzrinne, H. ; Casner, S. ; Frederick, R. ; Jacobson, V.: **RTP: A Transport Protocol for Real-Time Applications**. July 2003

(Searle 1969) Searle, J. R.: **Speech Acts: An Essay in the Philosophy of Language**. Cambridge, London : Cambridge University Press, 1969

(Searle 1975a) Searle, J. R.: Indirect speech acts. In: Cole, P. ; Morgan, J. (Eds.): **Syntax and Semantics 3: Speech Acts**. New York : Academic Press, 1975, S. 59–82

(Searle 1975b) Searle, J. R.: A Taxonomy of Illocutionary Acts. In: Gunderson, K. (Eds.): **Language, Mind and Knowledge**, University of Minnesota Press, 1975, S. 344–369

(Seidlmeier 2010) Seidlmeier, H.: **Prozessmodellierung mit ARIS : eine beispielorientierte Einführung für Studium und Praxis**. Dritte. Wiesbaden : Vieweg + Teubner, 2010

(Sonntag u. a. 2009a) Sonntag, D. ; Deru, M. ; Bergweiler, S.: Design and Implementation of Combined Mobile and Touchscreen-based Multimodal Web 3.0 Interfaces. In: **Proceedings of the 2009 International Conference on Artificial Intelligence, ICAI 2009, July 13-16, 2009, Las Vegas Nevada, USA, 2 Volumes**, 2009, S. 974–979

(Sonntag u. a. 2007) Sonntag, D. ; Engel, R. ; Herzog, G. ; Pfalzgraf, A. ; Pfleger, N. ; Romanelli, M. ; Reithinger, N.: SmartWeb Handheld — Multimodal

Interaction with Ontological Knowledge Bases and Semantic Web Services. In: Huang, T. ; Nijholt, A. ; Pantic, M. ; Pentland, A. (Eds.): **Artifical Intelligence for Human Computing** Bd. 4451. Berlin / Heidelberg : Springer, 2007, S. 272–295

(Sonntag u. a. 2010a) Sonntag, D. ; Porta, D. ; Setz, J.: HTTP/REST-based Meta Web Services in Mobile Application Frameworks. In: **Proceedings of the 4nd International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM-10)**, IARIA/XPS (Xpert Publishing Services), Oktober 2010, S. 170–175

(Sonntag u. a. 2009b) Sonntag, D. ; Sonnenberg, G. ; Nesselrath, R. ; Herzog, G.: Supporting a Rapid Dialogue System Engineering Process. In: **Proceedings of the First International Workshop On Spoken Dialogue Systems Technology (IWSDS)**, Dezember 2009

(Sonntag u. a. 2010b) Sonntag, D. ; Weihrauch, C. ; Jacobs, O. ; Porta, D.: THESEUS CTC-WP4 Usability Guidelines for Use Case Applications / DFKI, BMWi, April 2010. – Technical Report. – 96 S

(Sousa u. a. 2007) Sousa, K. ; Mendonça, H. ; Vanderdonckt, J.: Towards Method Engineering of Model-Driven User Interface Development. In: Winckler, M. ; Johnson, H. ; Palanque, P. (Eds.): **Task Models and Diagrams for User Interface Design** Bd. 4849. Springer Berlin Heidelberg, 2007, S. 112–125

(Sousa u. a. 2010) Sousa, K. ; Mendonça, H. ; Vanderdonckt, J.: A Rule-Based Approach for Model Management in a User Interface – Business Alignment Framework. In: England, D. ; Palanque, P. ; Vanderdonckt, J. ; Wild, P. J. (Eds.): **Task Models and Diagrams for User Interface Design** Bd. 5963. Springer Berlin Heidelberg, 2010, S. 1–14

(Sousa u. a. 2008) Sousa, K. ; Mendonça, H. ; Vanderdonckt, J. ; Rogier, E. ; Vandermeulen, J.: User Interface Derivation from Business Processes: A Model-driven Approach for Organizational Engineering. In: **Proceedings of the 2008 ACM Symposium on Applied Computing**. New York, NY, USA : ACM, 2008 (SAC '08), S. 553–560

