Service-orientierte Architektur: Identifizierung äquivalenter Services in Form semantischer semiautomatischer Unterstützung des EMEO-Layers

Der Fakultät für Mathematik und Informatik

der Universität Leipzig

eingereichte

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

im Fachgebiet Informatik

vorgelegt von

Dipl. Wirt.-Inform., Master of Computer Science

Michael Herrmann

geboren am 08.09.1970 in Reutlingen

Leipzig, den 20. April 2008

Danksagung

Diese Dissertation wurde bei der Daimler AG in der Abteilung für „Methoden und Technologien Mercedes-Benz Cars“ in einem kooperativen Promotionsverfahren mit der Universität Leipzig, Fakultät für Mathematik und Informatik sowie der Hochschule Reutlingen, Fakultät für Wirtschaftsinformatik erstellt.

An erster Stelle möchte ich Herrn Prof. Dr.-Ing. Wilhelm G. Spruth für die Unterstützung, die Anregungen sowie die konstruktive Kritik während meiner gesamten Zeit als Doktorand danken.

Für sehr hilfreiche wissenschaftliche Diskussionen möchte ich mich bei den Professoren und Doktoranden der Universität Leipzig Herrn Prof. Dr. Heinrich Herre, Herrn Prof. Dr. Gerhard Brewka, Herrn Prof. Dr. Klaus-Peter Fähnrich, Herrn Dr.-Ing. M. A. Aslam, Herrn Jens Lehmann, Herrn Dr. Sören Auer, Herrn Stefan Kühne und Herrn Maik Thränert bedanken.

Der Hochschule Reutlingen, Fakultät für Wirtschaftsinformatik sowie im besonderen Maße Herrn Prof. Dr. Alfred Zimmermann und Herrn Prof. Dr. Fritz Laux möchte ich für die sehr gute Unterstützung und Hilfestellung zur Promotionsvorprüfung danken.

Für die Möglichkeit zur Promotion in der Daimler AG, das Vertrauen und die Unterstützung möchte ich mich ganz herzlich bei Herrn Dr. Claus Lehners, Herrn Hans Moertl und im besonderen Maße bei Herrn Hans-Jürgen Groß bedanken.

Mein Dank für zahlreiche inspirierende Anregungen gilt dem kompletten Team IT:Factory, insbesondere Herrn Baghdad Abdesselam, Herrn Boris Basica, Herrn Oliver Burgmaier und Herrn Reinhard Griesinger. Ebenso der Kollegin Frau Claudia Rauscher und den Kollegen Herrn Wolfgang Schuster und Herrn Klaus-Dieter Löb.

Bei Herrn Richard Golden bedanke ich mich für die positiven Anregungen, die Unterstützung bei meinen nationalen bzw. internationalen Präsentationen und die fachliche Konversation in englischer Sprache. Mein Dank für die prototypische Umsetzung des EMEO-Layers gilt Herrn Hans-Jürgen Brehm, Herrn Reinhard Hohberger, Herrn Dr. Ralf Bracht, Herrn Werner Führich, Herrn Plamen Kiradjiev, Herrn Wolfram Andreas Richter, Herrn Andreas Konecny von der IBM sowie Herrn Michael Sambeth und Herrn Dr. Daniel Oberle von der SAP AG.

Bei Frau Dr.-Ing. Tania Tudorache (University of Stanford), Frau Dr. Cristina Osterhoff (Toll-Collect GmbH) und Herrn Heiner Klaasen-van Husen möchte ich mich für die Hinweise und Anregungen während der Endphase meiner Arbeit bedanken. Schließlich möchte ich mich bei meiner Freundin und Lebensgefährtin Frau Andrea Appelt und meiner Familie für die Geduld und die moralische Unterstützung in den letzten Jahren ganz herzlich bedanken.

Die vorliegende Arbeit wurde bei dem Wettbewerb „Bestes SOA-Industrieprojekt 2007“ vom Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (BITKOM) in der Kategorie „Cross-Vendor“ mit dem ersten Preis ausgezeichnet. Die IBM plant das vorgestellte Konzept im nächsten Release des WebSphere Integration Developers zu implementieren. Die Daimler AG plant beim Aufbau der firmeninternen Software-Architektur das Konzept als Bestandteil aufzunehmen.

Die hier vorliegende Dissertation wurde in LaTeX erstellt und setzt *Adobe Reader 7.0* (oder höher) zur fehlerfreien Darstellung der Arbeit voraus.

Abstrakt

Die folgende Arbeit definiert ein neues Konzept zur Flexibilisierung der Schnittstellenbeschreibungen in der Service-orientierten Architektur. Ziel ist die Verbesserung der Verfügbarkeit von Prozessabläufen bei geplanter Ausfallzeit. Das neue Konzept nutzt Ontologien zur unterstützten Suche äquivalenter Services (Kandidaten). Die Untersuchungen wurden in der Daimler AG, Abteilung „Methoden und Technologien“ in Sindelfingen durchgeführt.

Ausgangspunkt der Analyse ist die historische, organisatorische und firmenpolitische Verteilung der Prozessabläufe über IT-Systemgrenzen hinweg. Dabei wurden IT-Systeme redundant industrialisiert. Die redundant verteilte Steuerung der Prozessabläufe setzt an den IT-Systemgrenzen die Notwendigkeit menschlichen Eingreifens durch Know-how-Träger voraus. Dadurch entstehen potentielle Gefahrenquellen, die auf Übertragungsfehler an den IT-Systemgrenzen und (in manchen Fällen) auf die Unverfügbarkeit der Know-how-Träger zurückzuführen sind.

Ausgehend von diesem Tatbestand wurden mit Hilfe der Anwendung von SOA Konzepten zwei aufeinander aufbauende Softwarearchitekturen im Labor erarbeitet und auf Korrektheit bzw. Anwendbarkeit validiert. Der Prozessfluss, der im Unternehmen Daimler AG exemplarisch selektierten Geschäftsprozesse (Bestellanforderung und Auftragsplanung Getriebe), wurde in Form von Funktionen, Ereignissen, Rollen und Geschäftsobjekten durch Interviews erfasst. Mit Hilfe des ARIS SOA Designers wurde der Prozessfluss fachlich modelliert und in Anwendung der Top-Down-Vorgehensweise in die Business Process Execution Language transformiert. Nach einem erfolgreichen Import in den WebSphere Integration Developer wurde (nach technischer Anreicherung) der erzeugte Code zur Ausführung in den WebSphere Process Server ausgeliefert. SAP Discovery System wurde als Service-Provider eingesetzt und stellt zwei Services bereit. Hier handelt es sich um einen Service zur Erstellung und einen Service zur Änderung der Bestellanforderung. Für diese Services wurden plattformneutrale Schnittstellen in Form einer WSDL bereitgestellt, die in der Ablaufumgebung (WebSphere Process Server) aufgerufen und ausgeführt werden. Die Top-Down-Vorgehensweise inkl. einer Prozessänderung durch zeitnahes Einbinden eines neuen Service, wurde dem Management in einer Live-Demonstration präsentiert.

Die quasi-statische Bindung zwischen den einzelnen Prozessschritten und den potentiellen Service-Providern wurde als zentrales Problem identifiziert. Sie ermöglicht nur eine partielle Ermittlung des Service-Endpunkts (IP-Adresse, Protokoll und Port) und setzt eine fixe Schnittstellenbeschreibung voraus. Dies führt bei Unverfügbarkeit der Service-Provider zu einer Unterbrechung in der Ablaufumgebung. Menschliches Eingreifen durch Know-how-Träger ist notwendig, um den statischen Charakter der Schnittstellenverarbeitung aufzulösen. Die quasi-statische Bindung mangelt somit an einer flexiblen Schnittstellenverarbeitung bei einer möglichen Unverfügbarkeit der Service-Provider. Der in dieser Arbeit vorgestellte EMEO-Layer,

gebildet aus **E**nterprise Service Bus, **M**ediation, **E**nterprise Service Repository und **O**ntologien, schlägt die Definition einer logischen Schicht zur Identifizierung äquivalenter Services (Kandidaten) durch semantische semiautomatische Unterstützung in der Entwicklungsumgebung vor. Das minimal anzunehmende deklarative Wissen der Know-how-Träger über die fachlichen Anforderungen der Services wird mit Hilfe einer Beschreibungslogik formal beschrieben. Das wissenschaftliche Werkzeug Protégé, federführend entwickelt an der Stanford University, findet Anwendung im EMEO-Layer durch die Modellierung einer Ontologie in der Web Ontology Language. In Kombination mit der Inferenzmaschine RACER (Renamed Abox and Concept Expression Reasoner; Universität Hamburg) wurde ein neuartiger Algorithmus (Semantik-Service-Finder) als Java-Prototyp implementiert. Hiermit wird eine bewertete Liste potentieller äquivalenter Services (Kandidaten) in der Entwicklungsumgebung zurückgeliefert.

Zwei weitere wesentliche Elemente des EMEO-Layers sind erstens die anschließende Nutzung der bewerteten Liste und zweitens die Erstellung unabhängiger Pfade äquivalenter Services in Form einer Mediation im Enterprise Service Bus. Nach der Generierung des ablauffähigen Codes und der Ausführung im WebSphere Process Server wird die Mediation über externalisierte Informationen im Enterprise Service Repository gesteuert. Hiermit wird die quasi-statische Bindung zwischen den einzelnen Prozessschritten und den potentiellen Service-Providern (Service-Endpunkte) durch eine flexible Schnittstellenverarbeitung ersetzt.

Mit dem hier vorgestellten Konzept wird bei Ausfall eines Service automatisch ein alternativer Service gebunden und ausgeführt. Damit sind wesentliche Verbesserungen und finanzielle Ersparnisse möglich. Mit diesem Konzept werden die erwähnten potentiellen Gefahrenquellen minimiert, die Verfügbarkeit der Prozessabläufe verbessert und redundante Industrialisierungen der Services genutzt.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Service-orientierte Architektur (SOA) beschreibt ein Architekturmodell bzw. eine Software-Infrastruktur, in der die wesentlichen fachlichen Funktionen einer IT-Anwendung als Service repräsentiert werden. Kennzeichnend sind hier granulare und wiederverwendbare Services, die in neue Anwendungen integriert werden können. Zudem bestehen die Möglichkeiten, laufende Anwendungen durch den Austausch einzelner Services anzupassen, zu erweitern und zu optimieren. Zwingende Abhängigkeiten der monolithischen Architekturen zwischen bestimmten Client-Serverarchitekturen sind damit aufgelöst. In dieser Dissertation werden innovative und zukunftsweisende Möglichkeiten Softwarelösungen zu entwickeln dargestellt. Basierend auf den Prinzipien einer SOA wird, mit Blick auf ein internationales Unternehmen, eine Implementierung dieses neuen Architekturmodells in der Domäne Automotive bei der Daimler AG vorgestellt.

1.1 Problemstellung und Ziel der Arbeit

Die verteilte Steuerung der Prozessabläufe setzt die Notwendigkeit menschlichen Eingreifens durch Know-how-Träger an den IT-Systemgrenzen voraus. Die dadurch potentiell entstehenden Gefahrenquellen sind auf Übertragungsfehler an den IT-Systemgrenzen sowie (in manchen Fällen) auf Unverfügbarkeit der Wissensträger zurückzuführen. Durch Anwendung der Konzepte einer SOA wird die Steuerung der Prozessabläufe zentralisiert. Somit werden die potentiellen Gefahrenquellen, wie Übertragungsfehler an den IT-Systemgrenzen und die Unverfügbarkeit der Wissensträger eliminiert. Die quasi-statische Bindung zwischen den einzelnen Prozessschritten und den potentiellen Service-Providern ermöglicht eine partielle Ermittlung (IP-Adresse, Protokoll und Port) des Service-Endpunkts in der Ablaufumgebung und mangelt somit an einer flexiblen Schnittstellenverarbeitung bei einer möglicher Unverfügbarkeit der Service-Provider.

Diese Arbeit zeigt die heutigen Realisierungsmöglichkeiten in Form einer durchgängigen Top-Down-Vorgehensweise auf und schlägt den EMEO-Layer als einen semiautomatischen Ansatz zur Findung äquivalenter Services (Kandidaten) in der Entwicklungsumgebung vor. Schnittstellenbeschreibungen äquivalenter Services wer-

den auf die Geschäftsobjekte abgebildet und mit der Mediation verbunden. In der Ablaufumgebung ermöglicht die Lookup-Komponente eine Ermittlung des Service-Endpunktes in Form IP-Adresse, Protokoll und Port. Durch die Mediation werden die bereits bestimmten Schnittstellenbeschreibungen gebunden und ausgeführt. Die redundante Industrialisierung der Services wird genutzt, um die Verfügbarkeit der Prozessabläufe bei geplanter Ausfallzeit zu erhöhen. Technisch wurde der EMEO-Layer durch die semantische Anreicherung der plattformneutralen Servicebeschreibung, der Mediation im Enterprise Service Bus und der partiellen Implementierung semantischer Konzepte validiert.

1.2 Lösungsansatz

Initiativen wie *Bottom-Up Approach* [93], *Template Based Composition* [147], *OWL-S Composition Approach* [146], *SHOP2* [148], *METEOR-S* [2], *SEMAPLAN* [5] und *WSMO Approach* [144], basierend auf den semantischen Konzepten OWL-S [95], WSDL-S [4] und WMSO [126], forschen an einer loseren Kopplung zwischen den fachlichen Funktionen und der technischen Ausführung. Die beiden Initiativen *SWORD* [121] und *Plængine* [99] basieren nicht auf den semantischen Konzepten OWL-S, WSDL-S und WSMO. Die Grundlage von *SWORD* ist unabhängig von semantischen Konzepten und *Plængine* basiert auf einem integrierten Meta-Modell. Ziel aller aufgeführten Initiativen ist die Anreicherung der Services mit semantischen Informationen um die dynamische Einbindung von Services (in einen Prozess) in der Ablaufumgebung zu ermöglichen. Die Dissertation von Muhammad Ahtisham Aslam [3] unterstützt dieses Ziel durch das Abbilden technischer Aktivitäten auf semantische Konzepte in OWL-S in der Entwicklungsumgebung.

Der EMEO-Layer, gebildet aus **E**nterprise Service Bus (ESB), **M**ediation, **E**nterprise Service Repository (ESR) und **O**ntologien, definiert eine logische Schicht *(engl. Layer)* zur Identifizierung äquivalenter Services (Kandidaten) durch semantische semiautomatische Unterstützung in einer Service-orientierten Architektur. Durch die Einbindung der Konzepte einer SOA (*ESB*, *ESR*) in Kombination mit einem Entwurfsmuster aus der Objekt-orientierten Programmierung *Mediation* und *Ontologien* stellt der EMEO-Layer einen nicht erforschten neuen Ansatz dar.

Der EMEO-Layer basiert, wie *METEOR-S* und *SEMAPLAN*, auf dem semantischen Konzept WSDL-S, grenzt sich jedoch durch die konzeptuelle Integration des semiautomatischen Ansatzes in die Konzepte ESB und ESR sowie durch eine prototypische Implementierung in der Domäne Automotive von diesen ab. Der EMEO-Layer unterstützt keine nicht-funktionalen Anforderungen wie bspw. Antwortzeiten, geografische Lokationen etc., sondern Funktionen und Geschäftsobjekte der Funktionen (funktionale Anforderungen). Funktionen werden auf Serviceoperationen, Vor- (`P`) und Nachbedingung (`E`) auf semantische Konzepte abgebildet. Geschäftsobjekte sind die Eingabe- (`I`) und Ausgabeparameter (`O`) der Funktion und somit der Serviceoperation. Serviceoperationen werden ebenfalls auf semantische Konzepte abgebildet. Die funktionalen Anforderungen Eingabe- und Ausgabe-

parameter sowie Vor- und Nachbedingung werden in dieser Arbeit als `IOPE` *(engl. Input, Output, Precondition, Effect)* bezeichnet. Die Arbeit findet ihre Anwendung in der Problemklasse plattformübergreifender Prozesse der Domäne Automotive unter der Randbedingung der Verfügbarkeitsklasse XH (höchstverfügbar).

1.3 Aufbau der Arbeit

In der vorliegenden Arbeit werden in **Kapitel 2** die Grundlagen der Service-orientierten Architektur sowie Ontologien und semantische Konzepte als Basis zur Ermittlung äquivalenter Services (Kandidaten) im EMEO-Layer vorgestellt. Methoden wie die Architektur integrierter Informationssysteme, Technologien wie Web Services, Business Process Execution Language und Web Ontology Language sowie das Entwurfsmuster Mediator, die in den nächsten Kapiteln Verwendung finden, werden kurz zusammengefasst.

Die notwendigen fachlichen Anforderungen und Randbedingungen zur Anwendung der theoretischen Grundlagen in einer Service-orientierten Architektur werden in **Kapitel 3** beschrieben. Hier werden der fachliche Prozess Bestellanforderung der Verfügbarkeitsklasse N (normalverfügbar), die durchgängige Realisierung, begonnen bei der Modellierung bis hin zum ablauffähigen Prototyp sowie die abgeleitete generische Top-Down-Vorgehensweise (hier als TD+ bezeichnet) dokumentiert.

Die Erhöhung der Verfügbarkeitsklasse N (normalverfügbar) auf die höchste Verfügbarkeitsklasse XH (höchstverfügbar) stellt eine Erweiterung der nicht funktionalen Anforderung dar. **Kapitel 4** beschreibt einen neuen fachlichen Prozess der höchsten Verfügbarkeitsklasse XH (höchstverfügbar), die erneute Anwendung der Top-Down-Vorgehensweise TD+ und die Ergänzung um die Lookup-Komponente im Enterprise Service Bus.

In **Kapitel 5** werden die Vorteile und Grenzen einer Service-orientierten Architektur unter der Randbedingung der Verfügbarkeitsklasse XH in Kombination mit geplanter Unverfügbarkeit der Service-Provider erörtert. Dabei wurde die quasi-statische Bindung zwischen den einzelnen Prozessschritten und den potentiellen Service-Providern als zentrales Problem bei geplanter Ausfallzeit der Service-Provider identifiziert.

In dieser Arbeit wird in K**apitel 6** der EMEO-Layer vorgeschlagen, um durch einen semantischen semiautomatischen Ansatz (in der Entwicklungsumgebung) die Defizite der Service-orientierten Architektur bei geplanter Unverfügbarkeit der Service-Provider zu lösen. Der EMEO-Layer ersetzt die quasi-statische Bindung zwischen den einzelnen Prozessschritten und den potentiellen Service-Providern (Service-Endpunkte) durch eine flexible Schnittstellenverarbeitung.

Die Validierung des EMEO-Layers wird in **Kapitel 7** illustriert. Zur Validierung wird das notwendige exemplarische Szenario und ein lauffähiger Prototyp vorgestellt. Das Szenario besteht aus zwei (Anreicherung der Services und Externalisierung des Know-hows) und der Prototyp aus drei Teilelementen (Mediator in Java, Algorithmus Semantik-Service-Finder und Ablaufumgebung).

In **Kapitel 8** werden kausale Schlussfolgerungen abgeleitet sowie die Kernelemente des EMEO-Layers Mediation, Enterprise Service Bus, Enterprise Service Repository und Ontologien zusammengefasst. Der Ausblick zeigt weitere mögliche Entwicklungen auf Basis des EMEO-Layers auf.

Im **Anhang** werden Publikationen, fachliche Anforderungen, der Quell-Code der Prototypen, ein SLK in OWL, die komplette Ontologie in OWL und die Anreicherung semantischer Informationen der Service-Provider abgebildet.

Kapitel 2

Grundlagen

2.1 Service-orientierte Architektur

Die Service-orientierte Architektur (SOA) beschreibt ein Architekturmodell bzw. eine Software-Infrastruktur, in der die wesentlichen fachlichen Funktionen einer IT-Anwendung als Service repräsentiert werden. Kennzeichnend sind hier granulare wiederverwendbare Services, die in neue Anwendungen integriert werden können. Somit ist die SOA ein technologieneutrales Architekturkonzept basierend auf allgemein verwendbaren bzw. wiederverwendbaren Services.

In der Literatur wird die Service-orientierte Architektur nicht eindeutig definiert. Bieberstein definiert die Service-orientierte Architektur als einen Satz von Methoden zur Reduzierung bzw. Eliminierung der Enttäuschungen durch die Informationstechnologie (IT), eine Messgröße zur Quantifizierung des Unternehmenswert der IT und eine agile, wettbewerbsfähige Geschäftsumgebung: *„A set of business, process, organizational, governance, and technical methods to reduce or eliminate frustrations with IT and to quantifiably measure the business value of IT while creating an agile business environment for competitive advantage“* [16]. Die Definition von Erl basiert auf der Implementierung der Technologie Web Service (siehe Kapitel 2.1.6 auf Seite 27) und definiert technische Eigenschaften wie Offenheit, Erweiterbarkeit, Interoperabilität sowie die Möglichkeit zur Wiederverwendung von Services: *„Contemporary SOA represents an open, extensible, federated, composable architecture that promotes services-orientation and is comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services“* [42]. Krafzig definiert die Service-orientierte Architektur als Software-Architektur basierend auf den Schlüsselkonzepten: Repräsentierung der Anwendung, Service, Service Repository sowie Servicebus. Er definiert den Service bestehend aus Vertrag, einer oder mehreren Schnittstellen und seiner Implementierung: *„A Service-Oriented Architecture is a software architecture that is based on the key concepts of an application frontend, service, service repository, and service bus. A service consists of a contract, one or more interfaces, and an implementation“* [89]. Das World Wide Web Consortium (W3C) definiert die Service-orientierte Architektur als eine Menge von Komponenten, die aufgerufen

und deren Schnittstellenbeschreibung veröffentlicht und entdeckt werden können: *„A Service-Oriented Architecture is a set of components which can be invoked, and whose interface descriptions can be published and discovered“* [164]. Garter definiert die Service-orientierte Architektur als Software-Architektur bestehend aus einer Struktur von Schnittstellen, Implementierungen der Schnittstellen und Aufrufe der Schnittstellen sowie die Beziehung zwischen Service und Service-Consumer, die beide die Größe einer kompletten Geschäftsfunktion besitzen. Somit handelt es sich bei der Service-orientierte Architektur um Wiederverwendung, Kapselung, Schnittstellen und schließlich um Agilität: *„SOA is a software architecture that builds a topology of interfaces, interface implementations and interface calls. SOA is a relationship of services and service consumers, both software modules large enough to represent a complete business function. So, SOA is about reuse, encapsulation, interfaces, and ultimately, agility“* [97]. Software-Architekten und Forscher haben gemeinsam eine Definition als Richtlinie für die Daimler AG erstellt. Somit wird in der Daimler AG die Service-orientierte Architektur wie folgt verstanden:

Definition 1 *Eine **Service-orientierte Architektur (SOA)** beschreibt ein technologieneutrales Architekturkonzept basierend auf allgemeingültigen (wieder-)verwendbaren Services. Dieses Konzept der Software-Architektur repräsentiert eine oder mehrere Geschäftsfunktionen als Service. Die Beschreibung der Service-Schnittstelle ist plattformneutral. Service-Implementierungen sind wiederverwendbar, gekapselt und lose gekoppelt. Die Kommunikation mit den Services wird über eine standardisierte Infrastruktur realisiert [68].*

In der Literatur sowie in zahlreichen Praxisberichten wird die Service-orientierte Architektur mit der Technologie Web Service gleich gesetzt. Das aktuell abgeschlossene Forschungsprojekt der Daimler AG distanziert sich von der Gleichstellung der SOA mit der Technologie Web Services; es hebt aber Web Services als signifikante Technologie zur Realisierung einer SOA hervor (siehe Kapitel 2.1.6 auf Seite 27).

In diesem Forschungsprojekt wurden die Grundlagen der Service-orientierten Architektur erarbeitet. Gemeinsam mit den internationalen Unternehmen IDS Scheer AG, SAP AG und IBM Deutschland GmbH wurden vier SOA-Dimensionen (siehe Abbildung 2.1 auf Seite 17), ein internes Reifegradmodell *(engl. maturity model)* und die SOA-Roadmap beschrieben. Die Konzepte Enterprise Service Bus (siehe Kapitel 2.1.7 auf Seite 32) und Enterprise Service Repository (siehe Kapitel 2.1.8 auf Seite 34) wurden basierend auf dem generischen Modell (siehe Kapitel 2.1.3 auf Seite 22) definiert. Eine Aussage, bezogen auf die Anwendung der Konzepte einer SOA in einem Unternehmen, hat keinen booleschen Charakter. Somit besteht ein Bedarf zur Definition eines Reifegradmodells.

Hierzu wurde in Analogie zum Capability Maturity Model Integration (CMMI) [77] ein Reifegradmodell erstellt, das sich dem SOA Maturity Model (SOA MM) [150] der Sonic Software stark ähnelt. Ziel des Reifegradmodells und der SOA-Roadmap ist es nicht, die höchste Stufe zu erreichern, sondern die internen Aufwände und Effekte der SOA-Initiative jederzeit ermitteln zu können. Das Reifegradmodell definiert sechs Stufen (Stufe 0 bis Stufe 5), den erwarteten Effekt

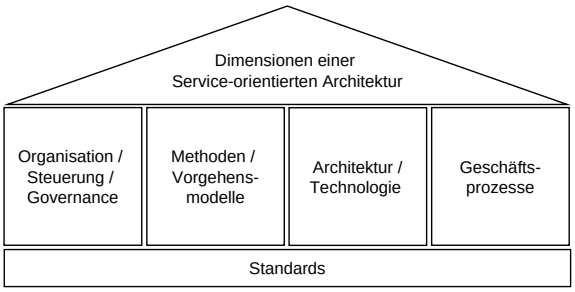

Abbildung 2.1: Die SOA-Dimensionen

auf Seiten der IT und auf Seiten des Fachbereichs. So definiert bspw. die Stufe 3 *gemeinsam genutzte fachliche Services auf Seite der IT* sowie *die fachliche Wiederverwendung der Services und eine fachliche Governance auf Seite des Fachbereichs.* Die definierte SOA-Roadmap basiert auf einer Matrix, in der die SOA-Dimensionen die X-Achse und das interne Reifegradmodell die Y-Achse darstellen. Die SOA-Roadmap skizziert Maßnahmen in allen vier SOA-Dimensionen, um den geplanten Reifegrad zu erreichen. Das Reifegradmodell sowie die SOA-Roadmap sind als vertraulich eingestuft und werden somit in der vorliegenden Arbeit nicht näher beschrieben. Die vier SOA-Dimensionen werden in Abbildung 2.1 illustriert und im Folgenden erläutert:

- Die erste Dimension *Organisation / Steuerung / Governance*, im Folgenden als *Governance* bezeichnet, betrachtet das Service-Portfolio-Management (siehe Kapitel 2.1.9 auf Seite 35), das Service-Lebenszyklus-Management, die Geschäftsdomänenstruktur, die Einhaltung der Vorgaben *(engl. Compliance)*, das SOA-Bewusstsein *(engl. SOA-Awareness)*, Key Performance Indikatoren (KPI's), Verrechnungsmodelle sowie Budget- und Ressourcenzuteilung.

- Die zweite Dimension *Methoden / Vorgehensmodelle* betrachtet die Verwaltung des Enterprise Service Repository's, die Orchestrierung der Services und Prozesse, Methoden zur Fachklassen- und Servicemodellierung, die Servicebeschreibung, Business Process Management (BPM) und Business Activity Monitoring (BAM), Ablagestrukturen sowie ein Konzept zur Geschäftsprozessmodellierung.

- Die dritte Dimension *Architektur / Technologie* betrachtet die Implementierung eines Enterprise Service Repository's, eine Security-Infrastruktur, die technische Service-Architektur, ein technisches SOA-Rahmenwerk *(engl. SOA-Framework)* sowie die Workflow-Engines.

- Die vierte und letzte Dimension *Geschäftsprozesse* betrachtet die Ableitung der Fachklassen und Services aus Domänen und Prozessen, den IT-Bebauungsplan, Optimierungen der Prozessmodelle, domänenübergreifende Prozessmodelle und das fachliche KPI-Monitoring.

Alle vier Dimensionen basieren auf dem Fundament der Standards. Dabei konzentriert sich das Unternehmen Daimler AG auf die Schnittmenge der Standards aller strategischen Lieferanten. Der Tenor dieser Arbeit liegt im Aufzeigen der heutigen Realisierungsmöglichkeiten einer SOA in einem internationalen Unternehmen der Domäne Automotive und fokussiert die SOA-Dimension *Architektur/Technologie*.

2.1.1 Ziele einer SOA

Die Globalisierung der Unternehmen ist geprägt durch permanente Marktveränderungen und unterliegt einem starken Kostendruck. Heterogene IT-Landschaften bilden sich u.a. durch Fusionen und Übernahmen *(engl. Mergers and Acquisitions)* sowie durch historisch wachsende Unternehmen [16]. Charakterisierend für heterogene IT-Landschaften sind bspw. COBOL-Anwendungen *(engl. Legacy-Systeme)*, J2EE-Anwendungen und EAI-Lösungen [89]. Permanente Veränderungen am Markt erfordern eine höhere Flexibilität der IT-Landschaften sowie ein verbessertes „Time-to-Market"-Verhalten der Unternehmen. Die aktuelle Studie *Geschäftsmodelle 2010* [81] von Kagermann und Oesterle prognostiziert einen verstärkten Bedarf an Flexibilität der Unternehmen. Mangelnde Flexibilität ist nicht auf fehlende Interoperabilität, sondern auf das Gestaltungsprinzip der festen Kopplung zwischen Geschäftsprozessen und IT-Anwendungen zurückzuführen [171]. Indikatoren dafür sind zeit- und kostenaufwändige Migrationsprojekte, die teilweise auch an der fachlichen Komplexität scheitern. Kennzeichnend für den hohen Kostenaufwand bei Erweiterung oder Ablösung einer IT-Anwendung sind, fachliche Integrationsanforderungen sowie eine fehlende Transparenz der Geschäftsprozesse. Die Service-orientierte Architektur soll durch das Gestaltungsprinzip der losen Kopplung und der Durchgängigkeit fachlicher Spezifikation zur IT-Applikation (Top-Down) die Flexibilität der IT-Landschaft erhöhen um das „Time-to-Market"-Verhalten zu verbessern [42]. Durch die Wiederverwendung der fachlichen Funktionalität (präsentiert durch Services) sollen die Kosten zur Realisierung neuer Anforderungen reduziert werden [79] [171]. Somit können zwei primäre Ziele der Service-orientierten Architektur identifiziert werden:

- Anwendung des Gestaltungsprinzip der losen Kopplung zur unterstützenden Flexibilisierung der IT, um das geforderte „Time-to-Market"-Verhalten zu verbessern.
- Wiederverwendung fachlicher Funktionalität, präsentiert durch Services, um Aufwände zur Realisierung neuer Anforderungen zu reduzieren.

2.1.2 Gestaltungsprinzip der losen Kopplung

Gestaltungsprinzipien einer Service-orientierten Architektur sind neben einer Wiederverwendung der Services als Blackbox *(engl. black box reuse)*, die technologieneutrale Beschreibung der Schnittstelle, die Abbildung fachlicher Funktionen durch einen oder mehrere Services sowie die lose Kopplung zwischen Geschäftsprozessen (fachlicher Funktion) und IT-Anwendungen (repräsentiert als eine Anzahl von

Services) [139]. Das Gestaltungsprinzip der losen Kopplung findet Anwendung im EMEO-Layer (siehe Kapitel 6 auf Seite 85) und wird im Folgenden diskutiert. Die lose Kopplung kann auf fachlicher und technischer Ebene stattfinden. Bspw. kann ein Geschäftsobjekt `Adresse`, im Gegensatz zur technischen Definition mit `Integer` und `String`, durch die fachlichen Eigenschaften `Straße`, `Hausnummer`, `Postleitzahl` und `Ort` definiert werden. Woods versteht die Abbildung der Services mit Blackbox-Charakter als lose Kopplung, da die Implementierung für den Service-Consumer, solange die fachlichen und technischen Anforderungen erfüllt werden, eine untergeordnete Rolle spielt [171]. Erl versteht die Abstraktion der zu Grunde liegenden Implementierung auf die Serviceoperation der Serviceschnittstelle als feste vertragliche Bindung zwischen Provider und Consumer, aber technisch als lose Kopplung zwischen bereitstellendem und aufrufendem IT-System [42].

Bieberstein unterscheidet zwischen den beiden Gestaltungsprinzipien: loser Kopplung und Bindekraft *(engl. cohesion)*. Die Faktoren Zeit *(engl. time)*, Protokoll *(engl. protocol)*, Datenformat *(engl. format)*, Vertrag *(engl. contract)*, Programmiersprache *(engl. language)*, Plattform *(engl. platform)* und Standort *(engl. location)* (in zufälliger Reihenfolge) werden dem Gestaltungsprinzip der losen Kopplung zugeordnet. Datentypen *(engl. types)*, Datenstruktur *(engl. structure)*, Verbindungen *(engl. relationships)*, Logik *(engl. logic)* und Vertrauen *(engl. trust)* (mit steigender Bindekraft) werden der Bindekraft zugeordnet [16].

Ebene	**Feste Kopplung**	**Lose Kopplung**
Physikalische Verbindung	Punkt-zu-Punkt	Mediation
Kommunikationsart	synchron	asynchron
Kommunikationsformat	technisch	semantisch
Interaktionsverhaltensmuster	objektorientiert	datenzentriert
Kontrolle der Prozesslogik	zentral	dezentral
Service-Bindung	statisch	dynamisch
Plattform	spezifische Datentypen	neutrale Datentypen
Transaktionsverhalten	Zwei-Phasen-Commit	Kompensation
Datenmodell	allgemeingültige `<complexTypes>`	allgemeingültige `<simpleTypes>`
Auslieferung	simultan	unterschiedlich
Versionsverwaltung	explizites Update	implizites Update

Tabelle 2.1: Lose Kopplung nach Krafzig [89] und Josuttis [79]

Krafzig grenzt die Ebenen *physikalische Verbindung (engl. physical connection)*, Kommunikationsart *(engl. communication style)*, Kommunikationsformat *(engl. type system)*, Interaktionsverhaltensmuster *(engl. interaction pattern)*, Kontrolle der Prozesslogik *(engl. control of process logic)*, Service-Bindung *(engl. binding)* und Plattform *(engl. platform)* zwischen fester und loser Kopplung voneinander ab [89]. Josuttis erweitert die Abgrenzung von Krafzig um die Ebenen Datenmodell *(engl. data model)*, Transaktionsverhalten *(engl. transactionality)*, Auslieferung *(engl. deployment)* sowie Versionsverwaltung *(engl. versioning)* (ohne Anspruch auf Vollständigkeit) [79] und wird in Tabelle 2.1 auf Seite 19 illustriert.

Auf der Ebene der *physikalischen Verbindung* wird die *Punkt-zu-Punkt*-Verbindung als feste und die *Mediation* (siehe Kapitel 2.9 auf Seite 38) als lose Kopplung verstanden. Punkt-zu-Punkt bezeichnet den direkten Aufruf der physikalischen Adresse des Services und somit den Service-Endpunkt (siehe Kapitel 2.1.6 auf Seite 30). Josuttis unterscheidet zwischen den zwei Arten der Ermittlung des Service-Endpunktes: *vor dem Senden* des Aufrufs und *nach dem Senden* des Aufrufs. Die erste Art wird funktional als Broker bezeichnet, die zweite nutzt symbolische Namen, um bspw. im Enterprise Service Bus aufgrund technischer Anforderungen den symbolischen Namen durch den Service-Endpunkt zu ersetzen. Auf der Ebene *Kommunikationsart* wird die *synchrone* Kommunikation als feste und die *asynchrone* Kommunikation als lose Kopplung verstanden. Bei der asynchronen Kommunikation ist, im Gegensatz zur synchronen Kommunikation, der Austausch von Nachrichten zwischen Provider und Consumer zeitunabhängig. Somit führt bei der asynchronen Kommunikation der Sender (Provider) seine Ablauflogik weiter aus, ohne auf eine mögliche Antwort des Empfängers (Consumer) zu warten. Im Gegensatz dazu wartet bei der synchronen Kommunikation der Sender (Provider) auf eine Antwort und setzt nach Eingang mit seiner Ablauflogik fort [69]. Technisch kann die asynchrone Kommunikation der Nachrichten bspw. durch Warteschlangen *(engl. Queues)* mit MQSeries [166] realisiert werden. Die *Mediation* ist eine Form der losen Kopplung, hat ihren Ursprung in der Objekt-orientierten Programmierung und wird in Kapitel 2.9 auf Seite 38 ausführlich beschrieben.

Auf der Ebene *Kommunikationsformat* wird die *technische* Beschreibung als feste und die *semantische* Beschreibung als lose Kopplung verstanden. So kann bspw. die Operation einer Serviceschnittstelle durch die Signatur syntaktisch beschrieben werden. Durch explizite Datentypen der Ein- (`I`) und Ausgabeparameter (`O`) der Operation findet eine feste Kopplung zwischen Provider und Consumer statt. Im Gegensatz dazu stellt die Definition eines Datenaustauschformates auf syntaktischer und semantischer Ebene eine lose Kopplung zwischen Provider und Consumer dar. Ein weiteres Beispiel der festen bzw. losen Kopplung des Kommunikationsformates zwischen Provider und Consumer ist die Implementierung der Geschäftsobjekte. Die feste Kopplung definiert harmonisierte Geschäftsobjekte als Basis zur Kommunikation, die bei Bedarf erweitert werden. Die lose Kopplung dagegen bildet *(engl. mapping)* die heterogenen Geschäftsobjekte aufeinander ab.

Auf der Ebene *Interaktionsverhaltensmuster* wird die Navigation durch Objektbäume (*objektorientiert*) als feste und unabhängige Nachrichten (*datenzentriert*) als lose Kopplung verstanden. In einer Infrastruktur verteilter Objekte wie bspw. der Common Object Request Broker Architecture (CORBA) [117], die einen Object Request Broker (ORB) als zentrale Komponente zum Datenaustausch vorsieht, wird u.a. das Serverobjekt aufgrund der angegebenen Objektreferenz lokalisiert, die Eingabeparameter an den Server übermittelt und die Rückgabeparamter an den Client zurückgeliefert [40]. Diese Interaktion zwischen verteilten Objekten impliziert die Navigation durch teilweise komplexe Objektbäume, eine adäquate Vorgehensweise bei der Navigation durch die Objektbäume und das Verständnis der logischen Struk-

tur der Objekte. Die dadurch entstanden Abhängigkeiten werden als feste Kopplung zwischen Provider und Consumer verstanden.

Auf der Ebene *Kontrolle der Prozesslogik* wird die *zentrale* Steuerung als feste und die *dezentrale* Steuerung als lose Kopplung verstanden. Die zentrale Steuerung der Prozesslogik reduziert die Komplexität (bezogen auf Persistenz, Konsistenz etc.) und koppelt bspw. Unterprozesse fest aneinander. Die dezentrale Steuerung der Prozesslogik ermöglicht durch Ereignisse *(engl. events)* eine lose Kopplung, wie bspw. in einer Ereignisgesteuerten Architektur *(engl. event-driven architecture (EDA))* zwischen den Unterprozessen, erschwert aber die Ermittlung eines definierten konsistenten Prozessstatus'.

Auf der Ebene der *Service-Bindung* wird die *statische* Bindung zwischen Provider und Consumer als feste und die *dynamische* Bindung als lose Kopplung verstanden. Die statische Bindung wird durch den direkten Aufruf des Service-Endpunkts realisiert. Die dynamische Bindung nutzt in der Entwicklungsumgebung symbolische Namen und ersetzt diese (in der Ablaufumgebung) durch die jeweils ermittelte Schnittstelle. Dabei werden die Einzelschritte: finden, auswählen, ggf. orchestrieren, binden und ausführen durchlaufen. Somit ist die dynamische Bindung eine Erweiterung der Mediation (physikalische Verbindung) und stellt einen hohen Grad einer loser Kopplung zwischen Provider und Consumer dar.

Auf der Ebene der *Plattform* wird die Verwendung *spezifischer Datentypen* als feste und die Verwendung *neutraler Datentypen* als lose Kopplung verstanden. Plattformspezifische Datentypen sind bspw. Blob (Binary Large Object) oder Packed Decimal, die Provider bzw. Consumer fest an die Plattform koppeln. Unter plattformneutralen Datentypen werden alle durch XML bereitgestellte Datentypen verstanden (vgl. Kapitel Abbildung der Datentypen auf Seite 31).

Auf der Ebene des *Transaktionsverhaltens* wird die Verwendung des *Zwei-Phasen-Commits* als feste und die Verwendung der *Kompensation (engl. compensation)* als lose Kopplung verstanden. Das Zwei-Phasen-Commit stellt die Transaktionssicherheit (atomar, konsistent, isoliert, dauerhaft (ACID)) über verteilte Ressourcen (bspw. Datenbanken) durch einen globalen Transaktionsmanager sicher und bindet somit fest. Die Kompensation definiert fachliche Undo-Operationen mit dem Ziel die vorangegangene Aktion oder Aktionen rückgängig zu machen. Der Mehraufwand durch die Implementierung der Undo-Operationen soll durch eine sinkende Latenzzeit beim Nachrichtenaustausch und eine höhere Flexibilität beim Zurücksetzen einer Teil-Transaktion kompensiert werden [151]. Die Ebenen *Datenmodell*, *Auslieferung* und *Versionsverwaltung* spielen eine untergeordnete Rollen und werden folglich nicht weiter diskutiert.

Somit konnten folgende Erkenntnisse des Gestaltungsprinzip der losen Kopplung in einer Service-orientierten Architektur identifiziert werden: Bei der losen Kopplung ist generell zwischen Entwicklungsumgebung und Ablaufumgebung zu unterscheiden. Die lose Kopplung unterstützt die Flexibilität der Prozesse, findet auf technischer und fachlicher Ebene statt und muss mit Mehraufwand implementiert werden.

2.1.3 Generisches Modell (logische Sicht)

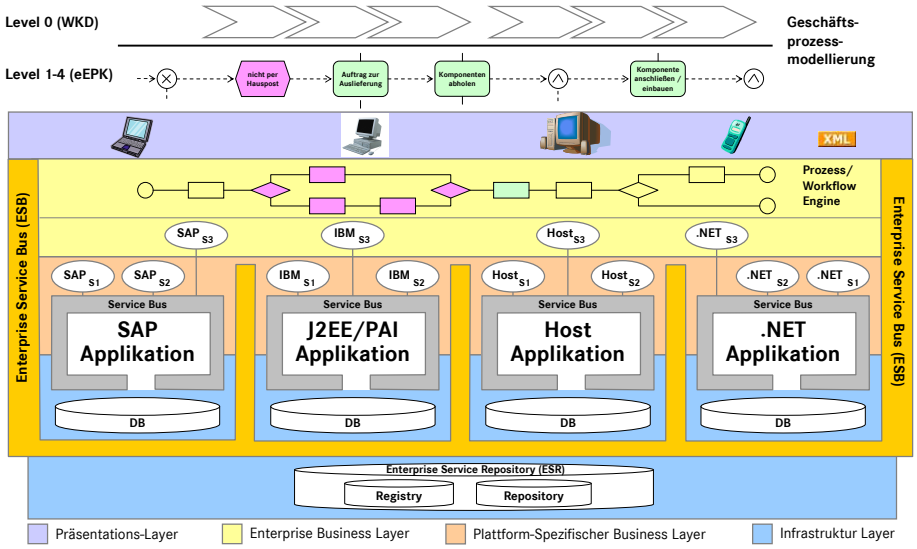

Abbildung 2.2: Generisches Modell (logische Sicht)

Abbildung 2.2 gibt ein generisches Modell einer SOA mit der Anpassung an die bei der Daimler AG vorhandenen strategischen Plattformen wieder. Es illustriert die logische Sicht auf eine mögliche SOA-Landschaft. Die strategischen Plattformen sind SAP, J2EE/PAI, Host/Mainframe und .NET; wobei die Pro-Aktive-Infrastruktur (PAI) ein internes Projekt zur Integration der IBM-Produkte in die Security-Infrastruktur der Daimler AG darstellt. Der *Infrastruktur Layer* ist die logische Sicht auf Sicherheitsaspekte, wie bspw. Autorisierung, Authentifizierung, Verschlüsselung und Logging. Der *plattform-spezifische Business Layer* ist die logische Sicht auf plattformspezifsche Services, wie bspw. BAPI's (Business Application Programming Interface). Der *Enterprise Business Layer* unterteilt sich in zwei Bereiche, den Prozessfluss im oberen und den plattformneutralen Bereich im unteren Teil. Der Prozessfluss-Bereich stellt die zentrale Steuerung ausführbarer Geschäftsprozesse dar. Der plattformneutrale Bereich ist die logische Sicht auf plattformneutrale Services, wie bspw. Web Services (siehe Kapitel 2.1.6 auf Seite 27) beschrieben durch WSDL-Dateien. Der *Präsentations-Layer* stellt die grafische Benutzeroberfläche (Fat-Clients, Rich-Clients, Thin-Clients) dar und ist somit die für den Anwender sichtbare Repräsentation des abgebildeten fachlichen Prozesses. Die beiden letzten Bereiche, erweiterte Ereignisgesteuerte Prozessketten und Wertschöpfungskettendiagramme, bilden den Bereich der fachlichen Anforderungen durch das Domänenmodell und grafisch modellierte Kernprozesse der Daimler AG ab.

Die Diskussion über die Darstellung der Prozessabläufe in Petri-Netzen [122] oder in der Unified Modeling Language (UML) [112] ist nach Meinung des Verfassers obsolet, da die Durchdringung der eEPK-Methode (erweiterte Ereignisgesteuerte Prozessketten) im Fachbereich signifikant ist.

2.1.4 Architektur integrierter Informationssysteme

ARIS-Haus

Architektur integrierter Informationssysteme (ARIS) [133] wurde von Scheer in den neunziger Jahren theoretisch erarbeitet. Seit 1993 wird das ARIS-Konzept durch mehrere Tools praxisnah unterstützt. Das ARIS-Haus im Anhang D auf Seite 157 stellt die unterschiedlichen Sichten (Daten-, Steuerung-, Funktion- und Organisationssicht) auf eine betriebswirtschaftliche Problemstellung dar. Jede dieser Sichten besteht aus dem Fachkonzept, dem DV-Konzept und der Implementierung. Generell können diese Sichten in Detaillierungsgrade unterteilt werden. Abhängig von der Sicht werden passende Modellierungsmethoden empfohlen. Bspw. für die Datensicht das Entity-Relationship-Modell (ERM), für die Steuerungssicht die erweiterte Ereignisgesteuerte Prozessketten (eEPK) und für die Funktionssicht der Funktionsbaum. Ein *Wertschöpfungskettendiagramm (WKD)* stellt Prozesse der strategischen oder oberen Unternehmensebenen dar. Ein Modell wird als Abbildung der betriebswirtschaftlichen Realität verstanden und besteht im Wesentlichen aus den Merkmalen: Funktion, Daten, Organisationseinheit, Ereignis, Ressource und Leistung. Scheer definiert einen Geschäftsprozess als *„...die Abfolge von Funktionen, die zusammengenommen ein Produkt und/oder eine Dienstleistung für einen internen oder externen Kunden schaffen“* [133]. Somit wird aus Kundensicht eine Wertsteigerung erzeugt.

Erweiterte Ereignisgesteuerte Prozessketten (eEPK)

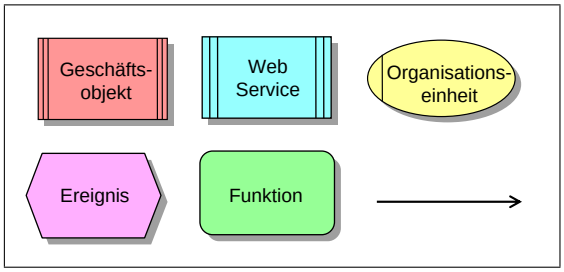

Abbildung 2.3: Übersicht der eEPK-Symbole

Folgend werden die für diese Arbeit relevanten Symbole (siehe Abbildung 2.3) einer *Erweiterte Ereignisgesteuerte Prozessketten (eEPK)* zur grafischen Abbildung von Prozessen, wie Geschäftsobjekt, Web Service, Ereignis, Funktion, Organisationseinheit und Kanten bzw. gerichtete Kanten, näher beschrieben.

Geschäftsobjekt. *Geschäftsobjekte (engl. business objects)* stellen die logische Sicht auf eine Menge von Daten (Cluster) in einem Datenmodell dar. Somit kann jede Art von Information durch ein *Geschäftsobjekt* wie bspw. Dokumente, Eingaben, Ausgaben etc. beschrieben werden. In dieser Arbeit symbolisiert das *Geschäftsobjekt* die Ein- (`I`) bzw. Ausgaben (`O`) einer oder mehrerer Funktionen.

Web Service. In der aktuellen Version der ARIS-Methode eEPK existieren mehrere Symbole, um die Implementierung einer Funktion durch einen *Web Service* auszudrücken. Die Abbildung 2.3 auf Seite 23 bildet das in dieser Arbeit verwendete Symbol ab. In Kombination mit dem Symbol einer *Funktion* drückt das Symbol *Web Service* die Ausführung der *Funktion* aus.

Funktion. Funktionen werden als Aufgaben *(engl. tasks)* verstanden, die eine körperliche oder geistige Aktivität darstellen. Modelle in der ARIS-Methode eEPK legen ihren Fokus auf die Dokumentation des Prozessziels. Im Gegensatz dazu wird in dieser Arbeit die Beschreibung des Prozessverhaltens fokussiert. Eine *Funktion* repräsentiert eine Aktivität, die Ein- (`I`) und Ausgabeparameter (`O`), die ggf. transformiert und verarbeitet werden müssen. Funktionen können solange gesplittet werden, bis der Fachbereich die *Funktion* als Aktivität wahrnimmt.

Ereignis. Ein *Ereignis* symbolisiert das Eintreten einer bestimmten Bedingung oder das Erreichen eines bestimmten Status. *Ereignisse* können Funktionen auslösen oder durch Funktionen ausgelöst werden. In dieser Arbeit wird das *Ereignis* zur Bestimmung der Vor- (`P`) und Nachbedingungen (`E`) einer Funktion genutzt.

Organisationseinheit. Die *Organisationseinheit* ist symbolierend für eine bestimmte Menge der aktuellen Mitarbeiter im Unternehmen und kann eine notwendige menschliche Handlung ausdrücken.

Kante. Eine *Kante* verbindet Geschäftsobjekte, Ereignisse, Funktionen, Web Services und Organisationseinheiten miteinander. Eine spezielle Form der Kante ist die *gerichtete Kante*, dargestellt durch einen Pfeil. Sie zeigt bspw. die Flussrichtung zwischen Geschäftsobjekt und Funktion. Kanten und gerichtete Kanten können bspw. die Werte: *ist Eingabe für*, *hat Ausgabe*, *ändert*, *archiviert*, *erstellt*, *löscht*, *verteilt*, *erzeugt* etc. annehmen.

Abbildung 2.4: Konnektoren (XOR, UND, ODER) in einer eEPK

Prozessabläufe, dargestellt durch *Ereignis* und *Funktion*, sind selten linear und benötigen somit logische Verknüpfungen, sog. *Konnektoren*. Abbildung 2.4 illustriert die drei Grundformen logischer Konnektoren. Konnektoren teilen den Kontrollfluss und führen ihn wieder zusammen. Sie entsprechen ihrem jeweiligen Äquivalent aus der Logik. Der Konnektor XOR bietet die Möglichkeit alternative Abläufe, der Konnektor UND (Konjunktion) bietet die Möglichkeit parallele Abläufe und der der Konnektor ODER (Adjunktion) bietet die Möglichkeit parallele und alternative Abläufe grafisch darzustellen (vgl. Junktoren in Kapitel 2.3.1 auf Seite 39).

Generell wird zwischen Ereignis-Verknüpfungen und Funktions-Verknüpfungen unterschieden. Ereignis-Verknüpfungen verbinden durch einen Konnektor (XOR, UND, ODER) zwei oder mehrere Ereignisse miteinander. Funktions-Verknüpfungen verbinden ebenfalls durch einen Konnektor zwei oder mehrere Funktionen miteinander. Ermittelt man die Anzahl der prinzipiell möglichen Verknüpfungen (Ereignis, Funktion) mit den Konnektoren (XOR, UND, ODER) und beschränkt die Anzahl der Ereignisse (E) bzw. Funktionen (F) auf zwei, so erhält man zwölf Möglichkeiten ($F_1 \rightarrow E_1 \wedge E_2$; $E_1 \wedge E_2 \rightarrow F_1$; $E_1 \rightarrow F_1 \wedge F_2$; $F_1 \wedge F_2 \rightarrow E_1$; etc). Die beiden Funktions-Verknüpfungen $E_1 \rightarrow F_1\ XOR\ F_2$ und $E_1 \rightarrow F_1 \vee F_2$ (der prinzipiell zwölf Möglichkeiten) sind methodisch nicht zulässig, da in beiden Fällen das Ereignis eine Entscheidung treffen müsste, generell Ereignisse aber nicht entscheiden können. Betrachtet man bspw. den Konnektor ODER verbunden mit vier eingehenden und einer ausgehenden Funktion in der Ablaufumgebung, so sind mehrere unterschiedliche Interpretationen möglich [34].

2.1.5 Business Process Execution Language (BPEL)

Die *Business Process Execution Language for Web Services (BPEL4WS)* [6] Version 1.1, im Folgenden als BPEL 1.1 bezeichnet, ist eine Vereinigung der Spezifikationen XLANG [160] von Microsoft und Web Services Flow Language (WSFL) [92] von IBM. Standardisiert durch das technische Komitee OASIS (Advancement of Structured Information Standards) [24] ermöglicht BPEL 1.1 die Definition abstrakter sowie ausführbarer Prozessdefinitionen durch die Aufrufe von Web Services (siehe Kapitel 2.1.6 auf Seite 27). Eine abstrakte Prozessdefinition kann bspw. zu Dokumentationszwecken genutzt werden. Ausführbare Prozessdefinitionen werden erstellt, um sie in eine Workflow-Engine auszuliefern *(engl. deployment)*.

Die Implementierung einer Workflow-Engine interpretiert ausführbare Prozessdefinitionen durch die Erstellung einer oder mehrerer Prozessinstanzen in der Ablaufumgebung. So ist die Erstellung ausführbarer Prozessdefinitionen mit dem Subjektorientieren Ansatz [49] und auch mit BPEL 1.1 möglich. Der Subjektorientiere Ansatz beschreibt die Subjekte „Prozessrollen“, „Prozessbeteiligte“ und „Akteure“ sowie ihre zentralen Eigenschaften „miteinander reden“ (Kommunikation), „zuhören“ (Information) und „handeln“ (Aktion). Analog zur deutschsprachigen Grammatik wird die Prozessdefinition als vollständiger Satz formuliert: Ein Mitarbeiter (Subjekt) erstellt (Prädikat) eine Bestellanforderung (Objekt). In der Ablaufumgebung sendet ein Subjekt Nachrichten an andere Subjekte, empfängt Nachrichten von anderen Subjekten oder führt Aktionen aus. Unabhängig von den jeweiligen Organisationen definiert jedes Subjekt seinen eigenen Ablauf. Die Koordination und Synchronisation findet ebenfalls über Nachrichten statt.

Im Gegensatz zum Subjektorientieren Ansatz hat die Prozessdefinition in BPEL 1.1 eine zentrale Prozesskontrolle. Eine komplette BPEL-Definition besteht aus zwei (BPEL- und WSDL-Datei) auf XML basierende Dateien. Die BPEL-Datei beschreibt den Prozessfluss inkl. Datenmanipulation sowie alle ausgehenden Aufru-

fe (Web Services). Die WSDL-Datei (siehe Kapitel 2.5 auf Seite 30) beschreibt die Schnittstelle des Prozesses. Somit ist der BPEL-Prozess selbst ein Web Service.

Muster *(engl. Pattern)* beschreiben beständig wiederkehrende Entwurfsprobleme und erläutern deren Problemlösung. Ziel ist es, die Entwurfsmuster als wiederverwendbare Vorlagen zu etablieren, um Synergien zu erzielen. Bekannt unter dem Akronym GoF („Gang of Four“) [56] wurden dreiundzwanzig Entwurfsmuster (vgl. Kapitel 2.9 auf Seite 38) für spezielle Herausforderungen in der Objekt-orientieren Software-Entwicklung erstellt. Analog zur Objekt-orientierten Software-Entwicklung wurden von den P4 („Process Four“) [163] zwanzig Workflow-Muster *(engl. Workflow Patterns)* wie bspw. Sequenz *(engl. Sequence)*, paralleler Prozess-Splitt *(engl. Parallel Split)*, exclusives Oder *(engl. Exclusive Choice)* etc. erstellt. Die Workflow-Muster der P4 dienen heute als Grundlage zur Bestimmung der Mächtigkeit der Modellierungssprachen ausführbarer Prozesse [63].

Im Folgenden werden die grundlegenden Sprachelemente von BPEL 1.1 kurz beschrieben. Das Element `<receive>` ist eine Aktivität und erwartet auf eine eingehende Nachricht *(engl. Message)* via SOAP (siehe Kapitel 2.1.6 auf Seite 28). `<pick>` ähnelt `<receive>` und unterscheidet sich jedoch in der Erwartung mehrerer eingehender Nachrichten. `<reply>` ist eine Aktivität, die synchrone Nachrichten (angestoßen von einem `<receive>`) seinem Aufrufer zurücksendet. `<invoke>` ruft einen Web Service (synchron oder asynchron) auf. `<assign>` kopiert bzw. transformiert Daten von Variable a nach Variable b. `<terminate>` sendet eine Fehlermeldung *(engl. exception)* und beendet die laufende Prozessinstanz sofort. `<wait>` ermöglicht analog zu Thread.sleep() in Java, das Warten einer Prozessinstanz. `<compensate>` ermöglicht im Fehlerfall das Ausführen von Undo-Operationen. Im Gegensatz zur Transaktionssicherheit von Datenbanken (ACID) müssen in BPEL 1.1 Undo-Operationen definiert und implementiert werden. `<validate>` prüft Variablen gegen XML-Schema Definitionen. `<sequence>` definiert sequentielle und `<flow>` parallele zu verarbeitende Aktivitäten. `<while>`, `<foreach>` und `<if>` verhalten sich wie die analogen Konstrukte *while*, *foreach* und *if* in Java.

BPELj [18] ist eine standardisierte Java-Erweiterung von BPEL 1.1, mit dem Ziel Java-Code im XML-basierenden BPEL-Code zu nutzen. Die Erweiterung des Standards BPEL 1.1 um menschliche Aktivitäten *(engl. Human Tasks (HT))* [41] wurde durch BPEL4People [73] standardisiert. Unter ETTK (Emerging Technologies Toolkit) wird eine Unterstützung der UML2BPEL-Transformation durch die IBM in Form einer Open-Source Java-Entwicklung vorgestellt.