(Speicher 2015) Speicher, A.: **Unterstützung der Erzeugung von UI-Modellen durch fallbasiertes Schließen**, Universität des Saarlandes, Bachelorarbeit, 2015

(Spillner u. a. 2009) Spillner, J. ; Buder, B. ; Schiefer, T. ; Schill, A.: Contract Services for Post-discovery Guarantee Management. In: Shishkov, B. ; Cordeiro, J. ; Ranchordas, A. (Eds.): **Proceedings of the 4th International Conference on Software and Data Technologies** Bd. 2, INSTICC Press, July 2009, S. 369–375

(Staab u. a. 2010) Staab, S. ; Walter, T. ; Gröner, G. ; Parreiras, F.: Model Driven Engineering with Ontology Technologies. In: Assmann, U. ; Bartho, A. ; Wende, C. (Eds.): **Reasoning Web. Semantic Technologies for Software Engineering** Bd. 6325. Springer Berlin / Heidelberg, 2010, S. 62–98

(Stachowiak 1973) Stachowiak, H.: **Allgemeine Modelltheorie**. Wien, New York : Springer-Verlag, 1973

(Stachowiak 1989) Stachowiak, H.: Modell. In: Seiffert, H. ; Radnitzky, G. (Eds.): **Handlexikon zur Wissenschaftstheorie**. München : Ehrenwirth Verlag, 1989, S. 219–222

(Stahl u. a. 2007) Stahl, T. ; Völter, M. ; Efftinge, S. ; Haase, A.: **Modellgetriebene Softwareentwicklung - Techniken, Engineering, Management**. Heidelberg : dpunkt.verlag, Mai 2007

(Stanciulescu u. a. 2005) Stanciulescu, A. ; Limbourg, Q. ; Vanderdonckt, J. ; Michotte, B. ; Montero, F.: A transformational approach for multimodal web user interfaces based on UsiXML. In: **Proceedings of the 7th international conference on Multimodal interfaces**. New York, NY, USA : ACM, 2005 (ICMI '05), S. 259–266

(Stanciulescu und Vanderdonckt 2007) Stanciulescu, A. ; Vanderdonckt, J.: Design Options for Multimodal Web Applications. In: Calvary, G. ; Pribeanu, C. ; Santucci, G. ; Vanderdonckt, J. (Eds.): **Computer-Aided Design of User Interfaces V**. Dordrecht, Netherlands : Springer-Verlag, 2007, Kap. 4, S. 41–56

(Staud 2006) Staud, J.: **Geschäftsprozessanalyse - Ereignisgesteuerte Prozessketten und objektorientierte Geschäftsprozessmodellierung für Betriebswirtschaftliche Standardsoftware**. Dritte. Berlin : Springer, January 2006

(Steinberg u. a. 2009) Steinberg, D. ; Budinsky, F. ; Paternostro, M. ; Merks, E.: **EMF : Eclipse Modeling Framework**. Zweite. Upper Saddle River, NJ : Addison-Wesley, 2009

(Swenson 2010) Swenson, K. D.: **Mastering the unpredictable : how adaptive case management will revolutionize the way that knowledge workers get things done**. Tampa, Fla. : Meghan-Kiffer Press, 2010

(Swenson 2011) Swenson, K. D.: **Taming the unpredictable Real World Adaptive Case Management**. London : Future Strategies, 2011

(Thomas 2001) Thomas, M.: Die Vielfalt der Modelle in der Informatik. In: **Informatikunterricht und Medienbildung, INFOS 2001, 9. GI-Fachtagung Informatik und Schule**, GI, 2001, S. 173–186

(Thomas und Nüttgens 2014) Thomas, O. ; Nüttgens, M.: **Dienstleistungsmodellierung 2014 : Vom Servicemodell zum Anwendungssystem**. Springer Gabler, Wiesbaden, 2014

(von Thun 2016, Erstausgabe 1981) Thun, F. S. von: **Miteinander reden 1: Störungen und Klärungen: Allgemeine Psychologie der Kommunikation**. 53. Reinbek bei Hamburg : Rowohlt-Taschenbuch-Verlag, 2016, Erstausgabe 1981 (rororo)

(Trætteberg 2003) Trætteberg, H.: Dialog Modelling with Interactors and UML Statecharts – A Hybrid Approach. In: Jorge, J. ; Jardim Nunes, N. ; Cunha, J. Falcão e (Eds.): **Interactive Systems. Design, Specification, and Verification** Bd. 2844. Berlin/Heidelberg : Springer-Verlag, 2003, S. 289–301

(Trætteberg 2007) Trætteberg, H.: A Hybrid Tool for User Interface Modeling and Prototyping. In: Calvary, G. ; Pribeanu, C. ; Santucci, G. ; Vanderdonckt, J. (Eds.): **Computer-Aided Design of User Interfaces V**. Dordrecht, Netherlands : Springer-Verlag, 2007, Kap. 18, S. 215–230

(Traum 1999) Traum, D.: Computational Models of Grounding in Collaborative Systems / AAAI, 1999 (FS-99-03). – Forschungsbericht. – 124–131 S

(Traum und Larsson 2003) Traum, D. ; Larsson, S.: The Information State Approach to Dialogue Management. In: **Current and New Directions in Discourse and Dialogue** Bd. 22. Springer Netherlands, 2003, S. 325–353

(Tuncer u. a. 2014) Tuncer, Z. ; Porta, D. ; Wirth, M. ; Hellenschmidt, M.: Multimodal Coordination of an Agricultural Harvesting Process. In: **Proceedings of the 72th International Conference Agricultural Engineering (LAND.TECHNIK.2014)** Verein Deutscher Ingenieure (VDI) (Veranst.), November 2014

(Van Der Aalst u. a. 2003) Van Der Aalst, W. M. P. ; Ter Hofstede, A. H. M. ; Kiepuszewski, B. ; Barros, A. P.: Workflow Patterns. In: **Distrib. Parallel Databases** 14 (2003), July, Nr. 1, S. 5–51

(Vancea u. a. 2008) Vancea, A. ; Grossniklaus, M. ; Norrie, M. C.: Database-Driven Web Mashups. In: **2008 Eighth International Conference on Web Engineering**, IEEE, 2008, S. 162–174

(Vanderdonckt 2005) Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: Pastor, O. ; Cunha, J. Falcão e (Eds.): **Advanced Information Systems Engineering** Bd. 3520. Springer Berlin Heidelberg, 2005, S. 16–31

(W3C OWL Working Group 2012) W3C OWL Working Group: **OWL 2 Web Ontology Language Document Overview (Second Edition) - W3C Recommendation 11 December 2012**. Dezember 2012

(Wahlster 1991) Wahlster, W.: User and discourse models for multimodal communication. In: Sullivan, J. W. ; Tyler, S. W. (Eds.): **Intelligent User Interfaces**. New York, NY, USA : ACM Press, 1991, S. 45–67

(Wahlster 2003) Wahlster, W.: Towards Symmetric Multimodality: Fusion and Fission of Speech, Gesture, and Facial Expression. In: Günter, A. ; Kruse, R. ; Neumann, B. (Eds.): **KI 2003: Advances in Artificial Intelligence** Bd. 2821. Berlin/Heidelberg : Springer-Verlag, 2003, Kap. 1, S. 1–18

(Wahlster 2006) Wahlster, W.: **SmartKom: Foundations of Multimodal Dialogue Systems**. Berlin/Heidelberg : Springer-Verlag, August 2006

(Wahlster u. a. 1993) Wahlster, W. ; André, E. ; Finkler, W. ; Profitlich, H.-J. ; Rist, T.: Plan-based Integration of Natural Language and Graphics Generation. In: **Artif. Intell.** 63 (1993), Oktober, Nr. 1-2, S. 387–427

(Wahlster u. a. 2014) Wahlster, W. ; Grallert, H.-J. ; Wess, S. ; Friedrich, H. ; Widenka, T.: **Towards the Internet of Services: The THESEUS Research Program**. Springer, Cham, 2014

(Wasinger 2006) Wasinger, R.: **Multimodal interaction with mobile devices: fusing a broad spectrum of modality combinations.**, Saarland University, Dissertation, 2006