2.1.6 Web Service

Definitionen

In der Literatur wird Web Service [29] nicht eindeutig definiert und ist somit terminologisch unscharf. So definiert beispielsweise SUN einen Web Service als Software-System mit dem Ziel einer vollständig kompatiblen *(engl. interoperable)* Maschine-zu-Maschine Kommunikation über ein Netzwerk in einer heterogenen Umgebung: *„Web services are software systems designed to support interoperable machine-to-machine interaction over a network in a heterogeneous environment“* [142]. Zimmermann definiert einen Web Service als auf eine XML-Nachrichten basierende verfügbare Kollektion von Operationen: *„A Web Service implements an interface that describes a collection of network-accessible operations through standard XML messaging“* [173]. Das World Wide Web Consortium (W3C) definiert die Technologie Web Service mittels der Spezifikationen XML [12], SOAP [60] und WSDL [21] wie folgt:

Definition 2 *A **Web service** is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machineprocessable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with XML serialization in conjunction with other Web-related standards. [168]*

Uniform Resource Identifier (URI)

Der *Uniform Resource Identifier (URI)* [27] besteht aus einer Sequenz von Zeichenketten und identifiziert eine abstrakte oder eine physikalische Ressource. Die Syntax [13] definiert eine allgemein gültige Obermenge aller gültigen URIs mit dem Ziel, jeden möglichen Bezeichner, der keine Kenntnisse des spezifischen Schemata besitzt, parsen zu können. Der *Internationalized Resource Identifier (IRI)* [39] besteht ebenfalls aus einer Sequenz von Zeichenketten und grenzt sich zu URI ab, indem eine IRI nur aus dem Zeichensatz Unicode/ISO 10646 bestehen kann. Uniform Resource Locator (URL) [15] ist eine Untermenge der URI und identifiziert eine Ressource über das verwendete Netzwerkprotokoll (http, ftp, telnet, file etc.). Im Folgenden werden URIs exemplarisch illustriert: *telnet://192.0.2.16:80/*, *tel:+1-816-555-1212*, *http://www.ietf.org/dcx/getNumber* etc.

eXtensible Markup Language (XML)

Die *eXtensible Markup Language (XML)* [12] wird als Untermenge der Standard *Generalized Markup Language (SGML)* [26] in Form einer *Document Type Definition (DTD)* definiert. XML ist eine stark verbreitete, textbasierende Beschreibungssprache. Als modularer Standard wird die plattformneutrale Strukturierung von Daten und Informationen lizenzfrei sehr gut unterstützt. Optional ermöglichen die *DTD* und das *XML-Schema* die Strukturdefinition der Dokumentinstanzen. DTD grenzt sich zu XML-Schema dadurch ab, dass die DTD selbst kein XML-Dokument ist und nicht die Ausdrucksmächtigkeit eines XML-Schema besitzt. Die XML-Familie ist ein

wachsender Satz von Modulen; Bspw. wird im Modul *eXtensible Stylesheet Language (XSL)* [124] die grafische Darstellung des XML-Dokumentes definiert sowie im Modul *XSL Transformation (XSLT)* [22] das Umstellen, Hinzufügen und Löschen von Elementen und Attributen ermöglicht. Basierend auf der XML-Familie vergrößert sich das Angebot an XML-Standards durch die Auswirkungen und Einflüsse von Internet, E-Commerce, Künstlicher Intelligenz, Multimedia, Objekt-Orientierung, Datenbanken etc. Alle *Web Services* Standards basieren auf der XML-Technologie. Im Folgenden wird eine XSL-Transformation dargestellt, die in Form einer Schleife alle `<S-Klasse>`-Elemente nach ihrem Verkaufs-Preis (VK) sortiert [72].

```
<xsl:for-each select="S-Klasse">
   <xsl:sort select="VK"/>
</xsl:for-each>
```

SOAP

Auf XML basierend ist SOAP 1.2 [60] ein Protokoll, das den Austausch struktuierter Daten zwischen Service-Provider, Service-Consumer und Vermittler plattformneutral ermöglicht. Bis zur Version 1.1 stand SOAP für *Simple Object Access Protocol*; die aktuelle Version 1.2 distanziert sich von dieser Bezeichnung. Generell ist SOAP unabhängig vom darunterliegenden Transportprotokoll. Dennoch ist HTTP das präferierte und mit Beispielen dokumentierte Transportprotokoll des W3C. SOAP gliedert sich in die drei Elemente *SOAP-Envelope (dt. Umschlag)*, *SOAP-Header (dt. Kopf)* und *SOAP-Body (dt. Hauptteil)*. Das Element `Envelope` dient als Container für `Header` und `Body`. Das Element `Header` ist optional und transportiert Informationen über die Sicherheit, wie bspw. Verschlüsselung sowie über die Service-Güte *(engl. Quality of Service (QoS))* wie bspw. Antwortzeiten. Das Element `Body` ist zwingend und transportiert die Nutzdaten in Form der beiden Kommunikationsarten: *SOAP-Dokument-Style* oder *SOAP-RPC-Style*. Im Folgenden wird exemplarisch die Operation `xmethodsBabelFish` des Service `BabelFish` mit den beiden Übergabeparametern `Translationsmode` (*Deutsch* → *English*) und `Quelle` *(Hallo Welt!)* mittels `SOAP-Request` aufgerufen.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas..."
                   xmlns:xsi="http://www.w3.org/..."
                   xmlns:xsd="http://www.w3.org/...">
   <SOAP-ENV:Body>
      <ns1:BabelFish xmlns:ns1="urn:xmethodsBabelFish"
         SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/...">
            <transl... xsi:type="xsd:string">de_en</...ationmode>
            <source... xsi:type="xsd:string">Hallo Welt!</...data>
      </ns1:BabelFish>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Die Rückgabe erfolgt mittels `SOAP-Response` und dem Rückgabeparameter `return` mit dem Wert *hello world!*.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas..."
                SOAP-ENV:encodingStyle="http://schemas..."
                   xmlns:xsi="http://www.w3.org/2001/..."
                   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/..."
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <namesp1:BabelFishResponse xmlns:
       namesp1="urn:xmethodsBabelFish">
         <return xsi:type="xsd:string">hello world!</return>
      </namesp1:BabelFishResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Wie bereits im ersten Absatz erwähnt, ermöglichen die beiden Arten *SOAP-Dokument-Style* und *SOAP-RPC-Style* die Kommunikation zwischen Service-Provider, Service-Consumer und Vermittler. Der *SOAP-Dokument-Style* erlaubt eine vollkommen freie Strukturierung des `Body`-Elements passend zu einem XML-Schema. Der *SOAP-RPC-Style* erfordert die folgende Struktur für die Methodennamen sowie für die Ein- (`I`) und Ausgabeparameter (`O`) [173].

```
   <env:Body>
      <m:&methodName xmlns:m="someURI">
         <m:&param1>...</m:&param1>
         <m:&param2>...</m:&param2>
      </m:&methodName>
  </env:Body>
```

Web Service Description Language (WSDL)

Die Web Service Description Language 1.1 [21] wird in zwei Bereiche, die Schnittstellenbeschreibung *(engl. abstract interface)* und die Implementierung *(engl. concrete implementation)*, unterteilt. Im ersten Bereich werden die Elemente `<definitions>`, `<types>`, `<messages>` und `<portTypes>` definiert; der zweite schließt die WSDL mit den Elementen `<binding>` und `<service>` ab. Das Element `<definitions>` deklariert weltweit eindeutige Bereiche *(engl. Namespaces)* in Form einer URI (siehe Kapitel 2.1.6 auf Seite 27) sowie die Adresse *(engl. TargetNamespace)* des Services. Das Element `<types>` definiert komplexe XML Datentypen *(engl. complex types)* (bspw. Adresse, Fax-Nummer und Währung). Einfache XML Datentypen *(engl. simple types)* (bspw. `xsd:integer`, `xsd:string` und `xsd:float`) werden durch das XML-Schema deklariert und sind somit direkt nutzbar. Das Element `<message>` baut auf die Datentypdefinition auf und deklariert die einzelnen Ein- und Ausgabenachrichten *(engl. messages)*, die zwischen dem Service-Provider und dem Service-Consumer ausgetauscht werden.

Das Element `<portType>` definiert Operatoren durch die Elemente `<operation>`, `<input>`, `<output>` und `<fault>`. Das Element `<binding>` definiert das Transportprotokoll zwischen Service-Provider und -Consumer und eine optionale Verschlüsse-

```
<definitions>
    <!-- Abstrakt -->
    <types> ... </types>
    <message>
        <part> ... </part>
    </message>

    <portType>
        <operation>
            <input> ... </input>
            <output> ... </output>
            <fault> ... </fault>
            <documentation> ... </documentation>
        </operation>
    </portType>

    <!-- Konkret -->
    <binding>
        <operation>
            <input> ... </input>
            <output> ... </output>
            <fault> ... </fault>
        </operation>
    </binding>

    <service>
        <port> ... </port>
    </service>
</definitions>
```

Abbildung 2.5: Web Service Discription Language 1.1

lung der Nachrichten auf der Ebene der Operatoren. Das Element `<service>` verbindet das zuvor definierte Element `<binding>` mit dem Service-Endpunkt [173].

Service-Endpunkt

Der Begriff `binding` beschreibt die Anforderungen an eine physikalische Verbindung für die Kommunikation bzw. den Transport der Nachrichten wie das Transportprotokoll (i.d.R. SOAP) und den Dokument-Style. Der Term `portType` beschreibt abstrakt, in Form symbolischer Namen, die Operationen und die Ein- bzw. Ausgabenachrichten eines Services. Der Term `port` beschreibt eine Assoziation zwischen `binding` sowie einem Uniform Resource Identifier (siehe Kapitel 2.1.6 auf Seite 27) und kann somit als Service-Endpunkt bezeichnet werden [167] [37]. Zur Verbesserung der Transparenz wurde in der Spezifikation WSDL 2.0 der Term `portType` in `interface` und der Term `port` in `endpoint` umbenannt [42]. Der Standard Web Services Enpoint Language (WSEL) [149] verfolgt das Ziel, alle notwendigen und hinreichenden Charakteristiken eines Endpunkts zu beschreiben: QoS, Operationstypen (lesen, schreiben etc.), Kostenstrukturen und Sicherheitsanforderungen.

Abbildung der Datentypen

Interoperabilität zwischen den unterschiedlichen Implementierungen wie bspw. Java, C#, COBOL (Common Business Oriented Language) etc. ist eine der aktuellen Herausforderungen. Das *Web Services Apache SOAP-Framework AXIS* [52] definiert Standard-Abbildungen *(engl. standard mappings)* der Datentypen in der Programmiersprache Java. Primitive Datentypen werden mit ihren Wrapper-Klassen (Byte, Double, Boolean etc.) überschrieben. Im Folgenden wird die Standard-Abbildung für die Programmiersprache Java dargestellt (vgl. Tabelle 2.2):

WSDL	Java
xsd:base64Binary	byte[]
xsd:boolean	boolean
xsd:byte	byte
xsd:dateTime	java.util.Calendar
xsd:decimal	java.math.BigDecimal
xsd:double	double
xsd:float	float
xsd:hexBinary	byte[]
xsd:int	int
xsd:integer	java.math.BigInteger
xsd:long	long
xsd:QName	javax.xml.namespace.QName
xsd:short	short
xsd:string	java.lang.String

Tabelle 2.2: Abbildung der Datentypen von WSDL auf Java

Universal Description, Discovery and Integration (UDDI)

Der Verzeichnisdienst *Universal Description, Discovery and Integration (UDDI)* [23] stellt Informationen über Services, Service-Endpunkte und ihre Service-Provider zur Verfügung. Die Schnittstelle der UDDI ist via SOAP-Messages erreichbar. Generell werden die Informationen in die drei Teilbereiche *White Pages*, *Yellow Pages* und *Green Pages* untergliedert. *White Pages* geben Auskunft über das Namensregister, Kontaktinformationen und eine Auflistung der Detail-Informationen der Provider. *Yellow Pages* sind das Branchenverzeichnis und unterstützten das spezifische Suchen in unterschiedlicher Taxonomien wie bspw. Service-Art. *Green Pages* informieren über das Geschäftsmodell und die Geschäftsprozesse des Providers sowie über technische Details zu den Web Services [107].

Service-Status

Der Lebenszyklus eines Services wird innerhalb der Operationsbeschreibungen im Feld `Status` dokumentiert und hat folgenden Verlauf (siehe Abbildung 2.6). Nach einer abgeschlossener Definition des Entwicklungsprozesses wird der Zustand *Kandidat* gegebenenfalls in mehrere Zustände aufgesplittet: *designed*, *implementiert* und *getestet*.

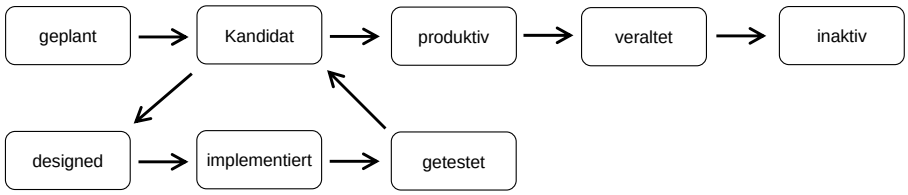

Abbildung 2.6: Service-Status

2.1.7 Enterprise Service Bus (ESB)

Der Begriff *Enterprise Service Bus (ESB)* wird von den Software-Herstellern teilweise als Namensbezeichnung für ein oder mehrere Produkte benutzt. Dieser Tatbestand führt, in Verbindung mit der nicht eindeutigen Verwendung in der Fachliteratur, erneut (analog zu Web Service) zu einer unscharfen Terminologie, die es zu beseitigen gilt. Woods definiert den ESB als ein auf eine standardisierte Menge von Services basierendes Rahmenwerk, das seine Wiederverwendung in einer Anwendung im Enterprise Computing findet [171]. Bieberstein definiert den ESB als intelligente, verteilte, transaktionale, nachrichten-basierte Schicht, die dazu dient, Applikationen miteinander zu verbinden und das Verarbeiten unterschiedlicher Daten, die in der Regel innerhalb einer Enterprise Computing Infrastruktur verteilt sind, ermöglicht [16]. Der Standard Java Business Integration (JBI) [80] von SUN konkretisiert den ESB durch die aktuelle Spezifikation. So wird bspw. durch den definierten Zugriff auf Web Services ohne die Festlegung des Service-Endpunkts, eine Interoperabilität verschiedener ESBs untereinander ermöglicht. Dieser Standard wird jedoch nicht von allen Herstellern unterstützt, dennoch sind bereits erste Implementationen u.a. in der Open Source Community verfügbar.

Diese Arbeit versteht den ESB als Konzept zur Kommunikation von Services über eine standardisierte Infrastruktur mit dem Ziel der Integration unterschiedlicher Plattformen. Internationale, historisch gewachsene Unternehmen nutzen COBOL-Programme auf Legacy-Systemen (bspw. OS/390 oder z/OS [69]) sowie J2EE-Anwendungen auf Applikations-Servern. Somit muss ein ESB die Kommunikation unterschiedlicher Plattformen unterstützten. Im Folgenden wird die Definition eines ESB bei der Daimler AG und das von der Definition abgleitete Konzept (siehe Abbildung 2.7 auf Seite 33) dargestellt.

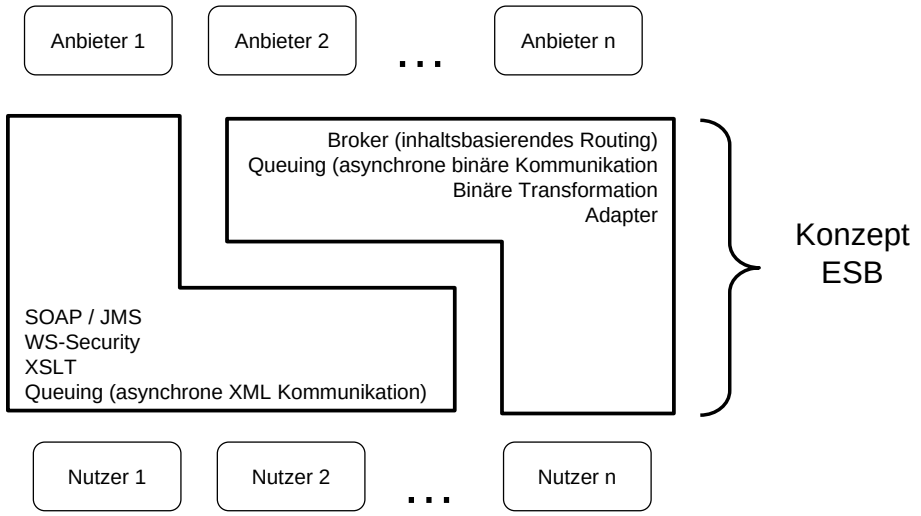

Abbildung 2.7: Elementares Konzept ESB

Definition 3 *Der Enterprise Service Bus (ESB) regelt die Kommunikation durch ein inhaltbasierendes Routing (engl. Content-Based Routing), Multi-Protokoll-Kommunikation sowie Validierung und Transformation von Nachrichten der Services über eine standardisierte Infrastruktur. Er unterstützt das Reagieren auf Ereignisse, die Sicherheit, das Monitoring, bietet Adapter zu Legacy-Systemen an und vereinfacht die Kommunikation mit dem Enterprise Service Repository [68].*

Die rechte Seite in Abbildung 2.7 illustriert die bereits etablierte Kommunikation über synchrone bzw. asynchrone (bspw. Message Queuing via MQSeries [166]) binäre Nachrichtenformate. Die Transformation von Nachrichten sowie Adapter zur Unterstützung der Kommunikation für unterschiedlichen Plattformen sind elementare Bestandteile eines ESBs. Die linke Seite in Abbildung 2.7 zeigt die synchrone und asynchrone standardisierte plattformneutrale Kommunikation über Web Services, die standardisierte Transformation der Nachrichten mit XSLT sowie die Unterstützung der Web Service Security Standards. Beide Seiten unterstützen inhaltsbasierendes Routing *(engl. Content-Based Routing)* durch die Verhaltensmuster: Beobachter *(engl. Observer)* (Synonym publiziere und abonniere *(engl. Publish-Subcribe)*), Anfrage-Antwort *(engl. Request-Response)* und das Reagieren auf Ereignisse *(engl. Event-Handling)*. Transportprotokolle, wie bspw. HTTP/SOAP, HTTPS/SOAP und JMS/SOAP werden komplett unterstützt, wobei hierbei die Hauptaufgabe des EBS's das Abbilden der Protokolle aufeinander (Service-Consumer und Service-Provider nutzen unterschiedliche Transportprotokolle) darstellt. Die Anwendung der standardisierten sicheren Kommunikation (Verschlüsselung, Autorisierung und Authentifizierung, Vertraulichkeitsstufen, Datenintegrität etc.) sind teilweise abhängig vom Reifegrad des Standards und von der proprietären Implementierung. Die rechte und die linke Seite des Konzeptes werden über einen Adapter miteinander verbunden, um eine geregelte Kommunikation Legacy-zu-Legacy, Web-Service-zu-Web-Service und Legacy-zu-Web-Service realisieren zu können [67]. Somit stellt

das Konzept ESB eine Möglichkeit dar, Service-Consumer und Service-Provider standardisiert in einer verteilten Umgebung technisch lose gekoppelt miteinander zu verbinden. Geschäftsobjekte werden technisch als Service Data Objects (SDO) [58] bzw. Service Message Objects (SMO) [30] transportiert. SDO und SMO basieren auf der selben Basisstruktur, definiert durch ein XML-Schema: *Hauptteil (engl. Body)*, *Kopf (engl. Header)* und *Kontext*. Der *Hauptteil* der Nachricht beinhaltet die Ein- bzw. Ausgabewerte (bspw. `getQuote.request.customerID`) der Service-Operationen, der *Kopf* beinhaltet Informationen über das Transportprotokoll (bspw. `messageUUID`, `version`, `messageType`) und der *Kontext* beinhaltet spezifische Informationen zum Ablauf *(engl. Flow)* (bspw. `correlation`, `transient`) oder Fehler-Informationen (bspw. `failInfo`). SMO erweitert SDO um bspw. COBOL und weitere Datenstrukturen. Diese standardisierten Austauschformate sind mit XPath, XSLT und Java bzw. C++ manipulierbar. Die erweiterten Anforderungen an einen ESB sind die Erstellung und Auswertung von Metriken basierend auf Key Performance Indikatoren (KPI), die Ermittlung von Quality of Service (QoS) und die Mediation in der Ablaufumgebung. Die Mediation beinhaltet eine Logik, abgebildet durch den Ablauf *(engl. Mediation Flow)*. Steuerelemente bestehen aus den Komponenten Logging, Lookup (Datenbank bzw. ESR), XSLT, Stopp, Fehler und Service-Endpunkt [30].

2.1.8 Enterprise Service Repository (ESR)

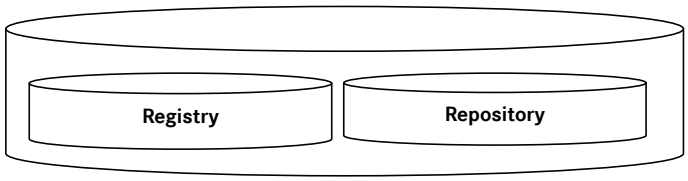

Abbildung 2.8: Konzept Enterprise Service Repository (ESR)

Die Abbildung 2.8 illustriert das Konzept *Enterprise Service Repository (ESR)* auf der höchsten Abstraktionsebene durch Aufzeigen der beiden Komponenten *Registry* und *Repository* im Infrastruktur-Layer. Auf Basis des unter Kapitel 2.1.6 vorgestellten Standards UDDI wurden bereits proprietäre Produkte der Software-Hersteller, wie bspw. IBM, SAP, Software AG (SAG) und Bea auf dem Markt positioniert. Der Standard UDDI legt Richtlinien für die Registrierung und die Aktualisierung der Metadaten sowie die maschinelle Suche und Zuordnung der Verantwortlichkeiten der Web Services fest (*White Pages, Yellow Pages, Green Pages*). Somit unterstützt der Standard UDDI nur eine Teilmenge der funktionalen und nicht-funktionalen Anforderungen (NFA) an ein ESR.

ebXML [111] ist eine Spezifikation zur Beschreibung von Geschäftsprozessen durch die Definition des Datenformats, der Nachrichten und der Austausch-Mechanismen [61]. Diese Spezifikation ist unabhängig von einer technischen Realisierung,

in der Web Services als Sonderfall eines Geschäftsprozesses betrachtet werden. Die *ebXML Registry* spezifiziert die Beschreibung und Speicherung von Information zur Integration von Geschäftsprozessen [11]. Implementierungen eines ESR unterstützen den Standard UDDI und erweitern die Anforderungen durch proprietäre Konzepte. Bspw. integriert das Unternehmen Bea den ebXML-basierten Ansatz und das Unternehmen IBM den OWL-basierten Ansatz in Ihre Produkt-Suite.

Qualitätsmerkmale der Software-Produkte wie Zuverlässigkeit, Wartbarkeit, Verfügbarkeit, Performanz, Portierbarkeit und Sicherheit lassen sich auf nicht-funktionale Anforderungen der Services abbilden [125]. Im Projekt *SOA for Automotive (SOAFA)* [113] wurden weitere nicht-funktionale Anforderungen im Speziellen für ESRs abgeleitet: Nutzung offener und frei-verfügbarer Standards für Datenmodell und Zugriffsschnittstelle, Nutzung von Standards mit einem größtmöglichen Grad an technischer Reife und hohem Maß an Verbreitung zur Sicherung der Investitionssicherheit und Interoperabilität sowie Bereitstellung einer für die Nutzergruppen adäquaten Schnittstelle zum Zugriff auf das ESR [8]. Funktionale Anforderungen lassen sich in die Klassen Informationsregistrierung, Sicherheits- und Monitoringaspekte, Informationsmanagement sowie Architekturbildung einteilen und beschreiben die zu erwartende Funktionalität des ESRs [115] [8]. Im Anhang C auf Seite 155 werden die funktionalen Anforderungen an ein ESR dargestellt.

2.1.9 Service-Portfolio-Management

Das Service-Portfolio-Management ist die organisatorische Instanz zur Planung, Koordination und Verwaltung der funktionalen und nicht-funktionalen Informationen im Enterprise Service Repository (siehe Kapitel 2.1.8 auf Seite 34). Somit ist das Service-Portfolio-Management die Instanz zur Überwachung der Realisierung neuer Services, zur Prüfung redundanter Services, zur Verwaltung der Service-Versionen und zur Prüfung der Dokumentationspflicht vorhandener Services. Sie stellt weiterhin die Eskalationsinstanz mit Entscheidungskompetenz bei Differenzen zwischen Service-Besitzer, Service-Nutzer und Service-Betreiber dar. Unter SOA Governance wird die Steuerung aller SOA-Aktiväten verstanden. Die Herausforderungen einer SOA Governance werden in der Praxis durch Rollen wahrgenommen. Die SOA Governance & Management Methode (SGMM) der IBM unterstützt methodisch die Governance-Struktur eines Unternehmens. Im Anhang E auf Seite 159 werden Informationen beschrieben, die als Entscheidungshilfe dienen und vom Service-Portfolio-Management eingefordert werden.

2.2 Entwurfsmuster Mediator

Entwurfsmuster *(engl. Design Pattern)* beschreiben in der Software-Architektur beständig wiederkehrende Entwurfsprobleme und erläutern eine Problemlösung. Ziel ist es, die Entwurfsmuster als wiederverwendbare Vorlagen (Schablonen) in der Software-Architektur zu etablieren, um Synergien in der Software-Entwicklung zu erzielen. In der Fachliteratur wird das Entwurfsmuster untergliedert in *Erzeugungsmuster* (Abstrakte Fabrik, Erbauer, Prototyp etc.), *Strukturierungsmuster* (Adapter, Brücke, Proxy etc.) und *Verhaltensmuster* (Beobachter, Iterator, Mediator etc.). Erich Gamma beschreibt in seiner Dissertation [55] einen Teil der heute bekannten Entwurfsmuster und stellt auf der OOPSLA 91 einen ersten Katalog vor.

Generell beinhaltet ein Entwurfsmuster die folgenden vier grundlegenden Elemente: *Entwurfsmustername, Problemabschnitt, Lösungsabschnitt* und *Konsequenzenabschnitt*. Der *Entwurfsmustername* besteht aus ein oder zwei Worten, die das Entwurfsproblem, seine Lösung und Auswirkungen eindeutig benennt. Die Findung und die Akzeptanz des somit neu entstandenen Vokabulars stellt die erste Herausforderung dar. Aktueller Kontext und Entwurfsproblem werden im *Problemabschnitt* adressiert. Hier werden Bedingungen definiert, die erfüllt sein müssen, um das Entwurfsmuster erfolgreich anwenden zu können. Das letzte Grundelement *Konsequenzenabschnitt* beschreibt die Auswirkungen (wie bspw. Ausführungszeit und Speicherplatzverbrauch) der Anwendung des Entwurfsmuster in Form einer Auflistung der Vor- und Nachteile.

Um die Anwendung zu vereinfachen, werden alle Entwurfsmuster konsequent durch ein einheitliches Format beschrieben. *Mustername und Klassifzierung* vermitteln knapp, aber präzise den wesentlichen Inhalt des Entwurfsmusters. Die Klassifizierung besteht aus der vertikalen Dimension Aufgaben und der horizontalen Dimension Gültigkeitsbereich. Die Dimension Aufgaben unterteilt sich in Erzeugungs-, Strukturierungs- und Verhaltensmuster. Die Dimension Gültigkeitsbereich unterteilt sich in Klasse und Objekt. Der Abschnitt *Zweck* stellt kurz das Grundprinzip, die Aufgaben und den Zweck sowie spezifische Fragestellungen vor. Unter *auch bekannt als* werden bereits bekannte Namen des Entwurfsmusters aufgelistet. Der Abschnitt *Motivation* besteht aus dem Szenario, dem Entwurfsproblem und der Klassen- und Objektstruktur. Die *Anwendbarkeit* beschreibt Situationen zur Anwendung des Entwurfsmusters. Der Abschnitt *Struktur* stellt die grafische Repräsentation des Entwurfsmusters in Form eines Klassen-, Objekt-, oder Interaktionsdiagramms dar. Jedes Entwurfsmuster wird mindestens in Form eines Klassendiagramms abgebildet. Die Klassen- und Objektdiagramme basieren auf der Object-Modeling-Technique (OMT) [128]. Die weiteren Diagrammtypen finden ihre Anwendung bei Bedarf [78]. *Teilnehmer* beschreibt alle am Entwurfsmuster beteiligten Klassen, Objekte und ihre Zuständigkeiten. Der Abschnitt *Interaktionen* beschreibt die Zusammenarbeit der einzelnen Teilnehmer mit dem Ziel der gemeinsamen Aufgabenerfüllung. Im Abschnitt *Konsequenzen* werden Ziele sowie die Vor- und Nachteile des Entwurfsmusters diskutiert. *Implementierung* stellt spezifische Techniken,

mögliche Fallen sowie praxisnahe Tipps vor und zeigt ggf. programmiersprachenspezifische Aspekte auf. Im Abschnitt *Beispielcode* werden Codefragmente veranschaulicht diskutiert. *Bekannte Verwendungen* illustrieren bereits implementierte Entwurfsmuster in echten Systemen unterschiedlicher Anwendungsbereiche. Der letzte Abschnitt *Verwandte Muster* grenzt das bezeichnete Entwurfsmuster ab und diskutiert Kombinationsmöglichkeiten mit anderen Entwurfsmustern. Im folgenden wird das Verhaltensmuster Mediator *(dt. Vermittler)* im soeben dargestellten Format beschrieben [56].

Mediator. Der Mediator ist ein objektbasiertes Verhaltensmuster.

Zweck. Der Mediator definiert ein Objekt, um das Zusammenspiel zwischen einer Menge von Objekten zu kapseln. Durch eine Unterbindung des direkten und expliziten Bezugs auf Objekte wird die lose Kopplung der Objekte gefördert.

Motivation. Die objektorientierte Software-Entwicklung fördert ein verteiltes Verhalten zwischen Objekten. Somit können Objektstrukturen und ihre Beziehungen eine Komplexität von $\frac{1}{2}(n^2 - n)$ erreichen. Generell unterstützt die Unterteilung eines Kommunikation-Systems in Klassen und Objekte die Wiederverwendung. Durch komplexe Beziehungen der Objekte untereinander tendiert das System zu einem Monolithen. Der Mediator kapselt das Verhalten der Objekte und vermittelt als eigenständiges Objekt zwischen den beteiligten Objekten.

Anwendbarkeit. Das Mediator Verhaltensmuster ist anzuwenden, wenn

- eine Menge von Objekten auf komplexer Weise miteinander kommuniziert. Somit besteht die Gefahr von unstrukturierten und schwer verständlichen Abhängigkeiten.
- die Wiederverwendung eines Objektes schwierig ist, da dieses Objekt mit vielen anderen Objekten kommuniziert oder diese Objekte referenziert.

Struktur. In Abbildung 2.9 wird die generische Struktur des Mediator Verhaltensmusters grafisch dargestellt. Ziel ist die lose Kopplung der Klasseninstanzen bei verteiltem Verhalten. Dabei kennt die Klasse *Kollege*, symbolisiert durch die gerichtete Assoziation, die abstrakte Klasse *Vermittler*. Die Vererbungsbeziehung, zwischen der Oberklasse *Kollege* und den Unterlassen *KonkreterKollegeA* sowie *KonkreterKollegeB*, als auch zwischen der abstrakten Oberklasse *Vermittler* und der Unterklassse *KonkreterVermittler*, wird durch die Generalisierung symbolisiert. Die gerichtete Assoziation macht die beiden Klassen *KonkreterKollegeA* und *KonkreterKollegeB* bei der Klasse *KonkreterVermittler* bekannt.

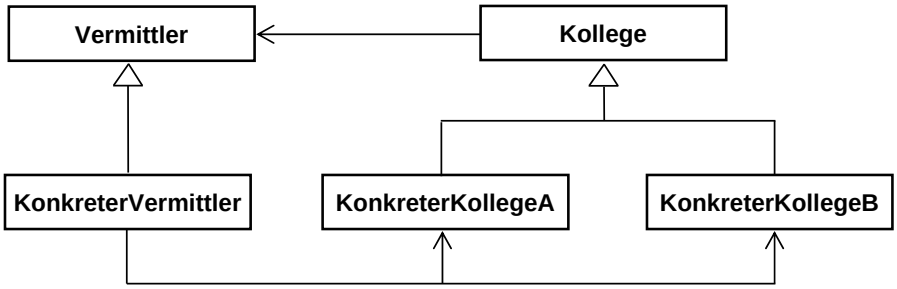

Abbildung 2.9: Generische Klassenstruktur Mediator [56]

Teilnehmer. Der *Vermittler* definiert als abstrakte Klasse die Interaktion mit den Kollegen-Objekten. Der *KonkreteVermittler* kennt, koordiniert und verwaltet die Kollegen-Objekte durch Implementierung des Gesamtverhaltens. Die *Kollegen-Klassen* kommunizieren über den Vermittler und nicht mit sich selbst.

Interaktionen. Der Vermittler koordiniert die Kommunikation zwischen den Anfragen und den Kollegenobjekten, indem er beim Senden und Empfangen der Anfragen an die Objekte vermittelt.