(Watzlawick u. a. 2011, Erstausgabe 1967) Watzlawick, P. ; Bavelas, J. B. ; Jackson, D. D.: **Menschliche Kommunikation: Formen, Störungen, Paradoxien**. 12. Hogrefe, vorm. Verlag Hans Huber, 2011, Erstausgabe 1967 (Psychologie-Klassiker)

(Williams u. a. 2008) Williams, J. D. ; Poupart, P. ; Young, S.: **Partially Observable Markov Decision Processes with Continuous Observations for Dialogue Management**. S. 191–217. In: Dybkjær, L. ; Minker, W. (Eds.): **Recent Trends in Discourse and Dialogue**. Dordrecht : Springer Netherlands, 2008

(Willink und Matragkas 2015) Willink, E. ; Matragkas, N.: QVT Traceability: What does it really mean? In: **Proceedings of the 4th Workshop on the Analysis of Model Transformations (AMT 2015)**, Selbstverlag der Workshop-Organisatoren, September 2015

(Winkler u. a. 2009) Winkler, M. ; Spillner, J. ; Schill, A.: SLA Management and Contract-Based Service Execution. In: Baresi, L. ; Chi, C.-H. ; Suzuki, J.

(Eds.): **Service-Oriented Computing** Bd. 5900. Springer Berlin Heidelberg, 2009, S. 653–655

(Wittgenstein 1953) Wittgenstein, L.: **Philosophical Investigations**. Oxford : Basil Blackwell, 1953

(Wütherich u. a. 2008) Wütherich, G. ; Hartmann, N. ; Kolb, B. ; Lübken, M.: **Die OSGi Service Platform : Eine Einführung mit Eclipse Equinox**. Heidelberg : dpunkt.verl., 2008

(Yao u. a. 2010) Yao, X. ; Bhutada, P. ; Georgila, K. ; Sagae, K. ; Artstein, R. ; Traum, D. R.: Practical Evaluation of Speech Recognizers for Virtual Human Dialogue Systems. In: Calzolari, N. ; Choukri, K. ; Maegaard, B. ; Mariani, J. ; Odijk, J. ; Piperidis, S. ; Rosner, M. ; Tapias, D. (Eds.): **LREC**, European Language Resources Association, 2010

(Zachman 1987) Zachman, J. A.: A Framework for Information Systems Architecture. In: **IBM Systems Journal** 26 (1987), Nr. 3, S. 276–292

(Zanten 1998) Zanten, G. V. V.: Adaptive Mixed-Initiative Dialogue Management. In: **Interactive Voice Technology for Telecommunications Applications (IVTTA'98)**. Washington, DC, USA : IEEE Computer Society, September 1998, S. 65–70

(Zühlke 2004) Zühlke, D.: **Useware-Engineering für technische Systeme**. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg New York, 2004 (VDI-Buch)

(Zimmermann u. a. 2011) Zimmermann, G. ; Alexandersson, J. ; Buiza, C. ; Urdaneta, E. ; Diaz, U. ; Carrasco, E. ; Klima, M. ; Pfalzgraf, A.: **Meeting the Needs of Diverse User Groups: Benefits and Costs of Pluggable User Interfaces in Designing for Older People and People with Cognitive Impairments**. S. 80–93. In: Soar, J. ; Swindell, R. ; Tsang, P. (Eds.): **Intelligent Technologies for Bridging the Grey Digital Divide**. Hershey, PA, USA : IGI Global, 2011

(Zimmermann und Vanderheiden 2007) Zimmermann, G. ; Vanderheiden, G. C.: The Universal Control Hub: An Open Platform for Remote User Interfaces in the Digital Home. In: **Springer LNCS. Human-Computer Interaction. Interaction Platforms and Techniques.** 4551 (2007), S. 1040–1049

(Zühlke 2012) Zühlke, D.: **Vorgehensweise bei der Useware-Entwicklung**. S. 35–122. In: **Nutzergerechte Entwicklung von Mensch-Maschine-Systemen: Useware-Engineering für technische Systeme**. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012