Konsequenzen. Durch die zentrale Kapselung des Verhaltens in der Klasse Vermittler sind die Kollegen-Klassen lose gekoppelt und können einfach wiederverwendet werden. Zuvor bestandene n-zu-m Beziehungen wurden durch eine 1-zu-n Beziehung ersetzt, womit das Verwalten und Erweitern der Klassen vereinfacht wird. Durch Abbildung der Komplexität in der Klasse Vermittler kann diese selbst du einem Monolithen werden.

Implementierung. Bei der Implementierung wurde die abstrakte Klasse Vermittler nicht realisiert. Die Notwendigkeit, eine abstrakte Klasse Vermittler einzuführen, ist nur durch eine gegebene Kommunikation mit weiteren abstrakten Vermittler-Klassen sinnvoll. Ereignisse, die durch die Kommunikation der Objekte ausgelöst werden, müssen durch den Vermittler beobachtet werden. Das Beobachter-Verhaltensmuster [56] ist ein möglicher Ansatz bei der Implementierung.

Bekannte Verwendungen und verwandte Muster. Das Strukturierungsmuster *Fassade (engl. Facade)* dient zur Realisierung einer vereinfachten abstrakten Schnittstelle und wird bspw. zur Abstraktion von Datenbankzugriffen eingesetzt. Das Verhaltensmuster *Beobachter (engl. Observer)* dient zur Überwachung des Zustands eines Objektes, der *Beobachter* benachrichtigt und aktualisiert automatisch alle anhängigen Objekte.

2.3 Semantische Web Services

2.3.1 Ontologien

Der Terminus technicus Ontologie verweist etymologisch auf die altgriechische Philosophie: Aristoteles definierte zum ersten Mal rationale Komponenten des Verstandes in Form einer exakten Menge von Gesetzen. Von diesem Verständnis ausgehend erlangte diese Wissenschaft von *dem Seienden* ihre philosophiegeschichtliche so eminente Rolle. Somit soll die Ontologie Grundstrukturen *des Seins* ermitteln: Was ist der Mensch? Gibt es Gott? Gibt es ein Anfang der Welt? Der Terminus Ontologie wird in der Wissensrepräsentation *(engl. Knowledge Representation (KR))* zum Speichern deklarativen Wissens in einer Wissensbasis (WB) genutzt.

Die Künstliche Intelligenz (KI) *(engl. Artificial Intelligence (AI))* nutzt die Wissensrepräsentation und die Inferenz, um Agenten zu steuern. Die Philosophen unterscheiden zwischen der *schwachen* und der *starken KI*. Die *schwache KI* untersucht die Hypothese *„Können Maschinen so agieren, als wären sie intelligent?“*, die *starke KI* versucht (im Gegensatz zum simulierten Denken) mentale Denkprozesse abzubilden [94]. Die KI selbst definiert den Terminus technicus Ontologie unterschiedlich, so definiert bspw. Gruber *„An ontology is a formal specification of a conceptualization“* [59]. Eine Konzeptualisierung ist eine abstrakte vereinfachte Sicht auf eine Domäne, die die Konzepte, Objekte und Beziehungen untereinander (innerhalb dieser Domäne) beschreibt. Im Jahre 1997 wurde die Definition von Gruber um den Terminus *shared* ergänzt und wie folgt definiert: *„An ontology is a formal specification of a shared conceptualization“* [19]. Im Jahre 1998 wird die Definition um den Terminus *explizit* ergänzt und wie folgt definiert: *„An ontology is a formal, explicit specification of a shared conceptualization“* [158].

In der Informatik wird unter dem Begriff der Ontologie die formale Repräsentation eines Systems von Begriffen mit zwischen diesen bestehenden Relationen verstanden [65]. Domänenspezifisches Wissen kann durch eine oder mehrere Ontologien beschrieben werden. Somit beschreibt eine Ontologie auf eine formale Art und Weise Konzepte und Relationen dieser Konzepte innerhalb einer Domäne eindeutig [154]. IT-Systeme nutzen implizite und explizite Konzeptualisierungen einer Domäne ihres Arbeitsgebiets [161].

Unterschiedliche Formalismen, wie bspw. Rahmen *(engl. Frames)*, Semantische Netze und Beschreibungslogik *(engl. Description Logic (DL))* bilden die theoretischen Grundlagen für die Repräsentation von Ontologien. Minsky beschreibt Rahmen als Objekte ohne Methoden, die Schubfächer *(engl. slots)* besitzen. Übergeordnete Rahmen können dabei die Werte der Schubfächer untergeordneter Rahmen vererben [104]. Dagegen postuliert Quillian Semantische Netze als ein formales Modell in Form von Begriffen und ihren Beziehungen und definiert somit Konzepte in Form von Taxonomien, Thesauri und Wortnetzen [123] [64]. Das Semantische Web *(engl. Semantic Web)* [14] erweitert das World Wide Web (WWW) um maschinenlesbare Informationen, durch welche die Bedeutung und der Inhalt formal festgelegt werden. Technisch kann die Erweiterung durch das, im Vergleich zu Se-

mantischen Netzen, ausdrucksstärkere Resource Description Framework (RDF) [87] und RDF-Schema realisiert werden [9]. Im Folgenden werden die Hauptmerkmale zur Wissensmodellierung in der Beschreibungslogik aufgezeigt. Die Beschreibungslogik nutzt eine abstrakte Sprache, in der Konzepte *(engl. concepts)* und Objekte *(engl. individuals)* sowie Rollen/Relationen *(engl. roles)* zwischen Objekten modelliert werden. Durch die Junktoren $\sqcap$, $\sqcup$ und $\neg$ werden aus Konzepten neue Konzepte gebildet. Bspw. bildet die Konjunktion der beiden Konzepte C und D (modelliert als $C \sqcap D$) ein (neues) wohldefiniertes Konzept, das die Objekte von Konzept C und Konzept D umfasst. Im Gegensatz zur Logik erster Stufe *(engl. First-Order-Logik (FOL))* [94] ist die Beschreibungslogik komplett frei von Variablen. Bspw. wird die Relation $C \sqcap D$, bei äquivalenter Semantik, syntaktisch in der Logik erster Stufe in Form $C(x) \wedge D(x)$ abgebildet.

Durch die beiden Quantoren Allquantor ($\forall$) und Existenzquantor ($\exists$) können Konzepte in der Form $\forall R.C$ bzw. $\exists R.C$ abgebildet werden. Relationen dienen dazu, neue Konzepte zu bilden. Objekte, die sich auf das zweite Argument einer Relation beziehen (C), werden als Funktionsfüller *(engl. role filler)* bezeichnet. Die Relation $\forall R.C$ beschreibt eine Menge von Objekten, deren Funktionsfüller dem Konzept C angehören und die Rolle R wahrnehmen. Im Gegensatz dazu beschreibt die Relation $\exists R.C$ eine Menge von Objekten, in der mindestens ein Objekt (Funktionsfüller) dem Konzept C angehören und die Rolle R wahrnehmen muss. Zur Verdeutlichung nehmen wir $FrontDoor$ als Konzept und $hasDoorway$ als Relation an. Somit beschreibt $\forall hasDoorway.FrontDoor$ eine Menge von Objekten (Konzept) mit ausschließlich Vordertüren; bei äquivalenter Semantik, syntaktisch in der Logik erster Stufe: $(\forall y)\ (hasDoorway(x, y) \Rightarrow FrontDoor(y))$. Im Gegensatz dazu beschreibt $\exists hasDoorway.FrontDoor$ eine Menge von Objekten mit mindestens einer Vordertür; bei äquivalenter Semantik, syntaktisch in der Logik erster Stufe: $(\exists y)\ (hasDoorway(x, y) \wedge FrontDoor(y))$. Die Logik erster Stufe wird auch als Prädikatenkalkül erster Stufe oder Prädikatenlogik bezeichnet. Neben den Junktoren $\wedge$, $\vee$ und $\neg$ finden weitere logische Verknüpfungen wie $\Rightarrow$ (Implikation), $\Leftarrow$ (Replikation) und $\Leftrightarrow$ (Äquivalenz) Anwendung in der Logik erster Stufe. Die Implikation wird als Konditional bzw. „Wenn/Dann-Aussage“, die Replikation als „notwendige Bedingung“ und die Äquivalenz als Bi-Implikation bzw. „Genau/Dann/Wenn-Aussage“ bezeichnet [129].

Generell wird ein Konzept als Menge von Objekten und eine Relation als Menge von Objektpaaren in einer zu interpretierenden Domäne verstanden. Die Interpretation (ein Modell) einer Menge von Ausdrücken der Beschreibungslogik besteht aus einer Menge von Objekten, zwischen denen Relationen festgelegt werden. Die Unendlichkeit *(engl. non-finiteness)* und die Offene-Welt-Annahme *(engl. open world assumption)* sind die beiden Alleinstellungsmerkmale der Beschreibungslogik. Die Offene-Welt-Annahme geht davon aus, dass das Wissen über die Konzepte, Relationen und Objekte nicht vollständig vorhanden ist. Offene-Welt-Annahmen werden durch die formale Definition notwendiger und hinreichender Bedingungen zu Geschlossene-Welt-Annahmen. Die Wissensbasis besteht aus einer Menge TBox

(engl. Terminological Box) von Regeln und einer Menge ABox *(engl. Assertional Box)* von Grundatomen, die über den einstelligen Prädikaten (Konzepten) und zweistelligen Relationen (Rollen) gebildet werden [10]. Beschreibungslogiken bestehen aus einer Familie abstrakter Sprachen, so ist bspw. $\mathcal{ALC}$ *(engl. attribute language with complement)* [136] eine Sprache dieser Familie. $\mathcal{ALC}$ wurde bereits 1991 von Schmidt-Schauß und Smolka als minimaler Bestandteil einer abstrakten Sprache auf der Basis atomarer Konzepte C, Rollen R, dem allgemeingültigen Konzept $\top$ *(engl. universal concept)*, dem Grundkonzept $\bot$ *(engl. bottom concept)*, der Junktoren $\sqcap$, $\sqcup$, der Negation von Konzepten $\neg C$ durch die Relationen universelle Quantifizierung *(engl. universial quantification)* $\forall R.C$ und existenziellen Quantifizierung *(engl. existential quantification)* $\exists R.C$ vorgestellt. $\mathcal{S}$ beinhaltet die Ausdrucksmächtigkeit von $\mathcal{ALC}$ in gleichzeitiger Erweiterung durch transitive Relationen. $\mathcal{H}$ steht für die Abbildung von Hierarchien in Relationen, $\mathcal{O}$ für die Aufzählung von Werten *(engl. enumeration)*, $\mathcal{I}$ für inverse Relationen *(engl. inverse properties)*, $\mathcal{N}$ für Kardinalität *(engl. cardinality restrictions)* und $(\mathcal{D})$ für Datentypen in einer konkreten Domäne [10]. Zusammenfassend ist $\mathcal{SHOIN}(\mathcal{D})$ eine ausreichend mächtige abstrakte Sprache, um Konzepte wie bspw. einen *SLK* definieren zu können. Das Akronym SLK steht für „sportlich, leicht, kompakt“ und ist ein Personenkraftwagen vom Typ Roadster.

$$
\begin{aligned}
SLK \sqsubseteq\ & =1\ hasDoorway.leftFrontDoor \sqcap \\
& =1\ hasDoorway.rightFrontDoor \sqcap \\
& =1\ hasSeat.leftSeat \sqcap\ =1\ hasSeat.rightSeat \sqcap \\
& \forall hasDoorway\ (leftFrontDoor \sqcup rightFrontDoor) \sqcap \\
& \forall hasSeat\ (leftSeat \sqcup rightSeat) \sqcap \\
& \exists hasTop.topSoftTop\ \sqcap\ Car
\end{aligned}
\tag{2.1}
$$

Eine Ontologie beschreibt Konzepte und Relationen innerhalb einer Domäne eindeutig. Somit wird ein gemeinsamer Wort- bzw. Sprachschatz definiert, der das Teilen, Verteilen und Austauschen von Informationen in einer Domäne signifikant verbessert (bspw. im Vergleich zu Datenbanken) [110]. Die Dokumentation der IT-Systeme ist in heterogenen Datenformaten (Word, Excel, Powerpoint, Datenbanken etc.) im Unternehmen verteilt gespeichert. Durch die Externalisierung des Know-hows in eine Ontologie wird folgender Nutzen erwartet:

- Identifizierung äquivalenter Services durch semantische semiautomatische Unterstützung (primär)
- Allgemein gültiges Verständnis der Domäne Automotive sowie Unterstützung der Mensch-Maschine Kommunikation (sekundär)

2.3.2 Web Ontology Language (OWL)

Technologien zur Beschreibung von Ontologien sind unter anderem das RDF und RDF-Schema (RDF(S)) , DAML[1]-OIL[2] [28], F-Logic [83], die Web Service Modeling Language (WSML) [33] und die unter ISO/IEC 13250:2000 normierten XML Topic Maps (XTM) [153]. Die vom World Wide Web Consortium (W3C) für das semantische Web propagierte Web Ontology Language (OWL) [98], ist ein W3C Standard und der Nachfolger der DAML-OIL Initiative. OWL basiert auf den theoretischen Grundlagen der Beschreibungslogik (siehe Kapitel 2.3.1 auf Seite 39) und technisch auf einer Erweiterung von RDF(S). Die Tabelle 2.3 illustriert die terminologische Abbildung der Beschreibungslogik auf OWL.

Beschreibungslogik	OWL
Konzept	Klasse
Rolle/Relation	Eigenschaft
Objekt	Objekt

Tabelle 2.3: Terminologische Abbildung der Beschreibungslogik auf OWL

Eine Klasse wird verstanden als eine bestimmte Menge von Objekten in einer zu interpretierenden Domäne. OWL erlaubt Klasse-Unterklasse-Beziehungen *(engl. subclass)*, Eigenschaft-Untereigenschaft-Beziehungen *(engl. subproperty)*, Domänen- und Bereichbegrenzungen sowie Klasseninstanzen (Objekte). Disjunkte Klassen, boolesche Bindung von Klassen, Angaben zur Kardinalität sowie besondere Merkmale von Eigenschaften sind in RDF(S) nicht darstellbar. Somit besitzt RDF(S) nur eine beschränkte Ausdrucksfähigkeit, Wissen in Form von Ontologien darzustellen [154]. Einen Kompromiss zwischen einer starken Ausdrucksfähigkeit und einer effizienten Inferenz (aufbereitetes Wissen, das aufgrund automatisierter logischer Schlussfolgerungen gewonnen wurde) bildet OWL. Die drei Ausprägungen *OWL Lite*, *OWL DL* und *OWL Full* (mit zunehmender Ausdrucksfähigkeit) wurden durch das W3C standardisiert. Im Folgenden werden die Sprachelemente der *OWL DL (engl. Description Logic)* beschrieben.

Die Wurzel *(engl. root)* aller OWL-Dokumente ist `<rdf:RDF>` und wird mit einer Kollektion von Vermerken (`<owl:label>`, `<owl:priorVersion>`, `<owl:imports>`, `<owl:versionInfo>`, `<owl:backwardCompatibleWith>`) begonnen. Die Eigenschaft `<owl:import>` ist transitiv definiert, womit bereits im Dokumentenkopf *(engl. Header)* die Mächtigkeit in Form einer speziellen Eigenschaft *(engl. special property)* der Sprache OWL aufgezeigt wird. Alle Konzepte werden in OWL mit `<owl:Class>` definiert. Eigenschaften *(engl. properties)* können über zwei unterschiedliche Typen, den Konzepteigenschaften und den Datentypeigenschaften, definiert werden. Konzepteigenschaften setzen Objekte zu Objekte in Relation (bspw. `isRealizedBy`), Datentypeneigenschaften setzen Objekte zu Datentypen (bspw. `xsd:int`) in Relation (bspw. `adultAge`).

[1] DARPA (Defense Advanced Research Projects Agency) Agent Markup Language
[2] Ontology Interchange Language

Die beiden einschränkenden Eigenschaften `<owl:allValuesFrom>` (∀) und `<owl:someValuesFrom>` (∃) werden in der praktischen Anwendung oft falsch verstanden. Das Äquivalent in deutscher Sprache für `<owl:allValuesFrom>` ist „nur“ und für `<owl:someValuesFrom>` ist „einige“ bzw. „mindestens ein“. Durch Ergänzungen der Eigenschaften, wie bspw. `<owl:minCardinality>` ist ein genaues Spezifizieren einer Menge möglich. Im Folgenden werden vier spezielle Eigenschaften *(engl. special properties)* und ihre Bedeutung bzw. Mächtigkeit beschrieben. Die spezielle Eigenschaft `<owl:TransitiveProperty>` definiert eine Eigenschaft als transitiv, d.h. aus `a isBiggerThan b` und `b isBiggerThan c` folgt `a isBiggerThan c`. Die spezielle Eigenschaft `<owl: SymetricProperty>` definiert eine Eigenschaft als symmetrisch, d.h. aus `y = x` folgt `x = y` (bspw. `hasSameGradeAs`, `isSiblingOf`). Eine Eigenschaft, die als spezielle Eigenschaft `<owl:FunctionalProperty>` für ein Objekt definiert wurde, kann höchstens mit einem Objekt (über diese Eigenschaft) in Beziehung gesetzt werden. Eine funktional definierte Eigenschaft wird auch als Einzelwert-Eigenschaft *(engl. single valued property)* bezeichnet. a, b und c repräsentieren Objekte wie bspw. Mütter. Eine Definition c `hasBirthMother` a und c `hasBirthMother` b impliziert, dass es sich bei a und b um die gleichen Objekte handelt (Menschen können nach dem heutigen Stand der Wissenschaft nur eine biologische Mutter haben). Die vierte und letzte spezielle Eigenschaft `<owl:Inverse-FunctionalProperty>` stellt die Inverse zu `<owl:FunctionalProperty>` dar. Definiert man die Eigenschaft `isBirthMotherOf` als invers-funktionale Eigenschaft von `hasBirthMother`, muss `hasBirthMother` als funktional definiert werden. Somit impliziert die Definition a `isBirthMotherOf` c und b `isBirthMotherOf` c, dass es sich bei a und b um die gleichen Objekte handelt.

Die Junktoren ⊓ *Schnittmenge (engl. intersection)*, ⊔ *Vereinigung (engl. union)* und ¬ *Komplement (engl. complement)* erweitern die Mächtigkeit von OWL durch die Abbildung boolescher Relationen zwischen Klassen. Aufzählungen *(engl. enumerations)* durch `<owl:oneOf>` ermöglichen die Definition einer Klasse oder eines Objektes durch die Auflistung seiner Elemente. Analog zu RDF werden die Objekte einer Klasse über `<rdf:ID>` instanziiert. OWL unterstützt analog zu dem XML Schema die Definition von eigenen bzw. fachlichen Datentypen wie bspw. `adultAge` oder `faxNumber`. In dem Anhang H auf Seite 167 wird das Konzept *SLK* (modelliert in Kapitel 2.3.1 auf Seite 39) dargestellt.

Inferenzmaschinen, wie bspw. RACER (Renamed Abox and Concept Expression Reasoner) [85], FaCT++ (Fast Classification of Terminologies) [71], pellet [102] oder KAON2 (KArlsruhe ONtologie) [83] validieren und erkennen logische Schlussfolgerungen durch Inferenz. So wird technisch betrachtet die Ontologie an bspw. RACER als Eingabe übergeben und nach der Validierung bzw. nach der Inferenz eine abgeleitete/gefolgerte *(engl. inferred)* Ontologie (angereichert mit logischen Schlussfolgerungen) zurückgegeben. Editoren wie bspw. Protégé [114] oder SWOOP [103] ermöglichen die Erstellung von Ontologien durch grafische Unterstützung. Resümierend kann gesagt werden, dass OWL der aktuell vorgeschlagene Standard für die Darstellung von Ontologien ist. OWL macht es möglich, Wissen in einer maschinen-

lesbaren Form zu beschreiben und basiert komplett auf RDF(S). Die theoretischen Grundlagen bilden die Prädikatenlogik [94], die Beschreibungslogik (siehe Kapitel 2.3.1 auf Seite 39) und die Inferenz [94] [10] [154].

2.3.3 Web Service Modeling Ontology (WSMO)

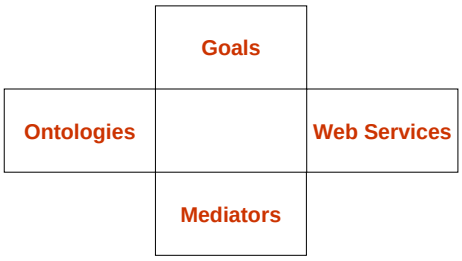

Abbildung 2.10: WSMO-Kernelemente

Die Web Service Modeling Ontology (WSMO) [126] ist ein durch das Digital Enterprise Research Institute (DERI) [76] in Innsbruck geleitetes wissenschaftliches Forschungsprojekt mit dem Ziel, alle Aspekte bzgl. Web Services (siehe Kapitel 2.1.6 auf Seite 27) zu beschreiben. Dabei liegt insbesondere das (komplett oder partiell) automatisierte suchen, finden, auswählen, ggf. orchestrieren, ggf. vermitteln (Mediation), ausführen und überwachen *(engl. monitoring)* von Web Services in der Ablaufumgebung im Fokus der Untersuchungen [48]. Die Grundelemente der WSMO sind Ontologien, Ziele, Mediation und Web Services (siehe Abbildung 2.10). Technologisch betrachtet besteht WSMO aus dem Web Service Modeling Framework (WSMF) [47], Ontologien und formalen Sprachen. Das Meta-Modell ist in Form eine Ontologie spezifiziert, in OWL (siehe Kapitel 2.3.2 auf Seite 42) modelliert und ist die konzeptuelle Grundlage für die Web Service Modeling Language (WSML) [33]. WSML definiert eine formale Sprache, die in ihrer Syntax zwischen Konzept und Logik unterscheidet. Die Syntax zur Beschreibung der Konzepte findet ihre Anwendung in der Beschreibung von Web Services, Ontologien und Ziele in Form von Konzepten. Im Gegensatz zur Beschreibung der Syntax findet die Beschreibung der Logik ihre Anwendung im Definieren von Bedingungen und Axiomen. WSML besteht aus einer formalen Beschreibungslogik unterschiedlicher Komplexitätsklassen, die als WSML-Core, WSML-DL, WSML-Flight, WSML-Rule und WSML-Full bezeichnet werden. WSML-Core ist die Klasse mit der schwächsten Ausdrucksmächtigkeit durch die Bildung der Schnittmenge zwischen Beschreibungslogik und Horn-Klauseln (Horn-Klauseln sind disjunkte negierte Literale mit genau einem positiven Literal). WSML-DL, WSML-Flight und WSML-Rule haben eine steigende Ausdrucksmächtigkeit in Richtung Beschreibungslogik. WSML-Full ist die Vereinigung zwischen den beiden Komplexitätsklassen und stellt somit die ausdrücksmächtigste Variante dar.

Ein Grundelement der WSMO ist die Mediation mit dem Ziel die Heterogenität aufzulösen bzw. das Ermöglichen der Interoperabilität auf den Ebenen Terminologie, Datenformat, Funktion und Prozess. WSMO definiert vier unterschiedliche Typen von Mediatoren: OO (Ontologie-zu-Ontologie), GG (Goal-zu-Goal) *(dt. Ziel-zu-Ziel)*, WG (WebService-zu-Goal) und WW (WebService-zu-WebService). OO-Mediatoren lösen die fehlende Äquivalenz der Konzepte unterschiedlicher Ontologien *(engl. mismatch between ontologies)* durch Fusion *(engl. merging)*, Anpassen *(engl. aligning)* und Abbilden *(engl. mapping)* auf. GG-Mediatoren verbinden Ziele miteinander und ermöglichen, basierend auf einem bereits existierenden Ziel, das Erstellen eines neuen Ziels. WG-Mediatoren verknüpfen *(engl. link)* einen Web Service mit einem Ziel, lösen terminologische Unterschiede auf und stellen ggfs. funktionale Unterschiede fest (falls vorhanden). WW-Mediatoren etablieren *(engl. establish)* Interoperabilität zwischen Web Services auf der Ebene Daten und Protokolle [157].

2.3.4 Semantic Markup for Web Services (OWL-S)

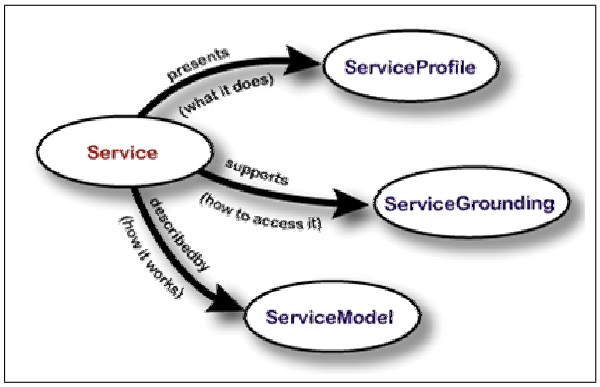

Abbildung 2.11: Bereiche in OWL-S

Analog zu OWL (siehe Kapitel 2.3.2 auf Seite 42) ist Semantic Markup for Web Services (OWL-S) [95] seit 2001 ein wissenschaftliches Forschungsprojekt der DAML-OIL [28] Initiative und seit 2004 ein W3C Standard. OWL-S, in der aktuellen Version 1.1, betrachtet verschiedene Aspekte wie das Profil *(engl. Profile)*, das Abbilden des OWL-S Service auf einen technischen Web Service *(engl. Grounding)* und das Modell *(engl. Model)* des Service (siehe Abbildung 2.11). Zur semantischen Beschreibung eines Web Service wurde eine Ontologie in OWL definiert. Ziel des wissenschaftlichen Forschungsprojekts ist die formale Beschreibung von Semantischen Web Services (SWS) *(engl. Semantic Web Services)*, um in der Ablaufumgebung automatisiert suchen, finden, auswählen, ggf. orchestrieren, ausführen und überwachen zu können. OWL-S basiert somit technologisch betrachtet auf den beiden W3C Standards Web Services sowie OWL und besteht aus den drei Bereichen *Service Profile*, *Service Model* und *Service Grounding*. Das *Service Profile* stellt eine Vorlage *(engl. Template)* zur Auszeichnung des Service dar und beschreibt die

Aufgaben *(engl. what a services does)* des Service in Form von IOPEs *(engl. Input, Output, Precondition, Effect)*. Das *Service Model* dient dazu, den Datenfluss *(engl. how a service works)* innerhalb des Service zu beschreiben und verfügt über drei verscheide Prozesstypen. Atomare Prozesse *(engl. atomic process)*, die direkt ausführbar sind und keine Unterprozesse beinhalten. Einfache Prozesse *(engl. simple process)*, die zum spezifizieren abstrakter Sichten dienen und nicht ausführbar sind. Kombinierbare Prozesse *(engl. composite process)*, die Kombinationen von atomaren, einfachen und kombinierbaren Prozessen ermöglichen. Das *Service Grounding* interagiert mit WSDL-Dokumenten *(engl. how to access it)* der Services und regelt die Ein- und Ausgabeparameter [130].

2.3.5 Web Service Semantics (WSDL-S)

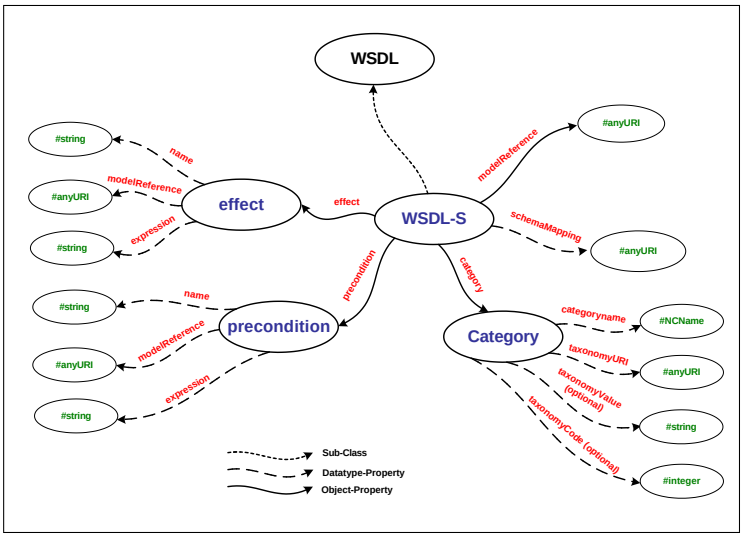

Abbildung 2.12: Web Service Semantik

Das dargestellte Meta-Modell (siehe Abbildung 2.12) wurde gemeinsam in der Dissertation Muhammad Ahtisham Aslam [3] aus den Beschreibungen im Web in Form von HTML-Seiten extrahiert und mit dem W3C abgestimmt. Die aktuellen Standards WSDL 1.1 (siehe Kapitel 2.1.6 auf Seite 27) und WSDL 2.0 [20] agieren auf syntaktischer Ebene und bieten somit keine Möglichkeit Web Services mit semantischer Information anzureichern. OWL (Kapitel 2.3.2) bietet eine ausreichend mächtige Sprache um Semantiken zu modellieren, aber keine Möglichkeit diese mit WSDL-Dokumenten zu verbinden. WSDL-S [4] wurde im Projekt METEOR-S [2] erarbeitet und ermöglicht durch die Definition weiterer Elemente die Anreicherung semantischer Informationen in WSDL-Dokumenten.

Im Folgenden werden die erweiternden Attribute und die Elemente kurz beschrieben. Das erweiternde Attribut `<wssem:modelReference>` dient als allgemeines Bindeglied zwischen dem WSDL-Dokument und den semantischen Informationen (bspw. modelliert in einer Ontologie). Anwendung findet das erweiterte Attribut

bei den Elementen `<xsd:simpleType>`, `<xsd:complexType>`, `<wsdl:operation>`, `<wssem:precondition>` und `<wssem:effect>`. Das zweite, erweiternde Attribut `<wssem:schemaMapping>` dient zur Auflösung struktureller Schemaunterschiede in `<xsd:complexType>`-Elementen. Die beiden neuen Elemente `<wssem:precondition>` bzw. `<wssem:effect>` sind Unterelemente von `<wsdl:operation>` und primär zur Servicefindung und nicht zum Aufruf eines Web Services vorgesehen. Das dritte, neue Element `<wssem:category>` wird zur Klassifizierung des Elements `<wsdl:interface>` genutzt, um optional den Web Service in einem Verzeichnis (abhängig von seiner semantischen Verwendung) kategorisieren zu können.

WSDL-S ermöglicht die Anreicherung der Eingabe (`I`), Ausgabe (`O`), Vor- (`P`) und Nachbedingung (`E`) sowie der Schnittstelle mit semantischen Informationen. Dabei wird bei der Ein- (`I`) und Ausgabe (`O`) zwischen Top-Level und Bottom-Level unterschieden. Top-Level bezeichnet die Anreicherung des Elementes `<xsd:complexType>` mit genau einer semantischen Information. Im Gegensatz dazu bezeichnet Bottom-Level die Anreicherung jedes einzelnen Elementes `<xsd:simpleType>` mit genau einer semantischen Information. Dabei werden strukturelle Schemaunterschiede durch das erweiternde Attribut `<wssem:schemaMapping>` in Form XSL-Transformation (siehe Kapitel 2.1.6 auf Seite 27) aufgelöst [54].

Bei Vor- (`P`) und Nachbedingung (`E`) wird, wie bei Ein- (`I`) und Ausgabe (`O`), die Anreicherung mit genau einer semantischen Information ermöglicht. Komplexe Konstrukte sollen extern (bspw. in einer Ontologie) verwaltet werden. Im Gegensatz dazu wird in dem aktuellen Folgeprojekt *Semantic Annotations for WSDL (SAWSDL)* [46] das mehrfache Vorkommen *(engl. multiplicity)* [51] von Anreicherungen semantischer Informationen in Form einer URI-Liste [75] ermöglicht. Die Anwendung wurde in Form von Qualitätsmerkmalen (bspw. RosettaNet [127]) durch das Ermitteln von Services demonstriert und ein Algorithmus zum Abgleich *(engl. machting)* der unterschiedlichen Qualitätsmerkmale vorgeschlagen [116].

Im Vergleich zu Forschungsinitiativen wie bspw. OWL-S (siehe Kapitel 2.3.4 auf Seite 45) und WSMO (siehe Kapitel 2.3.3 auf Seite 44) ist WSDL-S der minimalste Ansatz zur Anreicherung semantischer Informationen in WSDL-Dokumenten [48]. Neben den Nachteilen eines fehlenden definierten Algorithmus (explizites Modell) zur Beschreibung von Auswertung der semantischen Informationen sowie dem Mix aus syntaktischer/funktionaler und semantischer Beschreibungen sind folgende Punkte als positiv zu bewerten:

- Erweiterung eines verbreiteten WSDL-Standards
- Übergang zu mächtigeren Ansätzen wie OWL-S oder WSMO möglich
- Externalisierung des fachlichen Know-hows
- Freie Wahl der Sprache zur Wissensrepräsentation (bspw. UML, RDF, OWL)
- Einfachheit in der Anwendung
- Der Standard bietet die benötigte Stabilität (Einsatz in der Praxis)

2.3.6 Semantische Konzepte in einer SOA

Business Services Fabric. Die IBM stellte am 31. Oktober 2006 die *WebSphere Business Services Fabric* [1] vor, nachfolend als *Fabric* bezeichnet, bestehend aus *Foundation Pack*, *Tool Pack* und *Industry Content Packs*. Die Initiative verfolgt das Ziel, implementierte Prozesslogik und insbesondere Service-Auswahl-Logik in Ontologien (siehe Kapitel 2.3.1 auf Seite 39) auszulagern (Externalisierung), um dadurch die Flexibilität der Geschäftsprozesse zu erhöhen. Das *Foundation Pack* besteht aus einer Ablaufumgebung (Regel-Engine), die den WebSphere Prozess Server bzw. WebSphere Enterprise Service Bus erweitert. Das *Tool Pack* umfasst Anwendungen zur Entwicklung bzw. Administration und die *Industry Content Packs*, bestehend aus in OWL (siehe Kapitel 2.3.2 auf Seite 42) verfügbaren Ontologien (bspw. Domäne Finanzwesen), enthalten weitere Konzepte wie Service- und Geschäftsobjektdefinitionen. Die *Fabric* ermöglicht das Beschreiben von Services auf fachlicher Ebene mit Unterstützung der Ontologien. Bei Auswahl der Services in der Ablaufumgebung werden die Beschreibungen (fachliche Semantik) auf die erforderliche Service-Charakteristik mit Unterstützung einer erweiterten Regel-Engine abgebildet. Der Vorteil, im Vergleich zur Mediation im ESB in der Entwicklungszeit, ist die externalisierte deklarative Definition der Service-Auswahl-Logik und somit die Förderung der Wiederverwendung. Der Nachteil ist insbesondere die selektive (kein Näherungsverfahren), manuelle Bestimmung der Service-Signatur in der Entwicklungszeit sowie eine fehlende Validierung der Regeln.

SemanticMiner. In den beiden Veröffentlichungen [7] [105] der Ontoprise GmbH werden Möglichkeiten von semantischen Technologien im Einsatz innerhalb konkreter Anwendungen aufgezeigt, ohne dabei die WSDL-Dokumente mit semantischen Informationen anzureichern. Die Einsatzbereiche umfassen semantische Suchsysteme, die das Auffinden von Dokumenten und Datenbank-Informationen unterstützen, Lösungen zur semantischen Integration von verteilten Datenquellen bis hin zu semantischen Ratgeberlösungen.

Ubiquitous Semantic Web Services (USWS). Das Fraunhofer Institut für Arbeit und Organisation (IAO) stellte am 20. Dezember 2006 in Möhringen bei der Daimler AG den Abschlussbericht des Projektes *Ubiquitous Semantic Web Services (USWS)* [162] vor. Der Terminus technicus „Ubiquitous“ beutet allgegenwärtig und bezeichnet die Konjunktion zwischen Web Services, dem Semantischen Web und technologischen Voraussetzungen *(engl. enabling technologies)*. Das Projektziel war die Untersuchung von statischen und dynamischen Zugriffen (Web Service), Sicherheit *(engl. security)* beim Nutzen von Web Services und die Dokumentation des aktuellen Stands der Technik bezogen auf Semantische Web Services. Dabei wurde die aktuelle Werkzeugunterstützung von Inferenzmaschinen sowie grafische Werkzeuge zur Unterstützung der Modellierung von Ontologien dokumentiert. Die Untersuchung der sicheren Nutzung von Web Services prägte den Slogan *„Without Security, Web Services are Dead on Arrival“*.

SAP-Global-Datatypes (SAP-GDT). Die Eco-System Industrie Standards Gruppe (EIC), gegründet durch die SAP, hat das primäre Ziel kollaborative Geschäftsprozesse als Industriestandard zu etablieren [108]. Dieses Ziel soll durch Forschungsprojekte in Kombination mit praxisnahen Erfahrungen und Methoden *(engl. Best Practices)* erreicht werden und mittelfristig in die Produktentwicklung einfließen. SAP leistet durch die Teilnahme an verschiedenen Forschungsprojekten wie bspw. THESEUS [45], ATHENA [43] und R4eGov [44] einen signifikanten Beitrag zur Standardisierung und Interoperabilität von Geschäftsprozessen. Grundlage der Initiative EIC ist die enge Zusammenarbeit mit dem *United Nations Centre for Trade Facilitation and Electronic Business* (UN/CEFACT) [50]. Dort wurden bspw. die einheitlichen Namens- und Designregeln der Nachrichten sowie gemeinsame Datentypen verabschiedet, die maßgeblich in den aktuellen Anwendungen (SAP NetWeaver [35], SAP Business byDesign [36] etc.) genutzt werden.

Die SAP-Forschung stellte am 05. Februar 2007 bei der Daimler AG im Mercedes-Benz Werk Sindelfingen die SAP-Global-Datatypes (SAP-GDT) basierend auf dem Standard Core Component Technical Specification (CCTS) [109] vor. SAP-GDT wurde entwickelt, um die Integrationsaufwände (insbesondere bei der Abbildung von Schnittstellen) zu reduzieren und damit eine flexiblere Gestaltung von B2B-Kommunikation zu ermöglichen. SAP-GDT beschreibt Geschäftsobjekte in einer bereits definierten semantischen Struktur. Äquivalente Geschäftsobjekte werden kontextspezifisch durch die SAP-GDT (Anreicherung semantischer Informationen) strukturiert mit dem Ziel der Vereinheitlichung von Geschäftsobjekten, Schnittstellen und Operationen (Signatur). CCTS sind normierte bzw. abgestimmte Datentypen für Geschäftsobjekte und können wie folgt charakterisiert werden:

- wiederverwendbare Bausteine für Service-Schnittstellen oder Geschäftsobjekte
- formal: definiert durch ein Regelwerk, beschrieben im Standard UN/CEFACT
- technisch: definiert im ESR, beschrieben durch XML Schema
- bestätigt durch den SAP-Governance-Prozess

Im Folgenden wird die Anwendung der GDTs kurz skizziert. Durch die hierarchische Struktur bilden sich Blätter und Knoten (vgl. Anhang F auf den Seiten 163 und 164). Blätter sind sog. einfache *(engl. basic)* Datentypen und Konten sind sog. zusammengefasste *(engl. aggregate)* Datentypen. Geschäftsobjekte sowie Signaturen der Services werden durch die Kombination mehrerer Konten beschrieben. Dies führt zu einer übergreifenden Harmonisierung der Geschäftsobjekte und Services. Die Mitgliedschaft und aktive Teilnahme an der *SAP Enterprise Services Community Advisory Group Methodology* unterliegen einem Geheimhaltungsvertrag *(engl. Non-Disclosure Agreement (NDA))* und sind somit nicht zu publizieren. Tenor der Forschungsgruppe war bzw. ist die Entwicklung einer Vorgehensweise zur Identifizierung von Service-Kandidaten.

2.4 Stand der Wissenschaft

Im Folgenden wird die terminologische Redundanz der Begriffe *Mediation*, *Semantik* und *Dynamik* eindeutig abgebildet sowie im Passus Diskussion auf Seite 52 der aktuelle Stand der Wissenschaft in der Service-orientierten Architektur und den Semantischen Web Services diskutiert. Die vorliegende Arbeit kombiniert Konzepte der Service-orientierten Architektur (siehe Kapitel 2.1 auf Seite 15), das Entwurfsmuster Mediation (siehe Kapitel 2.9 auf Seite 38) und Konzepte der Semantischen Web Services (siehe Kapitel 2.3 auf Seite 39) zu der logischen Schicht EMEO-Layer (siehe Kapitel 6 auf Seite 85) mit dem Ziel der Identifizierung äquivalenter Services in der Entwicklungszeit. In Kapitel 2.1.2 auf Seite 18 wurde das Gestaltungsprinzip der losen Kopplung bereits detailliert diskutiert.

Mediation. Woods versteht Abstraktion der Implementierung auf seine Schnittstelle (Blackbox-Charakter) als lose Kopplung [171], Erl dagegen versteht die Abstraktion auf die Serviceoperation als lose Kopplung und ergänzt diese um die vertragliche Bindung zwischen Provider und Consumer [42]. Bieberstein führt die Unterscheidung zwischen loser Kopplung und Bindekraft sowie die Faktoren Zeit, Protokoll, Datenformat etc. ein [16]. Im Gegensatz dazu führt Krafzig die Unterscheidung zwischen fester und loser Kopplung sowie die Ebenen physikalische Bindung, Kommunikationsformat, Service-Bindung etc. ein [89]. Auf der Ebene der physikalischen Bindung unterscheidet Krafzig zwischen Punkt-zu-Punkt und Vermittlung *(engl. intermediary)*, die Josuttis später als *Mediation* bezeichnet und dazu weiterhin zwei Arten der *Mediation* einführt [79]. Die erste Art ist die Ermittlung des Service-Endpunkts vor dem Senden des Aufrufs, die zweite Art ist die Ermittlung des Service-Endpunkts nach dem Senden des Aufrufs. Ein Grundelement der Web Service Modeling Ontology (WSMO) [126] ist die *Mediation* mit dem Ziel, die Heterogenität aufzulösen bzw. die Interoperabilität zu ermöglichen (siehe Kapitel 2.3.3 auf Seite 44). Die WSMO definiert vier unterschiedliche *Typen von Mediatoren*: OO (Ontologie-zu-Ontologie), GG (Goal-zu-Goal) *(dt. Ziel-zu-Ziel)*, WG (WebService-zu-Goal) und WW (WebService-zu-WebService) mit dem Ziel der Konfliktlösung auf den Ebenen Terminologie, Datenformat, Funktion und Prozess [48]. In dieser Arbeit wird die *Mediation* als Vermittlung zwischen Service-Consumer und Service-Provider in Form einer Ermittlung des Service-Endpunkts nach dem Senden des Aufrufs verstanden.

Semantik. Auf der Ebene Kommunikationsformat wird *technisch* als feste und *semantisch* als lose Kopplung bezeichnet. Krafzig und Josuttis verstehen unter semantischer Kopplung die Definition eines Austauschformates ohne Abhängigkeit zur Signatur des Service. Im Gegensatz dazu definieren Tim Berners-Lee, James Hendler und Ora Lassila das Semantische Web als Erweiterung des aktuellen Webs, durch eindeutig definierte Informationen, die das Zusammenspiel zwischen Computer und Menschen verbessern: *„The Semantic Web is an extension of the current Web in*

which information is given well-defined meaning, better enabling computers and people to work in cooperation“ [14]. Folglich wird der Begriff Semantischer Web Service als eine Erweiterung der aktuellen Technologie Web Service (XML-basierende syntaktische Schnittstellenbeschreibung) verstanden, die, analog zum Semantischen Web, das Zusammenspiel zwischen Computer und Menschen durch eindeutig definierte Informationen verbessert. In dieser Arbeit wird *Semantik* im Kontext einer Service-orientierten Architektur, analog zu dem Semantischen Web und zu den Semantischen Web Services, verstanden als eindeutig definierte Informationen, die das Zusammenspiel zwischen Computer und Menschen verbessern.

Dynamik. Auf der Ebene Service-Bindung bezeichnen Krafzig, Erl, Josuttis und Bieberstein *statisch* als feste und *dynamisch* als lose Kopplung. Dabei wird die Zuweisung eines Aufrufs, zu einem konkreten Service-Endpunkt (siehe Kapitel 2.1.6 auf Seite 30), als *statisch* und das automatisierte Ermitteln des Service-Endpunkts, während der Ablaufumgebung, als *dynamisch* bezeichnet. Die Schnittstelle ist fix, womit Abweichungen der Schnittstelle nicht betrachtet werden [89] [42] [79] [16]. Die Initiativen Bottom-Up Approach [93], Template Based Composition [147], OWL-S Composition Approach [146], SHOP2 [148], welche technologisch auf dem wissenschaftlichen Forschungsprojekt OWL-S (siehe Kapitel 2.3.4 auf Seite 45) basieren, weiterhin die Initiativen METEOR-S [2] und SEMAPLAN [5], technologisch basierend auf dem wissenschaftlichen Forschungsprojekt WSDL-S (siehe Kapitel 7.1 auf Seite 100) sowie das wissenschaftliche Forschungsprojekt WSMO (siehe Kapitel 2.3.3 auf Seite 44), bezeichnen das automatisierte Ermitteln des Service-Endpunkts und der Schnittstelle während der Ablaufumgebung als *dynamisch*. Notwendiges menschliches Eingreifen wird von der Initiative METEOR-S als semiautomatisch bezeichnet. Bei der Orchestrierung während der Ablaufumgebung wird zwischen *dynamisch* und *automatisch* unterschieden, wobei *dynamisch* das Generieren eines technischen Prozessablaufs (bspw. in BPEL 1.1 (siehe Kapitel 2.1.5 auf Seite 25)) und *automatisch* die Steuerung eines oder mehrerer Agenten (KI-Ansatz) bezeichnet.

Service	finden & auswählen	Schnittstelle	binden
statisch	Entwicklungsumgeb.	fix	Entwicklungsumgeb.
quasi-statisch	Entwicklungsumgeb.	fix	Ablaufumgebung
semiautomatisch	Entwicklungsumgeb.	variabel	Ablaufumgebung
dynamisch	Ablaufumgebung	variabel	Ablaufumgebung

Tabelle 2.4: Katalogisierte Einbindung von Services in einen Prozess

In Tabelle 2.4 wird die Einbindung von Services in einen Prozess katalogisiert und tabellarisch dargestellt, wobei zwischen Entwicklungs- und Ablaufumgebung sowie fixe und variable Schnittstelle unterschieden wird. In dieser Arbeit wird die Orchestrierung in der Ablaufumgebung nicht betrachtet. *Dynamisch* wird, im Kontext einer Service-orientierten Architektur, analog zu Bottom-Up Approach, Template Based Composition, OWL-S Composition Approach, SHOP2, METEOR-S,

SEMAPLAN sowie OWL-S, WSMO und WSDL-S, verstanden als automatisierte Ermittlung (in der Ablaufumgebung) des Service-Endpunkts und der Schnittstelle. In Betrachtung der Ergebnisse der wissenschaftlichen Forschungsprojekte wird das Ermitteln des Service-Endpunkts mit einer fixen Schnittstelle in der Ablaufumgebung als quasi-statisch bezeichnet. Semiautomatisch wird, analog zu METEOR-S und WSDL-S, als notwendiges menschliches Eingreifen bezeichnet.

Diskussion

In prozeduralen Programmiersprachen wird ein Unterprogramm, das einen Programmblock durch die Signatur kennzeichnet und Variablen verarbeitet sowie zurückgibt, als Funktion bezeichnet [137]. In der Objekt-orientierten Programmierung wird eine bestimmte Menge von Operationen und Attributen, die durch den Klassennamen und die Vererbungshierarchie definiert werden, als Klasse bezeichnet. Eine Klasse kapselt die Implementierung der Algorithmen und Datenstrukturen über eine Schnittstelle und definiert das Verhalten der Klasseninstanzen (Objekte) [174]. Ein Objekt, ist die Instanz genau einer Klasse, repräsentiert den Objektzustand durch Attribute und realisiert die Zustandsänderungen über Operationen [159].

In der Literatur werden die Terme Funktion, Klasse und Objekt eindeutig definiert. Im Gegensatz dazu werden die Terme Komponente, Modul sowie Service nicht eindeutig definiert, sind somit unscharf und werden im folgenden diskutiert. D'Souza und Wills verstehen eine Komponente als zusammenhängende Einheit *(engl. package)* einer Softwareimplementierung mit expliziten sowie genau spezifizierten Schnittstellen für Service-Provider und selbst akzeptierten Services sowie die mögliche Kombinierbarkeit mit anderen Komponenten [38]. Szyperski charakterisiert den extern nicht ermittelbaren Status als Alleinstellungsmerkmal einer Komponente und definiert sie als unabhängig auslieferbare/installierbare Einheit mit expliziten Abhängigkeiten und unabhängiger Kombinierbarkeit mit Dritten: *„A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party“* [159]. Die Object Management Group (OMG) definiert durch die Unified Modeling Language (UML) eine Komponente als modulare austauschbare Einheit mit eindeutig definierten Schnittstellen: *„The component is a modular unit with well defined interfaces that is replaceable within its environment“* [112]. Siedersleben bezeichnet die Definitionen von D'Souza und Wills, Szyperski sowie der OMG als richtig, aber selbst in ihrer Gesamtheit als unvollständig. Zusätzlich ergänzt Siedersleben die Komponente als wesentliche Einheit (neben der Schnittstelle) für Planung, Entwurf und Implementierung. Reduziert im Gegensatz dazu die geforderte Eigenschaft der unabhängigen Auslieferbarkeit einer Komponente (Szyperski) und deklariert sie als optional [145]. In der Quality Software Architecture (Quasar) [120] definiert Siedersleben zusätzlich Softwarekategorien zum methodischen Auffinden von Komponenten wie bspw. A-Software und T-Software. Komponenten, die nicht weiter unterteilt sind, werden als Modul bezeichnet. Somit stellen Module die minimalste Form einer Komponente

dar [159] [145] [86]. Keen definiert einen Service als beliebige, eigenständige Funktion, Geschäftsfunktion oder eine Sammlung von Funktionen in Form eines Prozesses, die einem externen Consumer angeboten werden kann: *„A service can be defined as any discrete function that can be offered to an external consumer. This can be an individual business function, or a collection of functions that together form a process“* [84]. Aufgrund ihrer eindeutig definierten Schnittstellen und ihrer Unabhängigkeit in der Auslieferung werden in der Literatur Services teilweise als Komponenten bezeichnet. So definiert Krafzig einen Service als Software-Komponente, charakterisiert durch seine Funktionalität, die typischerweise ein Geschäftskonzept auf hoher Ebene kapselt: *„A service is a software component of distinctive functional meaning that typically encapsulates a high-level business concept“* [89]. Siedersleben nutzt die Definition einer Komponente von Szyperski und erweitert die Definition eines Service um die plattformneutrale Schnittstelle, um die Blackbox-Wiederverwendung und um die potentielle einmalige Auslieferung [145]. In dieser Arbeit wird der Term Service und der Term Web Service (siehe Kapitel 2.1.6 auf Seite 27) als Synonyme verstanden.

Pasik erkannte im Jahre 1994, durch die Bezeichnung von Hardware-Infrastruktur und Software-Architektur als Client/Server, die terminologische Redundanz des Terms Client/Server; Er klassifizierte die Kommunikation zwischen IT-Systemen in Form von Services als eine Middleware zur Gestaltung von Geschäftsanwendungen und prägte somit den Term SOA initial [79]. Schulte und Natis stellten in ihrer Publikation von 1996 die Service-orientierte Architektur in der heute bekannten Form (vgl. Kapitel 2.1 auf Seite 15) konzeptuell vor [138].

Mehrschichtige Architekturen *(engl. Multitier-Architectures)* bezeichnen Software-Architekturen, die ein IT-System in mehrere Ebenen *(engl. Tiers)* einteilen. Dreischichtige Architekturen *(engl. Three-Tier-Architectures)* sind Software-Architekturen, bestehend aus Präsentation *(engl. presentation)*, Geschäftslogik *(engl. business logic)* und Persistenz *(engl. persistence)* [131]. Ergänzend abstrahieren mehrschichtige Architekturen (mehr als drei Ebenen) durch erneute Aufteilung der Ebene Geschäftslogik in die beiden Bereiche Prozessablauf *(engl. workflow)* und Geschäftsfunktionen [132] [88] [155]. Muhammad Ahtisham Aslam stellt auf der OOSPLA eine vierschichtige Architektur [67] vor, die die Ebene Geschäftslogik, in die beiden Bereiche Orchestration und Services aufteilt. Die vierschichtige Architektur wird in der eingereichten Dissertation [3] konkretisiert.

Die Technologie Remote Procedure Call (RPC) [74] ermöglicht Funktionsaufrufe auf entfernten Rechnern über ein Netzwerk. XML-RPC (eXtensible Markup Language Remote Procedure Call) [172] definiert die Datentypen `int`, `double`, `boolean`, `string`, `dateTime` und `base64` zum Austausch zwischen Prozeduren und kann als Vorgänger von SOAP (siehe Kapitel 2.1.6 auf Seite 28) bezeichnet werden. Das Distributed Component Object Model (DCOM) [101], basierend auf dem Component Object Model (COM) [100], ermöglicht den proprietären Zugriff auf Objekte entfernter Rechner über ein Netzwerk. Im Gegensatz dazu ermöglicht die Tech-

nologie Remote Method Invocation (RMI) [141], basierend auf Java 2 Standard Edition (J2SE) [140], den plattformneutralen Zugriff auf Objekte entfernter Java Virtueller Maschinen (JVM) über ein Netzwerk. Die Common Object Request Broker Architecture (CORBA) [117], die einen Object Request Broker (ORB) als zentrale Komponente zum Datenaustausch vorsieht, ermöglicht den programmiersprachenneutralen Zugriff (spezifiziert in der IDL (Interface Definition Language)) auf Objekte entfernter Rechner über ein Netzwerk [40]. Die Technologie Web Services (siehe Kapitel 2.1.6 auf Seite 27) definiert, im Gegensatz zu DCOM, RMI und CORBA, den XML-basierenden Austausch der Nachrichten *(engl. Messages)* und umgeht somit die Spezifikation der Objektpersistenz sowie der Transaktionsmechanismen [173].

Die Grundlage des Verhaltensmusters Mediation wird in Kapitel 2.9 auf Seite 38 ausführlich beschrieben. So erweitert Bieberstein das Verhaltensmuster Mediator um XML-nach-XML Transformation, Nachrichten-Validierung sowie inhaltsbezogenes Routing. Damit ist der Mediator eine vermittelnde Komponente die, basierend auf Web Services, zwischen Service-Provider und Service-Consumer agiert [16]. Die Implementierung des erweiterten Mediators vermittelt durch die Multi-Protokoll-Fähigkeit (bspw. TCP/IP, JCA und JMS) nicht mehr ausschließlich zwischen Objekten einer Programmiersprache, sondern agiert plattformübergreifend. Wiederhold stellte bereits 1992 einen Leitfaden, bestehend aus sechs Hauptschritten, zur Mediation fehlender Zuordnungen zwischen Datenquellen vor [170]. Dieser Ansatz basiert auf Agententechnologie, betrachtet aber das Verhalten während der Ablaufumgebung in einem internen Netzwerk (Intranet) nicht. Ein Grundelement der Web Service Modeling Ontology (WSMO) [126] ist die *Mediation* mit dem Ziel der Auflösung von Heterogenität bzw. dem Ermöglichen von Interoperabilität (siehe Kapitel 2.3.3 auf Seite 44). Die WSMO definiert vier unterschiedliche *Typen von Mediatoren*: OO (Ontologie-zu-Ontologie), GG (Goal-zu-Goal) *(dt. Ziel-zu-Ziel)*, WG (WebService-zu-Goal) und WW (WebService-zu-WebService) mit dem Ziel der Konfliktlösung auf den Ebenen Terminologie, Datenformat, Funktion und Prozess [48]. In dieser Arbeit wird die *Mediation* als Vermittlung zwischen Service-Consumer und Service-Provider in Form einer Ermittlung des Service-Endpunkts (siehe Kapitel 2.1.6 auf Seite 30) nach dem Senden des Aufrufs verstanden.

Davenport definiert einen Prozess als messbare Menge von Aktivitäten die dazu dienen, einen bestimmten Kundenkreis am Markt versorgen zu können. Dabei sind Zeit und Raum ebenso wichtige Faktionen wie ein Anfang, ein Ende und klar definierte Ein- bzw. Ausgaben: *„A process is simply a structured, measured set of activities designed to produce a specified output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and place, with a beginning, an end, and clearly identified inputs and outputs“* [32]. Hammer orientiert sich an Davenport und definiert einen

Prozess als eine Menge von Aktivitäten. Der Prozess benötigt zur Zielerfüllung eine oder mehrere unterschiedliche Eingaben und ein Ergebnis, das für den Kunden einen Wert erzeugt [62]. Scheer bezeichnet einen Prozess als *„[...] eine zusammengehörende Abfolge von Unternehmensverrichtungen zum Zweck einer Leistungserstellung. Ausgang und Ergebnis des Prozesses ist eine Leistung, die von einem internen oder externen Kunden angefordert und abgenommen wird“* [134]. Die Daimler AG definiert einen Prozess als maschinell oder manuell zu verrichtende Tätigkeiten, der aus einer oder mehreren Funktionen besteht und einen Anfang sowie ein Ende besitzt [152]. Die Abbildung 4.1 in Kapitel 4 auf Seite 73 illustriert die Strukturierung der Prozesslandschaft in Führungs-, Leistungs- und Unterstützungsprozesse.

Zur Zeit finden sich in der Literatur unterschiedliche Definitionen des Terms Geschäftsprozess. Grundsätzlich kann zwischen zwei unterschiedlichen Interpretationen unterschieden werden. Zum einen die Interpretation als Synonym für den Term Prozess und zum anderen wird der Geschäftsprozess als Kernprozess eines Unternehmens gesehen [156]. Somit besteht ein Geschäftsprozess aus der funktionsüberschreitenden Verkettung wertschöpfender Aktivitäten, die spezifische, von Kunden erwartete Leistungen erzeugen und deren Ergebnisse strategische Bedeutung für das Unternehmen haben. Sie können sich über Unternehmensgrenzen hinweg erstrecken und Aktivitäten von Kunden, Zulieferern oder auch Konkurrenten einbinden [135]. Geschäftsprozesse werden als Kern-, Leistungs-, oder Unternehmensprozesse bezeichnet und beinhalten Kundenanforderungen, Eingaben, Wertschöpfung für das Unternehmen, ein oder mehrere Ergebnisse, genau einen Geschäftsprozessverantwortlichen sowie Ziel und Messgrößen zur Steuerung [152]. In dieser Arbeit wird nicht zwischen Prozess und Geschäftsprozess unterschieden, womit der Term Prozess als Synonym für den Term Geschäftsprozess betrachtet wird.

Der Term Workflow wird von der Workflow Management Coalition (WfMC) [169] als ganz oder teilweise automatisierten Geschäftsprozess definiert: *„The computerised facilitation or automation of a business process, in whole or part“* [70]. Die WfMC standardisierte die XML Process Definition Language (XPDL) als Mitbewerber von BPEL 1.1 (siehe Kapitel 2.1.5 auf Seite 25). Muster *(engl. Pattern)* beschreiben beständig wiederkehrende Entwurfsprobleme und erläutern seine Problemlösung. Analog zur Objekt-orientierten Software-Entwicklung wurden von den P4 („Process Four“) [163] zwanzig Workflow-Muster *(engl. Workflow Patterns)* wie bspw. Sequenz *(engl. Sequence)*, paralleler Prozess-Splitt *(engl. Parallel Split)*, exclusives Oder *(engl. Exclusive Choice)* etc. entwickelt. Die Workflow-Muster der P4 dienten der Business Process Modeling Initiative (BPMI) [118] als Grundlage zur Standardisierung der Business Process Modeling Notation (BPMN) und der Business Process Modeling Language (BPML). Die BPMN ist eine grafische Spezifikationssprache und die BPML, als Mitbewerber von BPEL 1.1 (siehe Kapitel 2.1.5 auf Seite 25), ist eine XML-basierte plattformneutrale Metasprache zur Beschreibung von Prozessen. Havey bildet die beiden Spezifikationen BPMN und BPEL 1.1 aufeinander ab [63].

Initiativen wie *Bottom-Up Approach* [93], *Template Based Composition* [147], *OWL-S Composition Approach* [146], *SHOP2* [148], *METEOR-S* [2], *SEMAPLAN* [5] und *WSMO Approach* [144], welche auf den semantischen Konzepten OWL-S [95], WSDL-S [4] und WSMO [126] basieren, forschen an einer loseren Kopplung zwischen fachlicher Funktionen und der technischen Ausführung. METEOR-S wird in einem aktuellen Folgeprojekt *Semantic Annotations for WSDL (SAWSDL)* [46] fortgesetzt und um die Anwendung von WSDL 2.0 erweitert. Die beiden Initiativen *SWORD* [121] und *Plængine* [99] basieren nicht auf den semantischen Konzepten OWL-S, WSDL-S und WSMO. Die Grundlage von *SWORD* ist unabhängig von semantischen Konzepten und *Plængine* basiert auf einem integrierten Meta-Modell. Durch die Anreicherung der Services mit semantischen Informationen soll die Zielsetzung der Einbindung von Services (in einen Prozess) in der Ablaufumgebung ermöglicht werden. Das Semantic Web Services Framework (SWSF) [165] ähnelt stark dem Konzept OWL-S (siehe Kapitel 2.3.4 auf Seite 45) und besteht aus einem Begriffsmodell (Ontologie) und einer Sprache zur Beschreibung der Web Services. Die Dissertation von A. M. Ahtisham [3] ermöglicht (zur Entwicklungszeit) das Abbilden technischer Aktivitäten auf semantische Konzepte in OWL-S.

Der EMEO-Layer, gebildet aus **E**nterprise Service Bus (ESB), **M**ediation, **E**nterprise Service Repository (ESR) und **O**ntologien, definiert eine logische Schicht *(engl. Layer)* zur Identifizierung äquivalenter Services durch die semantische semiautomatische Unterstützung in einer Service-orientierten Architektur (SOA). Durch die Einbindung der Konzepte einer SOA (ESB, ESR) in Kombination mit einem Entwurfsmuster Mediation und Ontologien stellt der EMEO-Layer einen nicht erforschten neuen Ansatz dar. Der EMEO-Layer basiert, wie *METEOR-S* und *SEMAPLAN*, auf dem semantischen Konzept WSDL-S, grenzt sich jedoch durch die konzeptuelle Integration des semiautomatischen Ansatzes in die Konzepte ESB und ESR sowie durch eine prototypische Implementierung in der Domäne Automotive von diesen ab. Der EMEO-Layer unterstützt keine nicht-funktionalen Anforderungen wie bspw. Antwortzeiten, geografische Lokationen etc., sondern Funktionen und Geschäftsobjekte der Funktionen (funktionale Anforderungen). Funktionen werden auf Serviceoperationen und Vor- (`P`) und Nachbedingung (`E`) auf semantische Konzepte abgebildet. Geschäftsobjekte sind die Eingabe- (`I`) und Ausgabeparameter (`O`) der Funktion und somit der Serviceoperation. Serviceoperationen werden ebenfalls auf semantische Konzepte abgebildet. Die funktionalen Anforderungen Eingabe- und Ausgabeparameter sowie Vor- und Nachbedingung werden in dieser Arbeit als `IOPE` *(engl. Input, Output, Precondition, Effect)* bezeichnet.

Kapitel 3

Bestellanforderung (normalverfügbar)

3.1 Fachlicher Kontext Bestellanforderung (BANF)

Im Folgenden wird ein Geschäftsprozess der Daimler AG untersucht sowie die tangierenden IT-Systeme beschrieben. Es handelt sich hierbei um den Bestellprozess sowie um die Erstellung einer Bestellanforderung (BANF). Die fachlichen Anforderungen dieses Prozesses werden in unterschiedlichen Datenquellen wie bspw. Word, Excel, Intranet etc. dokumentiert. Der Prozessablauf wird durch eine Animation in einem Power-Point-Dokument dargestellt. Um die Heterogenität der Prozessbeschreibung aufzulösen, wurde in mehreren Workshops gemeinsam mit dem Fachbereich ein gemeinsames Verständnis, basierend auf Abbildung 3.1 auf Seite 58, erarbeitet. Die Daimler AG unterteilt die IT-Systeme in die vier Verfügbarkeitsklassen *höchstverfügbar*, *hochverfügbar*, *schnellverfügbar* und *normalverfügbar* (vgl. Tabelle 7.3 auf Seite 132). Wird ein IT-System in die Verfügbarkeitsklasse *normalverfügbar* eingeteilt, so werden bei Unverfügbarkeit ab der 72. Stunde Umsatzeinbußen erwartet. Im Gegensatz dazu impliziert die Verfügbarkeitsklasse *höchstverfügbar* einen Stopp der Produktion und somit sofortige Umsatzeinbußen.

Jeder dieser vier Quader in Abbildung 3.1 auf Seite 58 repräsentiert ein IT-System. Das *MBD (Material-Bestandsführung und Disposition)* steuert die Sachnummern-, Lieferanten- und Verbraucherstammdatenverwaltung, die Lagerverwaltung, Bestandsführung und Inventur, die verschiedenen Dispositionsverfahren, den Bestellabruf beim Lieferanten, den Online-Materialabruf der Verbraucher sowie die Gefahrgut- und Konsignationslager. Das System *WES (Wareneingang und -abgleich-System)* dient zur Online-Erfassung aller Wareneingänge, Rücklieferungen und Ladungsträgerbewegungen (wie bspw. das Verschicken von Gitterboxen und Paletten). Der Verband der Automobilindustrie (VDA) definiert sog. VDA-Formate zum Datenaustausch. Das WES kommuniziert via DFÜ mit den Lieferanten auf Basis der beiden VDA-Formaten 4913 bzw. 4921 und unterstützt den maschinellen oder manuellen Abgleich der Lieferscheindaten sowie die Weitergabe an das Rechnungswesen. Die Eingangsfortschrittszahlen und die Ladungsträgerbestände werden

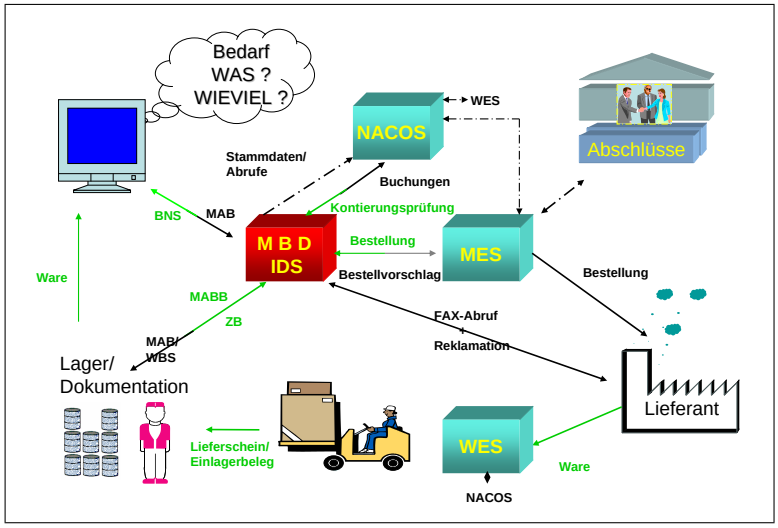

Abbildung 3.1: Bestellprozess Übersicht

an die Disposition und die zentrale Ladungsträger-Bestandsführung weitergegeben. Das System MES (Material-Einkauf-System) verwaltet die Soll- und Ist-Bedarfe und Preise zur Klärung und Rückmeldung sowie die Erstellung von Bestellvorschlägen. Das System NACOS (Neues Accounting- und Controlling-System) basiert komplett auf SAP/R3 und wird das System MBD in seiner Funktionalität ablösen. Das System IDS (Instandhaltung-, Ersatzteile- und Dokumentationssystem) legt die Zuordnung der Bezugsgrößen auf Sachnummern, die zu pflegenden Merkmale der Sachnummern und die Zusatzinformationen wie bspw. Ausweichteile, Bereichstexte, Lieferanten-Nummer-Historie etc. fest. Der Monitor in Abbildung 3.1, stellt eine Benutzerschnittstelle zum System MBD dar. Das abgebildete Haus (Abbildung 3.1) mit dem Titel *Abschlüsse* illustriert die Organisationseinheit Einkauf. Die Fabrik mit dem Titel *Lieferant* symbolisiert alle potentiellen Lieferanten. Der Stapelfahrer und die Behälter stellen die Lagerbewegungen und das Lager dar.

Durch den vorgelagerten Prozess Bestellanforderung (BANF) wird der Bestellprozess initiiert. Somit muss vor dem Auslösen einer Bestellung eine BANF genehmigt werden. Im Anhang I in Abbildung I.2 auf Seite 173 wird das Original illustriert.

Im Folgenden wird die fachliche Grundlage für die Bestellanforderung (BANF) komplett beschrieben (siehe Abbildung 3.2). Abhängig von der BANF-Summe müssen unterschiedliche Genehmigungsstufen durchlaufen werden. Die erste Stufe ist die Manager-Stufe E4 *(E4Role)*; Mitglieder dieser Stufe können eine BANF-Summe im Wert von bis zu 25.000 Euro freigeben. Die zweite Stufe ist die Senior-Manager-Stufe E3 *(E3Role)*; Mitglieder dieser Stufe können eine BANF-Summe im Wert von bis zu 150.000 Euro freigeben. Die dritte Stufe ist die Direktor-Stufe E2 *(E2Role)*; Mitglieder dieser Stufe können eine BANF-Summe im Wert von bis zu 500.000 Euro freigeben. Die vierte Stufe ist die CIO-Stufe E1 *(E1Role)*, Mitglieder dieser

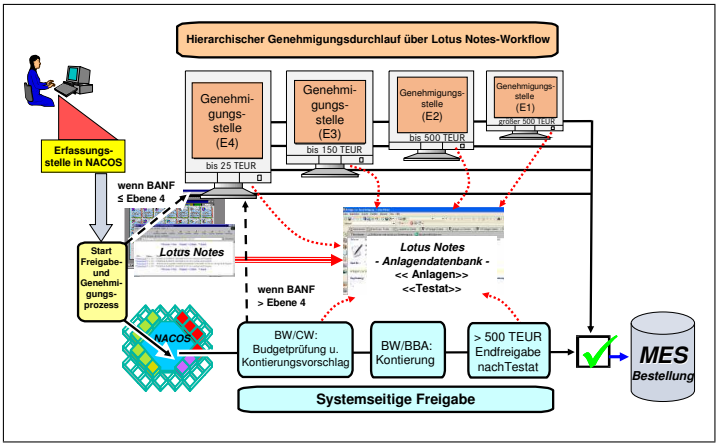

Abbildung 3.2: Fachliche Prozessbeschreibung BANF

Stufe können eine BANF-Summe im Wert von bis zu 5.000.000 Euro freigeben. Der Geschäftsprozess BANF besteht aus den Funktionen *BANF anlegen*, *BANF prüfen und freigeben* sowie *BANF aktualisieren*. Die soeben beschriebenen Rollen (E4Role, E3Role, E2Role und E1Role) werden nacheinander mit der Funktion *BANF prüfen und freigeben* verknüpft. Somit entstehen die fachlichen Funktionen *BANF prüfen und freigeben E3*, *BANF prüfen und freigeben E2* sowie *BANF prüfen und freigeben E1*. Die Rolle Besteller *(BestellerRole)* erstellt das Geschäftsobjekt BANF durch die Eingabe der Sachnummer, der Menge, der Lieferantennummer, des Lieferzeitpunkts etc. Nach der gültigen Speicherung der BANF werden, abhängig von der BANF-Summe, die Genehmigungsstufen durchlaufen. Die Mitglieder der Genehmigungsstufen (E4Role, E3Role, E2Role und E1Role) dürfen das Geschäftsobjekt BANF nicht mehr ändern. Somit wird ein PDF-Dokument generiert, das alle für die Freigabe benötigten Informationen enthält. Zur Simplifizierung des Modells wurde auf die Modellierung der ersten Stufe (Manager-Stufe E4) verzichtet.

Im Folgenden werden alle Ereignisse und Funktionen des BANF-Prozesses beschrieben. Im Anhang I in Abbildung I.2 auf Seite 173 wird das fachliche eEPK illustriert. Die Funktion *BANF prüfen und freigeben E3* hat die drei möglichen Folgeereignisse: *(1) BANF freigegeben und BANF-Summe* $\leq$ *150.000 Euro*, *(2) BANF freigegeben und BANF-Summe* $>$ *150.000 Euro* sowie *(3) BANF zurückgewiesen*. Die beiden Ereignisse (1) und (3) münden direkt in die Funktion *BANF aktualisieren*. Das Ereignis (2) mündet in die Funktion *BANF prüfen und freigeben E2* mit wiederum drei möglichen Folgeereignissen: *(4) BANF freigegeben und BANF-Summe* $\leq$ *500.000 Euro*, *(5) BANF freigegeben und BANF-Summe* $>$ *500.000 Euro* sowie *(6) BANF zurückgewiesen*. Die beiden Ereignisse (4) und (6) münden direkt in die Funktion *BANF aktualisieren*. Das Ereignis (5) mündet in die Funktion *BANF prüfen und freigeben E1* mit genau zwei möglichen Folgeereignissen: *(7) BANF freigegeben und BANF-Summe* $\leq$ *5.000.000 Euro* sowie *(8) BANF zurückgewiesen*. Die beiden Ereignisse (7) und (8) münden abschließend direkt in die Funktion *BANF aktualisieren* [106].

3.2 Durchgängige Realisierung (Prototyp)

3.2.1 Ebene Geschäftsprozess-Modellierung

Die Prozessbeschreibung im Anhang I in Abbildung I.3 auf Seite 173 mangelt an einer formalen Modellierung zur Transformation in BPEL. In gemeinsamen Workshops mit dem Fachbereich und der IDS Scheer wurde aus dem fachlichen Modell, durch manuelle Transformation, ein ablauffähiges fachliches Modell (Service-orientiere eEPK (erweiterte Ereignisgesteuerte Prozessketten)) erstellt. Der *ARIS SOA Architekt* unterstützte die manuelle Transformation sowie die daimler-spezifischen Modellierungsvorgaben zur verbesserten lesbaren Darstellung des Modells:

1. Eine eEPK sollte mit einem Ergebnis beginnen, niemals mit einer Funktion.
2. Rollen, die für eine besondere Aufgabe (Funktion) verantwortlich sind, sollten rechts von Funktionen angeordnet werden. Datenobjekte sollten lesbar dargestellt werden: Eingehende Datenobjekte sollten über ausgehende Datenobjekte links von Funktionen positioniert werden.
3. Teilnehmende IT-Systeme werden ganz links im Modell angeordnet.
4. Fallbedingungen spiegeln sich in den Ereignisnamen wider.
5. Ein XOR-Element trennt die unterschiedlichen Fälle voneinander ab.

Die derzeitig in Betrieb befindliche BANF-Version verwendet zwei Funktionen des SAP Discovery Systems. Das SAP Discovery System ist ein Lösungspaket (Hardware und Software) in Form einer komplett konfigurierten bzw. vorinstallierten Entwicklungs- und Ablaufumgebung. Die erste Funktion ist für die Erstellung einer BANF im SAP Discovery System verantwortlich, die zweite für die Aktualisierung der erstellten BANF. Für die SOA-Integration des Geschäftsprozessmodells ist es erforderlich, diese Funktionen als Web Services zu implementieren und spezifisch mit WSDL-Schnittstellen auszustatten. Der entsprechende WSDL-Code wurde mit Hilfe des WebSphere Integration Developers entwickelt (vgl. die Abbildungen I.10 und I.11 auf Seite 177).

Das ablauffähige fachliche Modell wird in Abbildung 3.3 auf Seite 61 inkl. den beiden Services zum Erstellen und Aktualisieren der BANF illustriert. Das grüne Dreieckssymbol bei den Datenobjekten verweist auf ein verstecktes Modell hinter dem Symbol, welches dieses detaillierter auf einer darunterliegenden Ebene beschreibt. Auf der darunterliegenden Ebene werden Objekte auf Klassen abgebildet.

Die verwendeten Web Services wurden durch das Software-Entwicklungswerkzeug WebSphere Integration Developer entwickelt. Die technische Beschreibung der Web Services (WSDL) wurde in den ARIS SOA Architekt importiert, um diese von dort aus den Funktionen zuweisen zu können. Der ARIS SOA Architekt stellt die Datenobjekte als UML-Klassendiagramm inkl. Attribute dar. Die UML-Darstellung der XML Schema Definition (XSD) des verwendeten Datenobjektes BANF wird im Anhang I in Abbildung I.1 auf Seite 171 grafisch illustriert.

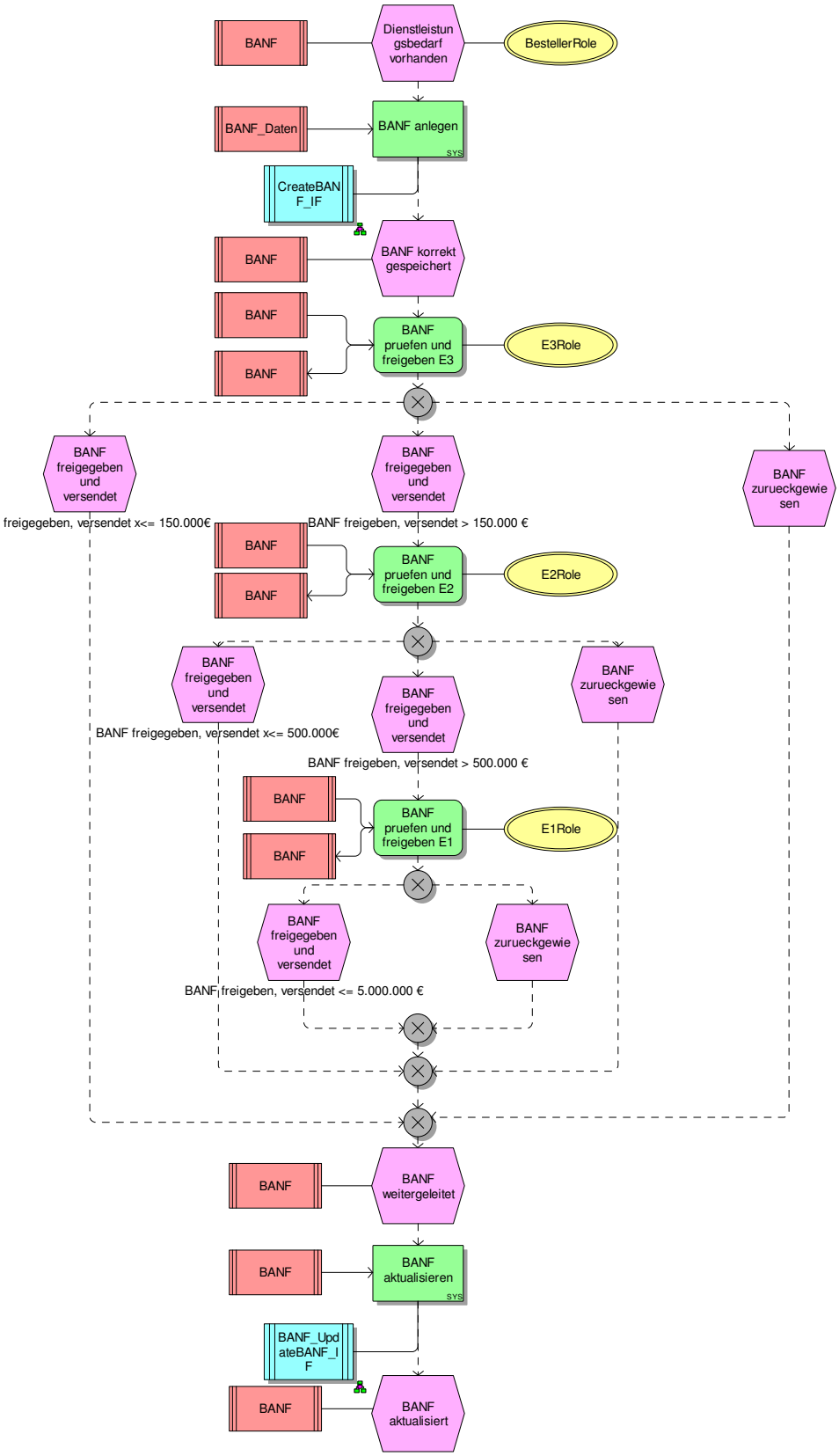

Abbildung 3.3: Service-orientiere eEPK

Im Folgenden wird der Ablauf zur Realisierung des Geschäftsprozesses BANF, beginnend beim bereits entwickelten Modell, technisch beschrieben. Ausgehend vom angereicherten Modell (Service-orientierten eEPK) des Geschäftsprozesses BANF kann BPEL-Code generiert werden. Das Service-orientierte eEPK in Abbildung 3.3 auf Seite 61 erfüllt nach erfolgreicher Validierung nicht alle Voraussetzungen für einen fehlerfreien Import in den WebSphere Integration Developer. So muss noch, vor der Transformation, die Logik zur Fallunterscheidung *(engl. condition expression)* in eine für den WebSphere Integration Developer lesbaren Form erstellt werden. Die Logik zur Fallunterscheidung wird für jeden einzelnen Pfad in Form von Attribut-Informationen definiert. Im Folgenden wird die Fallunterscheidung der Senior-Manager-Stufe E3 (E3Role) dargestellt.

```
bpws:getVariableData("BANF","/freigabe") = true() and
bpws:getVariableData("BANF","/Wert") <= 150000.
```

Der Standard BPEL 1.1 unterstützt keine menschlichen Aktivitäten *(engl. Human Tasks (HT))* (siehe Kapitel 2.1.5 auf Seite 25), womit alle Informationen über die menschlichen Aktivitäten bei der Transformation von einer Service-orientierten eEPK nach BPEL 1.1 verloren gehen. Zur Steuerung des BANF-Prozesses sind menschliche Aktivitäten zwingend notwendig. In Kooperation (IDS Scheer AG, IBM Deutschland GmbH, Daimler AG) wurden zwei proprietäre Skripte entwickelt, um die menschlichen Aktivitäten bei der Transformation nach BPEL 1.1 nicht zu verlieren. Im Anhang I in Abbildung I.9 auf Seite 176 demonstrieren die roten Pfeile die Vorgehensweise im Entwicklungsprozess, die dazu dienen sollen Kompatibilität zwischen dem Tool ARIS SOA Architekt und dem Tool WebSphere Integration Developer herzustellen. Der WebSphere Integration Developer unterstützt bereits Funktionalitäten aus den beiden aktuellen Standards BPEL 2.0 [91] und BPEL4People [73]. Er zeigt jedoch partielle Schwächen in der Abweichung der Konstrukte (BPEL → IBM) `<choice>` → `<switch>`, `<parallel aktivities>` → `<flow>` etc. [90].

Das erste Skript der Firma IDS Scheer AG ermöglicht die Übernahme aller Informationen bzgl. menschlicher Aktivitäten in BPEL. Im Anhang I in Abbildung I.4 auf Seite 174 wird exemplarisch ein Zuweisungsdiagramm *(engl. allocation diagram)* illustriert. Zunächst wird das Startereignis der eEPK ermittelt (*BANF-fachlich-freigeben*). Daraufhin wird ein `<partner link>` mit dem Namen *null* und ein `<wsdlPortType>` (*HumanTaskPT*) erzeugt und mit dem Startereignis im darunterliegenden BPEL Zuweisungsdiagramm verbunden. Das Skript ergänzt den Namensraum für die Attribute des Startereignisses. Für jede BPEL-Erweiterung wird der Name der entsprechenden Organisationseinheit der eEPK als Namensattribut eingefügt. Zu allen hinterlegten BPEL-Zuweisungsdiagrammen werden `<partner links>` mit dem Namen *null* und Operationen (*carryBANF-pruefen-und-freigeben* etc.) hinzugefügt. Im Anhang I in Abbildung I.5 auf Seite 174 wird das erstellte Zuweisungsdiagramm illustriert. Operationsnamen werden mit den Namen der entsprechenden Organisationseinheiten kombiniert (bspw. *carryBANF-pruefen-und-freigeben-E2*), Eingabe und Ausgabevariablen werden erstellt und ein `<port type>`

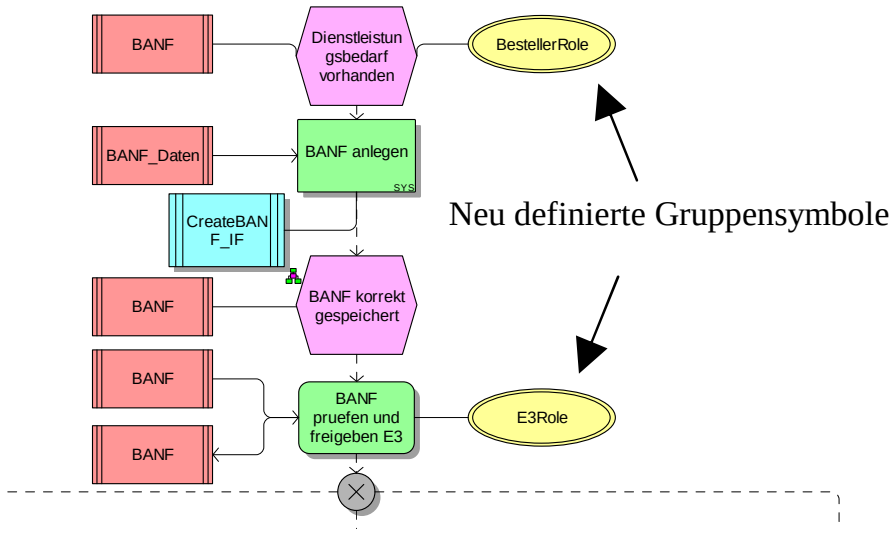

Abbildung 3.4: Organisationseinheiten

spezifiziert. Im Anhang I in Abbildung I.6 auf Seite 175 wird die graphische Darstellung des BPEL-Codes der obersten Ebene illustriert. Der BPEL-Export beinhaltet keine Informationen über die Autorisierung und Authentifizierung der menschlichen Aktivitäten. Durch die Bestimmung weiterer grafischer Gruppensymbole (siehe Abbildung 3.4) wurden gemeinsame (IDS Scheer AG, IBM Deutschland GmbH, Daimler AG) Symbole für Gruppen, Benutzer und weitere Organisationseinheiten bestimmt.

```
<wpc:staff>
  <wpc:potentialOwner>
    <sss:verb>
      <sss:name>Group Members</sss:name>
      <sss:id>Users</sss:id>
      <sss:parameter id="GroupName"><![CDATA[E3Role]]></sss:parameter>
      <sss:parameter id="IncludeSubgroups">false</sss:parameter>
    </sss:verb>
  </wpc:potentialOwner>
</wpc:staff>
```

Das hier dargestellte Beispiel demonstriert die durch Skript 1 ermöglichten BPEL-Erweiterungen. Die Rolle Senior Manager E3 (E3Role) im BANF-Prozess darf nur bestimmte Bestellanforderungen genehmigen, womit Informationen über Autorisierung und Authentifizierung von entscheidender Bedeutung sind. Durch Skript 1 werden diese Informationen in einer Syntax formuliert, um sie in den WebSphere Integration Developer importieren und verarbeiten zu können. Ein zweites Skript ist notwendig, um Informationen über die menschlichen Aktivitäten, aus der generierten WSDL-Datei (*nullLT.wsdl*), mit der generierten BPEL-Datei zu konsolidieren. Weitere marginale Anpassungen sind notwendig, um einen erfolgreichen

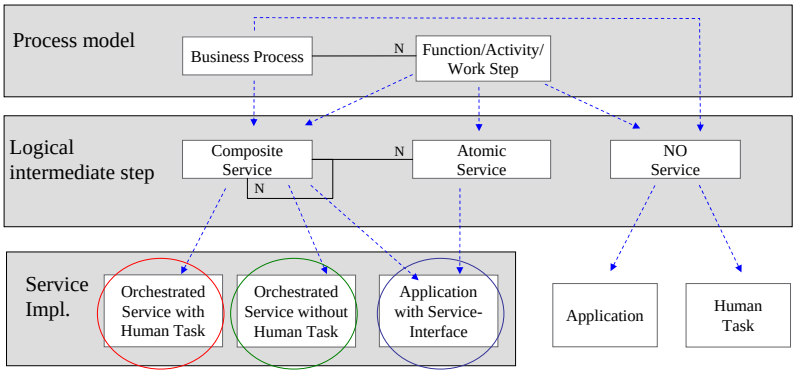

Abbildung 3.5: Übersicht zur Service-Implementierung [68]

Import in WebSphere Integration Developer zu ermöglichen (realisiert durch Skript 2). Nachdem Import der BPEL-Datei in den WebSphere Integration Developer sind noch einige Adaptionen notwendig, um aus dem BPEL-Code, der das erstellte Geschäftsprozessmodell widerspiegelt, einen lauffähigen Workflow zu generieren. Idealerweise sollten nach einem BPEL-Import nur noch kleinere Änderungen vorgenommen werden müssen. Der Aufwand der notwendigen Anpassungen ist Abhängig von der Qualität des Modells und vom Reifegrad der Software-Entwicklungstools.

3.2.2 Ebene Software-Entwicklung

Problem-Klassen der Service-Implementierung. Abbildung 3.5 illustriert drei Problem-Klassen zur Service-Implementierung, die notwendig sind, um einen lauffähigen Workflow an die Workflow-Engine ausliefern zu können. Direkt nach dem BPEL-Import ist die Implementierung der *Problem-Klasse 1* (orchestrierter Service mit menschlichen Aktivitäten) partiell realisiert. Dennoch mussten einige marginale Änderungen vorgenommen werden. Das Geschäftsobjekt BANF wird in einer SAP-Anwendung via Services persistiert. Das Erstellen des Geschäftsobjekts BANF wird über den SAP-Adapter Java-Connector (JCo)) und die Aktualisierung über einen Web Service realisiert. Somit sind die beiden Problem-Klassen Applikation mit Service-Schnittstelle (Problem-Klasse 2) und orchestrierter Service ohne menschlichen Aktivitäten (Problem-Klasse 3) zu realisieren. Die Problem-Klasse 2 nutzt den JCo-Adapter um auf das Business Application Programming Interface (BAPI) der SAP-Anwendung zuzugreifen. Der JCo-Adapter unterstützt Java zu SAP Systemaufrufe sowie SAP-System zu Java Aufrufe. Technisch wurde der BAPI-Aufruf mit einer Web Service Schnittstelle verpackt *(engl. wrapping)* und als WSDL-Datei bereitgestellt. Der Web Service zur Aktualisierung des Geschäftsobjekts BANF (Problem-Klasse 3) konnte sehr leicht in die Infrastruktur integriert werden, da für die Kommunikation kein Adapter (im Gegensatz zur Problem-Klasse 2) gebraucht wurde. Dennoch ist die Abbildung der Schnittstellen nicht trivial. Im Folgenden wird das Geschäftsobjekt BANF sowie die Schnittstellenabbildung beschrieben.

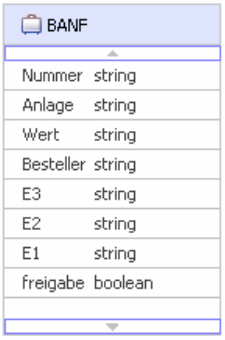

Abbildung 3.6: Geschäftsobjekt BANF

Geschäftsobjekt BANF. Das Geschäftsobjekt BANF hat im SAP-System über vierzig Datenfelder. Für die Implementierung des Prototypen sind die Abbildung der acht Datenfelder (siehe Abbildung 3.6) zur Demonstration ausreichend. Das Datenfeld `Nummer` vom Datentyp `xsd:string` speichert eine eindeutige Nummer zur Zuordnung des Geschäftsobjekts BANF aus dem SAP-System. Das Datenfeld `Anlage` vom Datentyp `xsd:string` speichert einen im Intranet gültigen Pfad auf das PDF-Dokument. Das Datenfeld `Wert` vom Datentyp `xsd:string` speichert aus Gründen der Zugriffshäufigkeit *(engl. performance)* die BANF-Summe. Das Datenfeld `Besteller` vom Datentyp `xsd:string` speichert die ID der Besteller-Rolle. Die Datenfelder `E3`, `E2` und `E1` vom Datentyp `xsd:string` speichern die IDs der Rollen E3Role, E2Role und E1Role. Das Datenfeld `freigabe` vom Datentyp `xsd:boolean` speichert die letzte Eingabe der Rolle E3Role, E2Role oder E1Role.

Schnittstellenabbildung. Für die Implementierung des Prototypen ist die Abbildung der acht Datenfelder (siehe Abbildung 3.6) ausreichend. Das Geschäftsobjekt BANF hat im SAP-System über vierzig Datenfelder. Somit ist eine Schnittstellenabbildung *(engl. interface map)* notwendig, um das Geschäftsobjekt BANF im WID auf das Geschäftsobjekt BANF im SAP-System abzubilden. Schnittstellenabbildungen regeln somit die Kommunikation zwischen Service und IT-System. Bei diesem manuellen Vorgang werden die Datenfelder (Parameter) sowie die Operationen aufeinander abgebildet. Die Möglichkeit, sog. untergeordnete Abbildungen *(engl. submaps)* zu erstellen, bieten Strukturierung sowie Komplexitätsreduzierung in der Visualisierung. Im Anhang I in Abbildung I.7 auf Seite 175 wird auszugsweise eine Abbildung auf das Geschäftsobjekt illustriert. Hier werden `ID` und der `HeaderText` der BANF auf dem SAP-System durch Abbildung zugewiesen. Ergänzend illustriert Abbildung I.8 auf Seite 176 einen BPEL-Code, der explizit zum Aufruf des Web Services erstellt wurde. Die Abbildung 3.7 zeigt die Zuweisung des Geschäftsobjekts BANF WID auf das Geschäftsobjekt BANF im SAP-System. `Move` dokumentiert ein direktes Kopieren des Wertes, `Assign` ein statisches Zuweisen des Wertes und `Custom Assign` eine Manipulation vor der Zuweisung des Wertes.

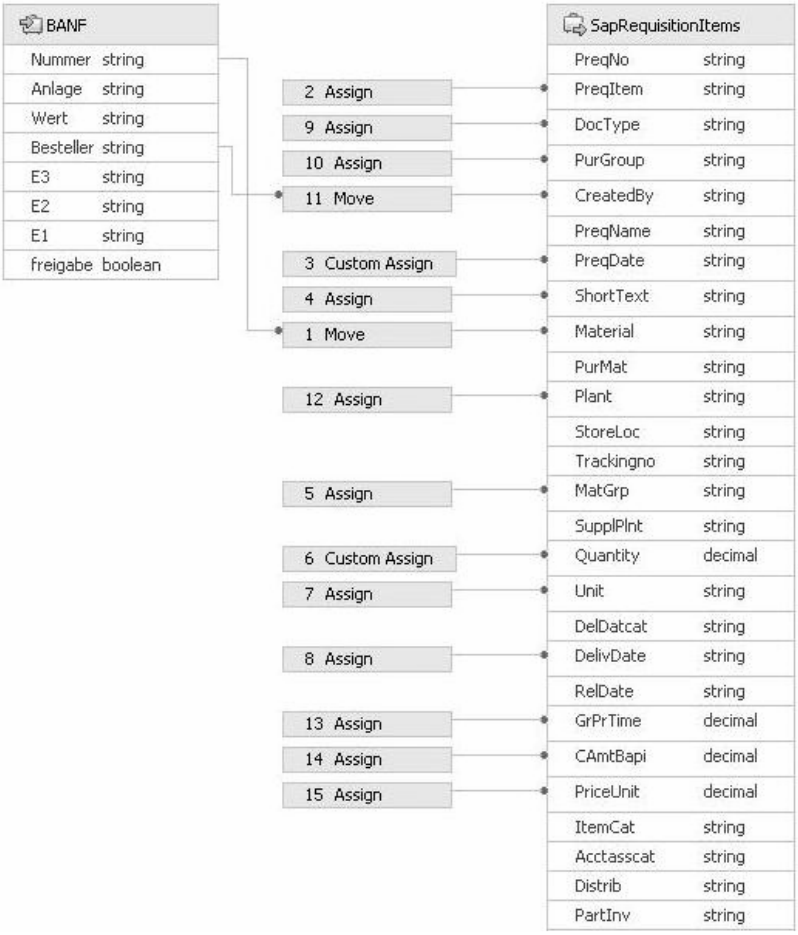

Abbildung 3.7: Abbildung der Geschäftsobjekte

Arrangements. Da beim BPEL-Import Probleme in Bezug auf aufgerufene Web Services auftraten, war ein manueller Import von WSDL-Dateien notwendig. Somit musste eine `Library` mit allen verfügbaren Servicebeschreibungen erstellt und mit dem importierten Projekt durch die Definition von Abhängigkeiten *(engl. dependencies)* verknüpft werden.

3.2.3 Realisierung im WebSphere Integration Developer

Um den Workflow, wie im Folgenden beschrieben, in die Workflow-Engine ausliefern zu können, müssen die Modellierungsvorgaben in Kapitel 3.2.1 auf Seite 60 sowie die Service-Implementierung, das Geschäftsobjekt BANF, die Schnittstellenabbildung und die Arrangements aus Kapitel 3.2.2 auf den Seiten 64, ff. erfüllt sein.

1. Nach dem Import in den WebSphere Integration Developer mussten alle Zugriffe auf `Message-Type`-Variablen in Zugriffe auf `Data-Type`-Variablen umgebaut werden. Dies ist eine Empfehlung der IBM, da `Message-Type`-Variablen im Gegensatz zu `Data-Type`-Variablen vor jedem Zugriff eine spezielle Zu-

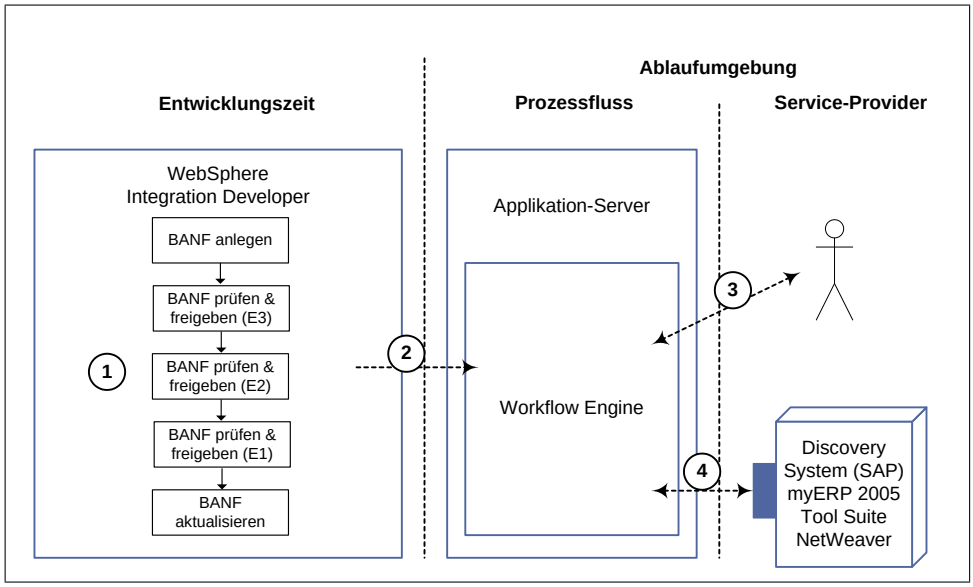

Abbildung 3.8: Statisch: BANF in der Entwicklungs- und Ablaufumgebung

weisung *(engl. assign)* benötigen, um die Inhalte der Variablen verarbeiten zu können. Mit Prozesssymbolen verknüpfte menschliche Aktivitäten wurden fehlerfrei importiert, dagegen wurden die menschlichen Aktivitäten, die mit Ereignissen in Verbindung standen, nicht unterstützt. Somit musste ein Initiator für den Prozess manuell erstellt werden. Ergänzend musste dem Prozess ein Administrator zugewiesen werden, da ohne die Zuweisung der Prozess-Initiator Administrator-Rechte hätte.

2. Nachdem alle weiteren Anpassungen vorgenommen sind, wird ein ablauffähiger Code generiert und der Prozessfluss in die Workflow-Engine ausgeliefert *(engl. deployed)*.

3. Menschliche Aktivitäten werden, abhängig von der BANF-Summe, von der Prozessinstanz bis zu 3-mal angefordert.

   *D*ie externen Services, bereitgestellt vom SAP-System, kommunizieren in Abhängigkeit von den menschlichen Aktivitäten und der BANF-Summe mit der Prozess-Engine.

Die Ablaufumgebung des WebSphere Integration Developer hat eine integrierte Oberfläche, den sog. `BPC-Explorer`, zur Simulation und Verifizierung des Prozess-Flusses. Somit ist eine grafische Benutzeroberfläche nicht zwingend erforderlich. Allerdings verbessert eine grafische Oberfläche, gestaltet nach den Richtlinien der Daimler AG, signifikant die Akzeptanz im Fachbereich. Hier ermöglichen JavaServer Faces (JSF) bzw. JavaServer Pages (JSP) eine standardisierte Entwicklung der grafischen Benutzeroberfläche. Im Anhang I in Abbildung I.12 auf Seite 178 wird exemplarisch eine grafische Oberfläche illustriert.

3.3 Abgeleitete generische Vorgehensweise TD+

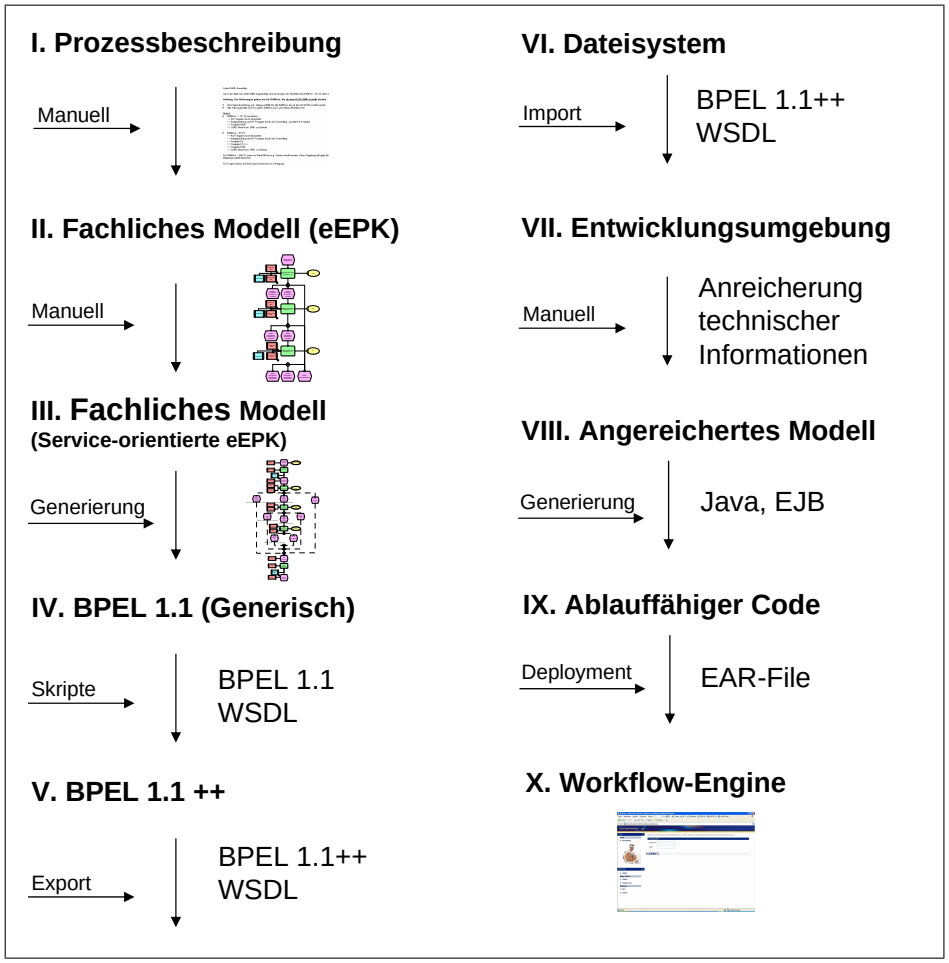

Abbildung 3.9: Generische Vorgehensweise (Top-Down)

TD+ ist ein Akronym gebildet aus dem englischen Term „Top-Down“, wobei das Plus (+) die Durchgängigkeit von der Modellierung bis zur Codegenerierung symbolisiert. Ein gemeinsames Verständnis der fachlichen Anforderung ist notwendig, um den Prozess in Form einer eEPK abbilden zu können. Das Ziel ist die Transformation eines Modells in eEPK in ein ablauffähiges Modell BPEL 1.1. Im Folgenden werden alle Schritte (siehe Abbildung 3.9) beschrieben.

I. Prozessbeschreibung. Die fachlichen Anforderungen sind in unterschiedlichen Datenquellen dokumentiert. Bspw. werden Geschäftsobjekte und Schnittstellen in einem Word-Dokument beschrieben, der Prozess in einem PDF-Dokument dargestellt und der Prozess-Ablauf in einem Power-Point-Dokument animiert. Durch die Heterogenität der Datenquellen ist eine Validierung auf Widersprüche der fachlichen Anforderungen, der Ablauffähigkeit des Prozesses sowie der Konsistenz der fachlichen Anforderungen nicht möglich.

II. Fachliches Modell (eEPK). Um die Heterogenität der Prozessbeschreibung aufzulösen, wurde in mehreren Workshops, gemeinsam mit dem Fachbereich, ein Prozess-Modell in der ARIS-Methode eEPK erstellt und im Tool *ARIS SOA Architekt* modelliert. Hierbei wurden die bereits allgemeingültigen und internen Richtlinien zur Modellierung der Prozesse in der ARIS-Methode eEPK angewandt. Das erstellte fachliche Modell dient für die Diskussion mit dem Fachbereich und ist noch kein ablauffähiges Modell [143]. Im Folgenden werden die Restriktionen aufgezählt, die bei der Modellierung einer eEPK eingehalten werden müssen (siehe Tabelle 3.1).

Struktur-Regeln	**Beschreibung**
Funktionen bzw. Ereignisse haben nur eine eingehende bzw. ausgehende Kante.	Diese Regel prüft, ob alle Funktionen und Ereignisse maximal eine eingehende oder ausgehende Kante haben.
Jeder Pfad muss mit einem Ereignis beginnen und enden.	Diese Regel prüft, ob alle Pfade mit einem Ereignis beginnen und enden.
Nach einem Ereignis kein OR bzw. XOR möglich.	Diese Regel prüft, ob keine auftrennenden OR- bzw. XOR-Regeln *(engl. distributors)* nach einem Ereignis innerhalb eines Prozesses auftreten.
Kein Objekt darf ohne Kante existieren.	Diese Regel prüft, ob ein Modell Objekte ohne Kanten zu anderen Objekten enthält. Jedes Objekt in einem Modell muss eine oder mehrere Vorgänger und/oder Nachfolger haben.
Zahl der eingehenden oder ausgehenden Kanten bei einer Regel.	Diese Regel prüft, ob es genau eine eingehende und minimal zwei ausgehende Kanten bei jeder einfachen Regel gibt, oder minimal zwei eingehende und genau eine ausgehende Kante.

Tabelle 3.1: Strukturregeln einer eEPK

III. Fachliches Modell (Service-orientiere eEPK). Um eine allgemeingültige Transformation von einer Service-orientierten eEPK nach BPEL 1.1 durchführen zu können, müssen erweiterte Regeln (siehe Tabelle 3.2) angewandt werden. Die Anwendung der erweiterten Regeln ist nicht automatisiert und wurde somit manuell durchgeführt. Das entstandene Modell wurde erneut mit dem Fachbereich diskutiert, um die fachliche Konsistenz und die Akzeptanz des Fachbereichs sicherzustellen.

Struktur-Regeln	**Beschreibung**
Eine Geschäftsfunktion sollte von einer einzigen Organisationseinheit ausgeführt werden.	Nur eine Geschäftsfunktion, die mit einem einzigen Objekt vom Typ Organisationseinheit über das Verhältnis *führt aus* verbunden ist, wird bei der Transformation des BPEL Prozesses interpretiert.

Tabelle 3.2: Regeln einer Service-orientierten eEPK (1)

Struktur-Regeln	**Beschreibung**
Eine Systemfunktion wird von einem einzigen Objekt vom Typ *Anwendungs-System-Typ* unterstützt.	Eine Systemfunktion darf nur von einem einzigen Objekt vom Typ *Anwendungs-System-Typ* unterstützt werden und wird dann bei der Transformation des BPEL Prozesses interpretiert.
Alle Ein- und Ausgabeobjekte müssen auf einen Typ *Klasse* abgebildet werden oder müssen selbst Objekte des Typs *Klasse* sein.	Alle Objekte, die mit Funktionen über die Beziehungen vom Typ *hat Input* oder *hat Output* verbunden sind, sollten vom Typ *Klasse* sein oder direkt oder indirekt auf einen oder mehrere Objekte vom Typ *Klasse* abgebildet werden, um korrekt bei der Transformation des BPEL-Prozesses interpretiert zu werden.
Ein Objekt vom Typ *Anwendungs-System-Typ*, das eine Systemfunktion unterstützt, darf mit nur einem Objekt vom Typ *Komponente* verbunden werden.	Nach den Modellierungsvereinbarungen für die Repräsentation eines Services in ARIS darf nur ein einziges Objekt vom Typ *Komponente* mit einem Objekt vom Typ *Anwendungs-System-Typ* über die Beziehung vom Typ *umfasst* verbunden werden.
Nur ein einziges Objekt vom Typ *Operation* wird mit einer Systemfunktion verbunden.	Nur ein einziges Objekt vom Typ *Operation* kann mit einer Systemfunktion verbunden und bei der Transformation des BPEL Prozesses interpretiert werden.
Nur spezifische Funktions-Symbol-Typen werden verwendet.	Nur die Geschäftsfunktion, die Systemfunktion und das Systemfunktionsziel sind für die Transformation des BPEL Prozesses erlaubt.
Prozess enthält öffentliche Datentransferaktivitäten.	Öffentliche Datentransferaktivitäten dienen der Spezifizierung der Ein- und Ausgabeinformation des Prozesses. Diese Information wird von anderen Komponenten verwendet, welche mit dem Prozess kommunizieren. Nach den Modellierungsvereinbarungen für Service-orientierte EPKs können öffentliche Datentransferaktivitäten Prozessschritte repräsentieren, die sowohl Startereignissen und vorhergehenden Endereignissen folgen und welche die zugehörigen Ein- und Ausgabeobjekte des Prozesses spezifizieren. Alternativ können die Ein- und Ausgabedaten des Prozesses als ein Datenobjekt des Start- oder Endereignisses spezifiziert werden.
Parallele Prozessflüsse, inklusive Entscheidungspfade und exklusive Entscheidungspfade sind wohlgeformt.	Parallele Prozessflüsse sollten durch das Auftrennen und Zusammenführen von AND/XOR Regeln spezifiziert werden. Oder sie sollten entweder eine auftrennende AND/XOR Regel enthalten, für welche es keine andere Kante zwischen deren Pfaden gibt, oder eine zusammenführende AND/XOR Regel, welche von allen Kanten getroffen wird.

Tabelle 3.3: Regeln einer Service-orientierten eEPK (2)

IV. BPEL 1.1 (Generisch). Das ablauffähige fachliche Modell wurde toolbasierend validiert und nach BPEL 1.1 transformiert. Die Validierung prüft eine mögliche Abbildung des eEPK-Modells nach BPEL 1.1 und dokumentiert das Ergebnis in der Hypertext-Auszeichnungssprache *(engl. Hypertext Markup Language (HTML))*. Die Transformation bildet das eEPK-Modell nach BPEL 1.1 ab und stellt das neue Modell grafisch dar. Die grafische Darstellung in BPEL 1.1 ist, im Gegensatz zu den Elementen und Attributen, nicht standardisiert. Somit weicht die grafische Darstellung von BPEL 1.1 in den unterschiedlichen Tools teilweise stark voneinander ab. Der Prozess wird in BPEL 1.1 und als Web Service in Form einer WSDL beschrieben.

V. BPEL 1.1++. Der aktuelle Standard BPEL 1.1 unterstützt keine menschlichen Aktionen *(engl. Human Tasks)* und wird teilweise von den Herstellern unterschiedlich interpretiert bzw. implementiert. Daraus folgt ein mangelnder Reifegrad des aktuellen Standards und eine mangelnde Unterstützung an den Toolgrenzen. In gemeinsamer Absprache mit der IDS Scheer AG und der IBM Deutschland GmbH wurden zwei proprietäre Skripts entwickelt, die beide Mängel beseitigen. Das Fehlen der menschlichen Aktionen wurde durch Skript 1 (basierend auf BPEL4People [73]) und die mangelnde Unterstützung an den Toolgrenzen durch Skript 2 gelöst. Der Prozess wurde somit in *BPEL 1.1++* transformiert und wird selbst als Service in Form einer WSDL beschrieben.

VI. Dateisystem. Die Artefakte aus dem Export sind Dokumente (*BPEL 1.1++*, WSDL) und ein neu angelegtes Verzeichnis (*Imports*). Im Verzeichnis *Imports* werden alle Service-Beschreibungen (WSDL) abgelegt, die vom Prozess aufgerufen werden. Beide Dateien werden in das Datei-System des Betriebssystems exportiert und von dort aus in die Entwicklungsumgebung WebSphere Integration Developer (WID) importiert.

VII. Entwicklungsumgebung. Nach dem erfolgreichen Import in den WID wird das Modell mit weiteren technischen Informationen angereichert: Auswahl der Service-Operation, Zuordnung *(engl. assign)* der Variablen an die Service-Operationen sowie Ersetzen der Message-Type durch Data-Type Variablen. Während der technischen Anreicherung ist eine iterative toolbasierte Generierung notwendig, um die manuelle technische Anreicherung regelmäßig zu verifizieren.

VIII. Angereichertes Modell. Nach kompletter Anreicherung der technischen Informationen im WID erstellt die toolbasierte Generierung EJB- und Java-Klassen bzw. EJB- und Java-Objekte.

IX. Ablauffähiger Code. Das Grafische User Interface (GUI) wird bspw. mit JavaServer Faces bzw. JavaServer Pages, basierend auf den generierten Klassen und Objekten, entwickelt. Abschließend werden alle generierten Klassen bzw. Objekte sowie alle JSP's in ein EAR-File komprimiert und in die bereits gestartete *(engl. running)* Workflow-Engine ausgeliefert *(engl. deployed)*.

X. Workflow-Engine Die gestartete Workflow-Engine de-komprimiert das EAR-File und ermöglicht somit das Erstellen einer oder mehrerer Prozess-Instanzen über das selbst entwickelte grafische User Interface [31].

Neben diesem Ansatz zur Transformation eines fachlichen (EPK) in einen technischen Prozess (BPEL), schlägt Maurer [96] eine erste Transformation in UML mit anschließender Transformation in einen technischen Prozess (BPEL) vor. Dieser Ansatz wurde nicht weiter untersucht. Die soeben vorgestellte generische Top-Down-Vorgehensweise TD+ in Verbindung mit der praktischen Anwendung (BANF) in Form eines Prototypen wurde bei dem Wettbewerb „Bestes SOA-Industrieprojekt 2007“ auf dem *Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (BITKOM)* [17] in der Kategorie „Cross-Vendor“ mit dem ersten Preis ausgezeichnet. Der Preis wurde am 15.11.2007 auf dem Jahresabschluss-Event der BITKOM-Hauptvorstands Taskforce SOA in Berlin entgegengenommen (Im Anhang I in Abbildung I.13 auf Seite 179).

Kapitel 4

Auftragsplanung Getriebe (höchstverfügbar)

Definition 4 *In der Automobilindustrie wird unter* ***Powertrain*** *der Antriebsstrang eines Fahrzeuges verstanden. Dies beinhaltet alle Komponenten, die im Fahrzeug das Drehmoment des Motors auf die Straße übertragen. Bspw. Schwungrad, Kupplung, Getriebe, Differential, Seitenwellen, Gelenkwelle, Kardanwelle (Heckantrieb oder Allrad) und die Räder.*

4.1 Fachlicher Kontext Auftragsplanung Getriebe

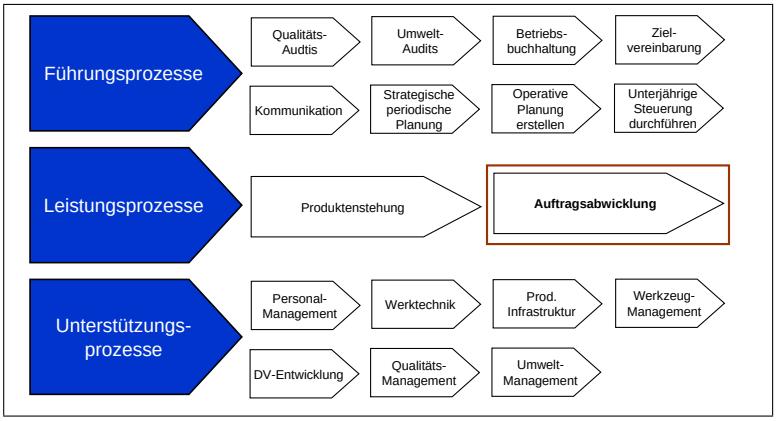

Abbildung 4.1: Übersicht Prozesse *Powertrain*

Dieses Modell wurde hierarchisch von Oben *(engl. top down)*, auf der Basis Übersichtsmodelle in Form der Wertschöpfungskettendiagrammen sowie weitere (tiefere) Ebenen konzipiert. Abbildung 4.1 illustriert die Prozesslandschaft sowie die einzelnen Center in der Domäne Powertrain (im Anhang J auf Seite 181 wird zur besseren Ansicht eine vergrößerte Grafik dargestellt). Für die Darstellung der Prozessmodelle werden je nach Detaillierungsgrad Wertschöpfungskettendiagramme (WKD) oder erweiterte Ereignisgesteuerte Prozessketten (eEPK) verwendet. Ziel der Modellierung ist es, eine einheitliche und verbindliche Dokumentation aller Prozesse

innerhalb der Domäne Powertrain darzustellen. Die Abläufe im Werk Untertürkheim sind dabei transparent und durchgängig über alle Center hinweg aus Kunden- und Produktsicht beschrieben. Durch die Integration aller prozessrelevanten Dokumentationen (bspw. Arbeitsanweisungen) ist der Zugriff auf die jeweils aktuelle Version gewährleistet.

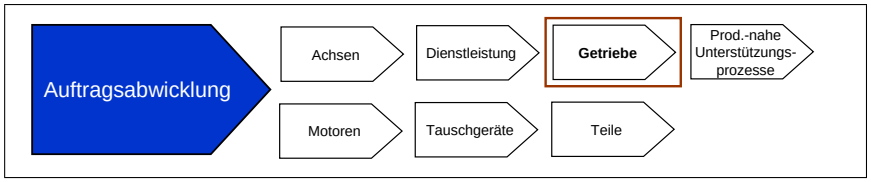

Abbildung 4.2: *Powertrain → Auftragsabwicklung*

Die Prozesslandschaft ist in mehrere Prozess-Struktur-Ebenen gegliedert. Prozess-Struktur-Ebene 1 (siehe Abbildung 4.1) zeigt die Gesamtübersicht über alle Werksprozesse, gegliedert nach den Führungs-, Leistungs- und Unterstützungsprozessen. Prozess-Struktur-Ebene 2 (siehe Abbildung 4.2) zeigt eine Differenzierung der Leistungsprozesse in der *Auftragsabwicklung*. Auf der dritten Prozess-Struktur-Ebene (siehe Abbildung 4.3) wird der Prozess *Getriebe* verfeinert dargestellt. Die Verfeinerung besteht in der Darstellung des Vorgängers und Nachfolgers eines Prozesses.

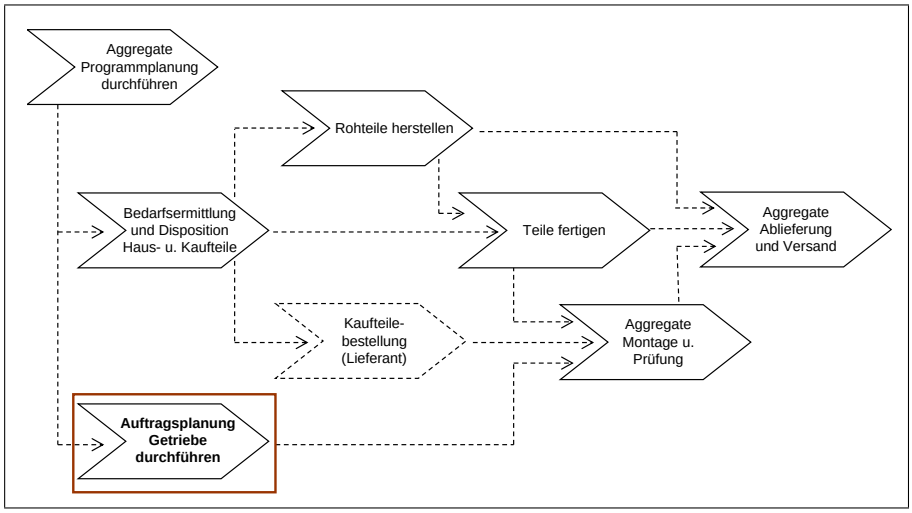

Abbildung 4.3: *Powertrain → Auftragsabwicklung → Getriebe*

In der Auftragsabwicklung wird zwischen Achsen, Motoren, Tauschaggregaten, Getrieben und Teilen unterschieden. Die Motoren lassen sich in Ottomotoren (Reihen 3 Zylinder, Reihen 4 Zylinder, V 6 Zylinder etc.) und Dieselmotoren unterteilen. Fasst man alle Baureihen wie bspw. Smart, A-Klasse, C-Klasse, E-Klasse, S-Klasse etc. in der Mercedes Car Group (MCG) und den Nutzfahrzeugen *(engl. Commercial Vehicle Division (CVD))* zusammen, so beträgt die Anzahl der Getriebe-Varianten

(NAG1, NAG-V, NAG2, NAG2i, FSG, F-CVT etc.) zweihundert. Alleinstellungsmerkmale sind Rechts-Lenker, Links-Lenker, Abgasvorschriften etc. Ein Baumuster (BM) kann eine oder mehrere Getriebe-Varianten beinhalten. Eine Getriebe-Variante passt genau zu einem BM. Die eindeutige Identifizierung (Primär Schlüssel) findet bei einem PKW über die Fahrgestellnummer, bei einem Teil oder Aggregat über die Sachnummer (SNR), bei Varianten über die Varianten-Nummer und bei einem Baumuster über die BM-Nummer statt. Zu einem späteren Zeitpunkt in der Wertschöpfungskette findet die ID-genaue Steuerung der Produktion statt. Dies bedeutet, dass alle zum Zusammenbau eines PKW benötigten Komponenten wie Motor, Karosserie, Spiegel etc. genau aufeinander abgestimmt (und somit in ihrer Reihenfolge nicht mehr austauschbar) sind. Die Produktion von Getrieben muss intern sehr detailliert geplant werden. Im Folgenden wird dieser Prozess näher beschreiben.

4.1.1 Prozess *Auftragsplanung Getriebe durchführen*

Der Prozess *Auftragsplanung Getriebe durchführen* wird täglich durchlaufen und plant die Produktion der Getriebe für 10 Tage im Voraus. Prozessziel ist, auf äußere Einwirkungen wie Änderungen der Lieferabrufe (Produktionsauftragsdatenbank Plan) sowie Lieferengpässe und höhere Gewalt sehr zeitnah reagieren zu können. Bei täglicher Neuplanung werden in Abhängigkeit der Teileverfügbarkeit, der Getriebebedarfe (Lieferabrufe Produktionsauftragsdatenbank Plan) und der werkspezifischen Montagerestriktionen (Anzahl der Mitarbeiter, Kapazitätsgrenzen, Verfügbarkeit der Anlagen, Min-Losgröße, Max-Losgröße und Rüstzeiten) Montageaufträge erstellt sowie die Montagenreihenfolge und die Montagelosgrößen bestimmt. Der Prozess *Auftragsplanung Getriebe durchführen* stellt einen Kernprozess des Unternehmens dar und grenzt sich somit, bezüglich des Effizienzgrades, von Mitbewerbern in der Domäne Automotive ab. Der Ablauf und die Reihenfolge der einzelnen elf Prozessschritte (siehe Abbildung 4.4) wird als vertraulich eingestuft; diese werden somit im Folgenden nur skizzenhaft erläutert.

Die Teilebedarfsermittlung (TBE) übergibt, mit dem Ziel der Plausibilitätsprüfung, einen oder mehrere Datensätze, in der Form `Sachnummer`, `Menge` und `Lieferdatum`, an die erste Funktion *Getriebe-Bedarfe bzgl. Plausibilität prüfen*. Eine der zu prüfenden Restriktionen ist bspw. `Menge > Produktionskapazitätsgrenze`? Um eine ausreichende Flexibilität der Produktion zu gewährleisten, besteht bspw. die Möglichkeit Getriebe in einer anderen Losgröße zu produzieren als vom System (Funktion) vorgeschlagen. Durch diese potentielle Nichteinhaltung der Restriktionen erhält die Funktion *Klärung und Korrektur der Aggregate-Bedarfe* das Start-Ereignis mit dem Ziel, Klärung bzw. Korrektur (falls notwendig) eines Datensatzes aus der TBE. Durch die Einhaltung der Restriktionen (Start-Ereignis *Mengen sind plausibel*) werden zwei parallele Funktionen aufgerufen. Die erste Funktion *Vorlaufrechnung Soll-Versand-Termin durchführen* errechnet den `Versand-Termin` passend zum `Liefertermin` aus der TBE. Die zweite Funktion *Vorlaufrechnung Soll-Montage-Termin durchführen* errechnet, analog zur Funktion *Vorlaufrechnung Soll-*

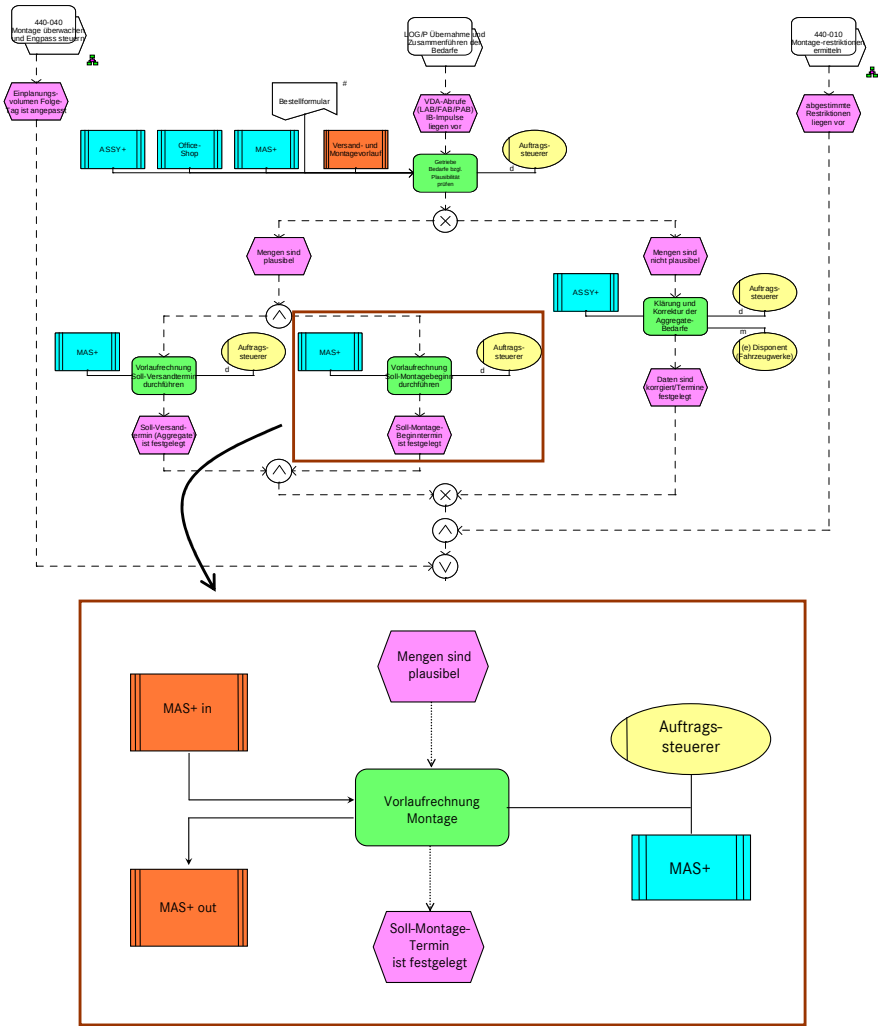

Abbildung 4.4: *Auftragsplanung Getriebe (Teil 1)*

Versand-Termin durchführen, den `Montage-Termin` passend zum `Liefertermin` aus der TBE. Somit ist der `Montage-Termin` ein Datum und liegt aus Sicht des `Liefertermin` in der Vergangenheit.

Die Funktion *Vorlaufrechnung Soll-Montage-Termin durchführen* dient als fachliche Grundlage für die Umsetzung der Top-Down-Vorgehensweise TD+ (siehe Kapitel 4.2 auf Seite 79), die Konzeption und Architektur des EMEO-Layers (siehe Kapitel 6 auf Seite 85) und für die Validierung des EMEO-Layers (siehe Kapitel 7 auf Seite 99). Im Folgenden wird die Funktion *Vorlaufrechnung Soll-Montage-Termin durchführen*, in Form von Vor- (`P`) bzw. Nachbedingung (`E`) sowie Ein- (`I`) bzw. Ausgabeobjekt (`O`), beschrieben und zur Vereinfachung im Folgenden als *Vorlaufrechnung Montage* bezeichnet.

Die Abbildung 4.4 illustriert den ersten Teil des Prozesses grafisch und vergrößert die Funktion *Vorlaufrechnung Montage* zur besseren Darstellung. Bei der Vergrößerung wurde die Funktion bereits von einem *fachlichen Modell* in ein *ablauffähiges*

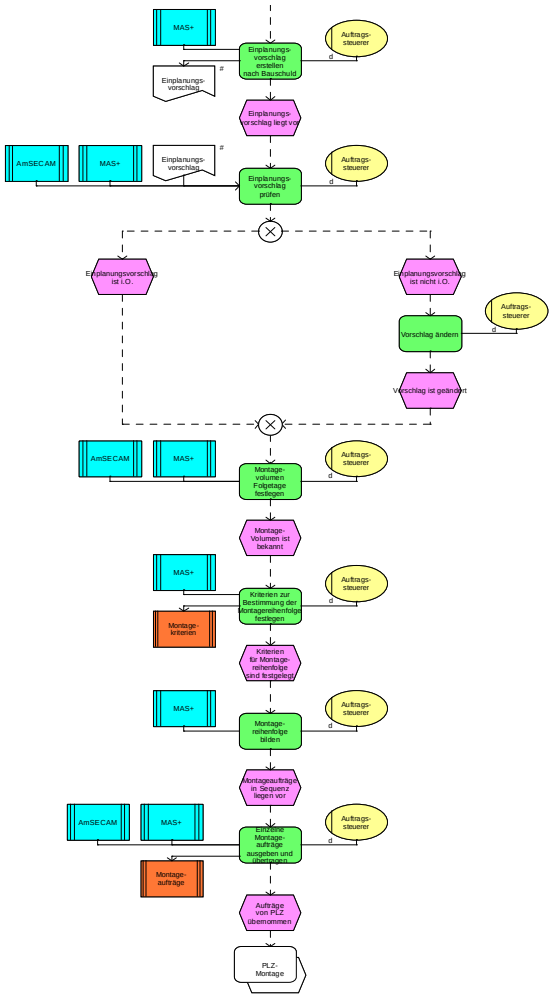

Abbildung 4.5: *Auftragsplanung Getriebe (Teil 2, Fortsetzung Abbildung 4.4)*

Modell manuell transformiert (siehe Kapitel 3.3 auf Seite 68), um die Funktion in Kapitel 4.1.2 technisch beschreiben zu können. Nach der Errechnung des `Versand-Termin` und des `Montage-Termin` wird ein Einplanungsvorschlag, durch die Funktion *Einplaungsvorschlag nach Bauschuld erstellen*, in Form einer Simulation der produktionsbeeinflussenden Restriktionen (bspw. Min-Losgröße, Max-Losgröße und Kapazitätsgrenzen) simuliert. Diese Simulation ist zeitsparend und liefert bereits einen realistischen Einplanungsvorschlag. Basierend auf diesem Einplanungsvorschlag wird ein zeitintensiver konkreter Einplanungsvorschlag durch die Funktion *Einplanungsvorschlag prüfen* erstellt. Ermittelt die Validierung einen Fehler, so wird der Einplanungsvorschlag durch die Funktion *Vorschlag ändern* und der fachlichen Rolle *Auftragssteuerer* angepasst. Aus dem Einplanungsvorschlag folgernd, wird durch die Funktion *Montagevolumen der Folgetage festlegen* das Produktionsvolumen für die Folgetage errechnet (basierend auf den TBE-Berechnungen). Vorhandene Rüstzeiten und Losgrößen sind beispielhafte Parameter für die Funktion

Kriterien zur Bestimmung der Montagereihenfolge festlegen, die nach Festlegung das Ziel zur Erstellung der Montagereihenfolge durch die Funktion *Montagereihenfolge bilden* ermöglicht. Die Funktion *Einzelne Montageaufträge ausgeben und übertragen* schließt den Prozess *Auftragsplanung Getriebe durchführen* ab, indem die Aufträge der zu produzierenden Getriebe an die Produktionsleitzentrale (PLZ) übergeben werden. Die Abbildung 4.5 illustriert den zweiten Teil des Prozesses grafisch.

4.1.2 Funktion *Vorlaufrechnung Montage*

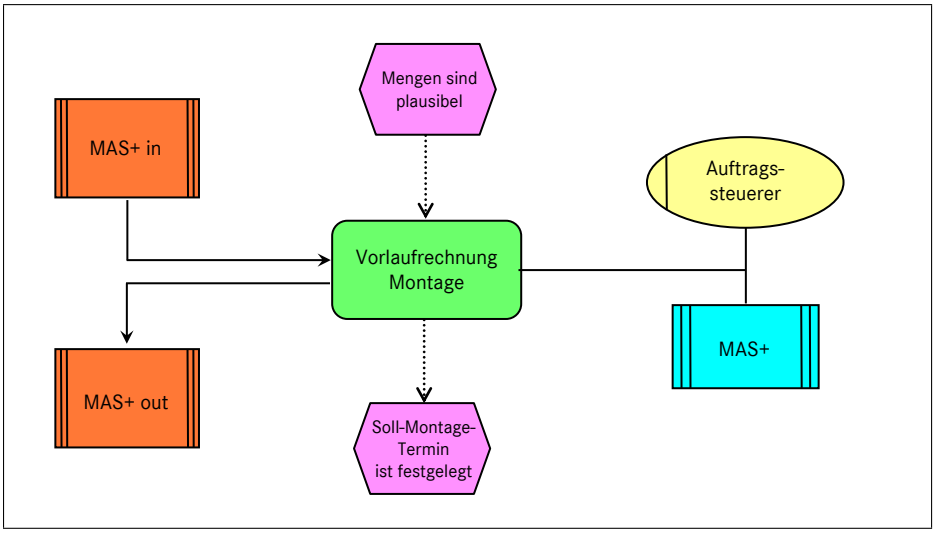

Abbildung 4.6: Funktion *Vorlaufrechnung Montage*

Die Funktion sowie die Geschäftsobjekte (*MAS+ in*, *MAS+ out*) werden vom IT-System *Montageauftragssteuerung Plus*, im Folgenden *MAS+* genannt, bereitgestellt. Das „+“ dokumentiert bereits die zweite Generation des Systems MAS mit erweiterten fachlichen Anforderungen. Das Geschäftsobjekt *MAS+ in* definiert die Eingabefelder illustriert durch die gerichtete Kante in Richtung Funktion für die Funktion *Vorlaufrechnung Montage*. Ausgehend von der Funktion *Vorlaufrechnung Montage*, illustriert die gerichtete Kante in Richtung Geschäftsobjekt *MAS+ out*, die Ausgabefelder. Vor- und Nachbedingung der Funktion werden durch die beiden Sexagons, *Mengen sind plausibel* und *Soll-Montage-Termin ist festgelegt*, dargestellt. Die Rolle *Auftragssteuerer* ist mit dem IT-System *MAS+* verbunden und führt die Funktion *Vorlaufrechnung Montage* aus. Im Folgenden werden die beiden Geschäftsobjekte *MAS+ in* und *MAS+ out* plattformneutral beschrieben. Die Geschäftsobjekte *MAS+ in* und *MAS+ out* der Funktion *Vorlaufrechnung Montage* definieren folgende Datenfelder:

```
MASPlus in:

Sachnummer xsd:String
Menge      xsd:int
Termin     xsd:DateTime
```

```
MASPlus out:

Soll-Termin xsd:DateTime
```

4.2 Umsetzung der Top-Down-Vorgehensweise TD+

Der *Auftragssteuerer* muss, um den Prozess *Auftragsplanung Getriebe durchführen* (siehe Kapitel 4.1.1 auf Seite 75) durchführen zu können, mehrere IT-Applikationen bedienen. Diese Vorgehensweise ist fehleranfällig, da es Übertragungsfehler an den Schnittstellen geben kann. Der service-orientierte Ansatz verspricht, durch die Zentralisierung der Steuerung, die Eliminierung der Übertragungsfehler. Generell wird die Software-Entwicklung in die Phasen Analyse, Design, Entwicklung sowie Deployment eingeteilt. Optional ist ein Vorprojekt, das Prototypen und/oder eine Untersuchung zur Machbarkeit beinhalten kann [66]. Im Folgenden wird die technische Anreicherung des Prozess-Schritts *Vorlaufrechnung Montage* (siehe Kapitel 4.1.2 auf Seite 78) in der Entwicklungsumgebung beschrieben. Die generische Top-Down-Vorgehensweise (siehe Kapitel 3.3 auf Seite 68) beschreibt in *Schritt I.* die Analysephase, in den Schritten *II.* – *V.* die Designphase, in den Schritten *VI.* – *VIII.* die Entwicklungsphase sowie im Schritt *IX.* und im *Schritt X.* das Deployment.

In der folgenden Darstellung (siehe Abbildung 4.7 auf Seite 80) nehmen wir an, dass Analyse- und Designphase *(Schritte I. – V.)* bereits realisiert wurden, womit alle durch Services realisierte Funktionen bereits im eEPK-Modell mit der plattformneutralen Service-Beschreibung verbunden sind. Ferner nehmen wir an, das Modell wurde bereits mit allen notwendigen technischen Informationen *(Schritt VI. und partiell Schritt XII.)* angereichert.

Der Enterprise Service Bus ist die Ablaufumgebung für die *Lookup-Komponente* und die Workflow Engine ist die Ablaufumgebung für den Prozessfluss. Die hier illustrierte *Lookup-Komponente* besteht aus genau einem Eingang (E_1), genau zwei Ausgängen (A_1, A_2) und ermöglicht das Auslesen des Service-Endpunkts (IP-Adresse, Protokoll, Port). A_1 repräsentiert ein positives Ergebnis (Service-Endpunkt konnte ermittelt werden) und A_2 repräsentiert ein negatives Ergebnis (Service-Endpunkt konnte nicht ermittelt werden) – A_2 wird nicht näher betrachtet.

Die Abbildung 4.7 auf Seite 80 zeigt die exemplarische Verbindung mit der *Lookup-Komponente* am Prozess-Schritt *Vorlaufrechnung Montage* (siehe Kapitel 4.1.2 auf Seite 78). Um eine Flexibilisierung des kompletten Prozesses zu erreichen, ist der Einbau der *Lookup-Komponente* bei jeder Servicenutzung notwendig und schließt *Schritt VII.* ab. Die *Schritte VIII.* – *X.* weichen nicht von der generischen Vorgehensweise in Kapitel 3.3 ab. Im Folgenden wird der Einbau und das Verhalten der *Lookup-Komponente* (siehe Kapitel 2.1.7 auf Seite 32) in der Entwicklungsumgebung und in der Ablaufumgebung beschrieben. Der linke Teil der Grafik 4.7 auf Seite 80 stellt die Entwicklungsumgebung, der rechte Teil die Ablaufumgebung dar.

1. Die Verbindung zwischen Serviceaufruf *Vorlaufrechnung Montage* und Service `WSDL_MAS_PLUS` wird entfernt. Eine neue Verbindung zwischen Serviceaufruf *Vorlaufrechnung Montage* und *Lookup-Komponente* wird hergestellt.

2. Durch das Verbinden der *Lookup-Komponente* mit dem Service `WSDL_MAS_PLUS` wird der *Schritt VII.* komplett abgeschlossen.

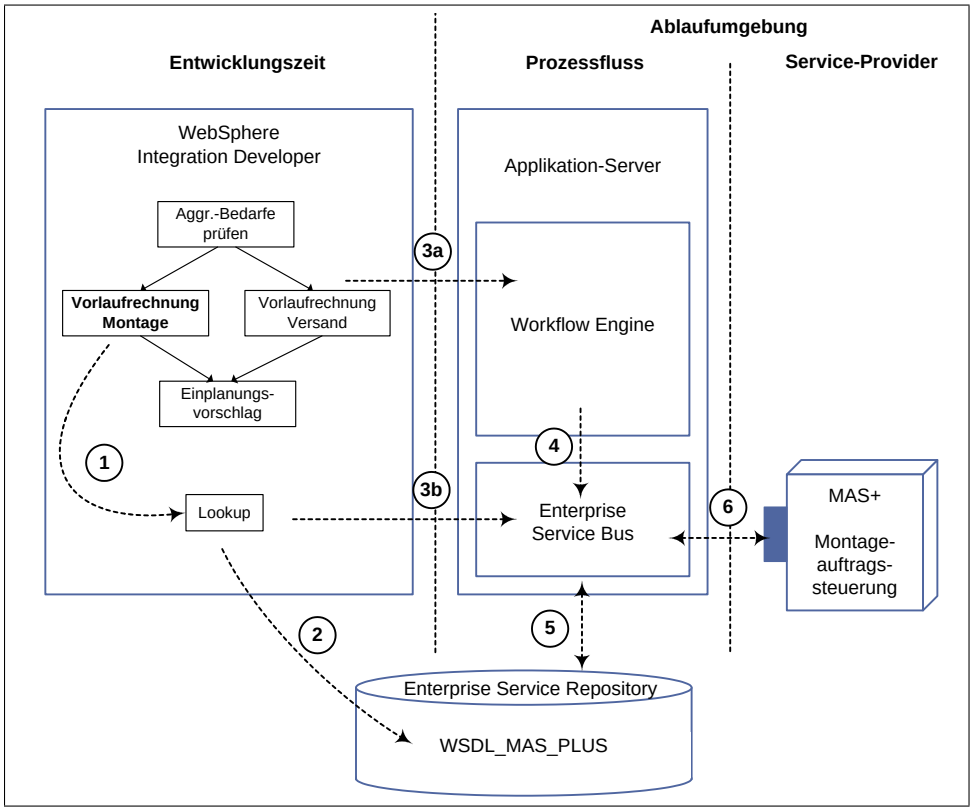

Abbildung 4.7: Quasi-statisch: Entwicklungs- und Deploymentphase

3. Ablauffähiger Code wird generiert. (a) der Prozessfluss wird in die Workflow-Engine ausgeliefert *(engl. deployed)*, (b) die *Lookup-Komponente* wird in den Enterprise Service Bus ausgeliefert *(engl. deployed)*. (Vgl. Kapitel 3.3 Schritte *VIII.* – *X.*)

4. In der Ablaufumgebung findet der Aufruf *Vorlaufrechnung Montage* des Service `WSDL_MAS_PLUS` nicht direkt, sondern über die *Lookup-Komponente* im Enterprise Service Bus statt.

5. Die *Lookup-Komponente* (im Enterprise Service Bus) ermittelt den Service-Endpunkt (siehe Kapitel 2.1.6 auf Seite 30) über das Enterprise Service Repository in der Ablaufumgebung.

6. Der Enterprise Service Bus ruft den Service `WSDL_MAS_PLUS` (bereitgestellt durch das System *MAS+*) mit dem soeben ermittelten Service-Endpunkt auf und übergibt das Geschäftsobjekt `MAS_PLUS_IN` dem Service-Provider. Der Service-Provider verarbeitet das Geschäftsobjekt und gibt das Geschäftsobjekt `MAS_PLUS_OUT` an den Enterprise Service Bus zurück. Der Enterprise Service Bus liefert das Geschäftsobjekt `MAS_PLUS_OUT` in die Workflow-Engine zurück.

Kapitel 5

Vorteile und Grenzen

Die demografische Entwicklung der Know-how-Träger bei Legacy-Systemen (etablierte, historisch gewachsene Systeme) und die festgelegte Strategie zur Unabhängigkeit (mittelfristig) eines Konzerns von seinen Lieferanten (bspw. Software-Hersteller), bedarf der Möglichkeit zur Portierung der Legacy-Systeme auf neue Plattformen. Der Lösungsansatz in Kapitel 4.2 auf Seite 79 ermöglicht eine Portierung des Systems *MAS+* auf eine neue Plattform, ohne den Prozessfluss zu beeinflussen. Dabei sind folgende Eigenschaften in der Ablaufumgebung als positiv zu bewerten:

- Protokoll-Wechsel von bspw. SOAP/HTTP auf SOAP/JMS
- Auswahl des Service-Endpunkts ohne Schnittstellenänderung

Diese Form der losen Kopplung unterstützt die Flexibilisierung der IT nur begrenzt, da bei *geplanter Ausfallzeit* des Systems *MAS+* der Prozessfluss unterbrochen wird (siehe Abbildung 5.1 auf Seite 82). Somit sind folgende Eigenschaften in der Ablaufumgebung als negativ zu bewerten:

- Die Schnittstellenbeschreibung ist lose gekoppelt, aber fix und daher nicht flexibel
- Die vorhandenen Ressourcen (äquivalente Services) in der Infrastruktur werden nicht genutzt
- Die Lookup-Komponente hat einen statischen Charakter und löst somit nicht die potentiellen Gefahrenquellen wie bspw. Unverfügbarkeit der Service-Provider

In Abbildung 5.1 auf Seite 82 wird das Verhalten in der Ablaufumgebung bei potentieller Unverfügbarkeit der Provider exemplarisch an dem Service-Provider *MAS+* illustriert.

1. Der Service-Provider *MAS+* ist zu einem bestimmten Zeitpunkt unverfügbar, womit die Kommunikation mit dem Provider nicht mehr möglich ist. Folglich ist der Service `WSDL_MAS_PLUS` ebenfalls unverfügbar und der Prozess wird beim Aufruf *Vorlaufrechnung Montage* unterbrochen.

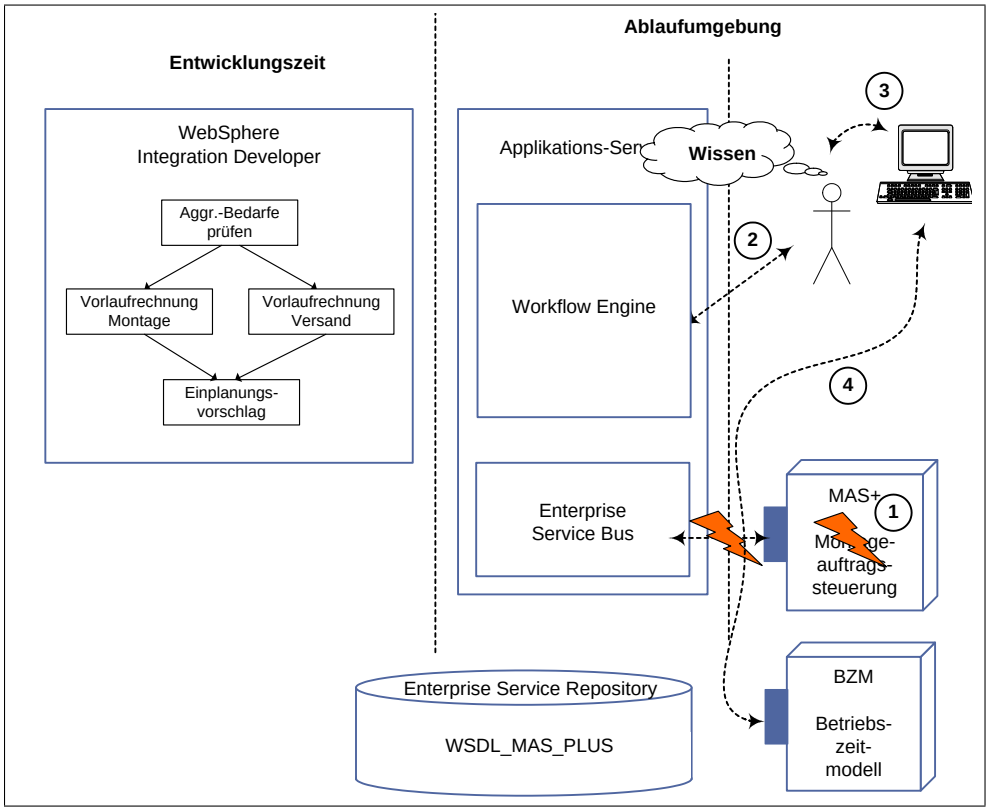

Abbildung 5.1: Implizierter Prozess-Stopp durch Unverfügbarkeit

2. Menschliches Eingreifen durch Herrn Maier (Instanz der spezifischen Personengruppe A, die das hinreichende Wissen über den Service-Provider *MAS+* und den Service-Provider *BZM* besitzt) ist notwendig um die fehlende Kommunikation mit dem Service-Provider *MAS+* zu ersetzen. Das Ergebnis (Verfügbarkeit der Personengruppe A) ist, bezogen auf das Ereignis (Service-Provider *MAS+* ist unverfügbar), nicht mehr kausal eindeutig bestimmbar (Zufall).

3. Herr Maier authentifiziert sich an einem Rechner im System *Betriebszeitmodell (BZM)*. Herr Maier nutzt sein Wissen über die vorhandenen und verfügbaren Systeme in der IT-Landschaft und besitzt die hinreichende Expertise, das System *BZM* fachlich nutzen zu können.

4. Herr Maier übergibt durch manuelle Eingabe das Geschäftsobjekt `MAS_PLUS_IN` an das System *BZM* und erhält das Geschäftsobjekt `MAS_PLUS_OUT` visuell am Monitor zurück. Nach der manuellen Übertragung des Geschäftsobjekts `MAS_PLUS_OUT` in die Workflow-Engine kann der Prozessablauf fortgesetzt werden.

In Kapitel 3 wurde der fachliche Kontext des Prozesses *Auftragsplanung Getriebe durchführen*, die Funktion *Vorlaufrechnung Montage*, die generische Top-Down-Vorgehensweise TD+ sowie die Anwendung der Top-Down-Vorgehensweise an einem quasi-statischen Modell vorgestellt. Dieser Prozess wurde redundant (getrennte Linienproduktion der Motoren, Getriebe, Achsen etc.) industrialisiert. Die Abbildung 5.1 auf Seite 82 zeigt den implizierten Prozess-Stopp bei Unverfügbarkeit des IT-Systems *MAS+* auf. Gelöst wurde die Unverfügbarkeit durch das spezifische Wissen der Personengruppe A. Wir nehmen wieder an, Herr Maier ist eine Instanz der Personengruppe A. Diese Vorgehensweise ist fehleranfällig, da Wissen über die IT-Applikationen und den Prozessablauf nur bei einer spezifischen Personengruppe A vorhanden ist. Somit ist der Lösungsansatz zur Eliminierung der Unverfügbarkeit Zufall, da das Ergebnis (Herr Maier ist verfübar), bezogen auf das Ereignis (Service-Provider *MAS+* ist unverfügbar), nicht mehr kausal eindeutig bestimmbar ist. Im Folgenden werden mögliche Szenarien benannt, die zu einer Störung des Prozessablaufs führen können:

1. Obwohl Herr Maier den Prozessablauf sehr gut kennt, ist es nicht auszuschließen, dass Übertragungsfehler von Prozess-Schritt A (in IT-Applikation I) nach Prozess-Schritt B (in IT-Applikation II) auftreten.

2. Herr Maier könnte sehr zeitnah erkranken und keiner der Personengruppe A ist kurzfristig verfügbar.

3. Die Urlaubsvertretung von Herrn Maier ist nicht mit jeder Herausforderung vertraut und handelt falsch.

4. Die Urlaubsvertretung hat nicht die hinreichende Expertise den Prozess zu steuern.

5. Herr Maier stirbt oder Herr Maier scheidet aus persönlichen bzw. unternehmerischen Gründen aus dem Unternehmen aus.

Zur Eliminierung der soeben aufgeführten potentiellen Gefahrenquellen wird der EMEO-Layer, basierend auf dem Ansatz der Service-orientierten Architektur, vorgeschlagen. Die Top-Down-Vorgehensweise TD+ in Kapitel 4.2 ist bereits heute mit den verfügbaren Mitteln realisierbar, koppelt die Prozess-Schritte an die Services *quasi-statisch* und erreicht somit nicht die komplette Eliminierung der potentiellen Gefahrenquellen. Der EMEO-Layer erweitert die Top-Down-Vorgehensweise TD+ durch die *semiautomatische* Kopplung der Prozess-Schritte an die Services, eliminiert die aufgeführten potentiellen Gefahrenquellen und erhöht die Verfügbarkeit des kompletten Prozesses *Auftragsplanung Getriebe durchführen*.

Kapitel 6

EMEO-Layer: Konzeption und Architektur

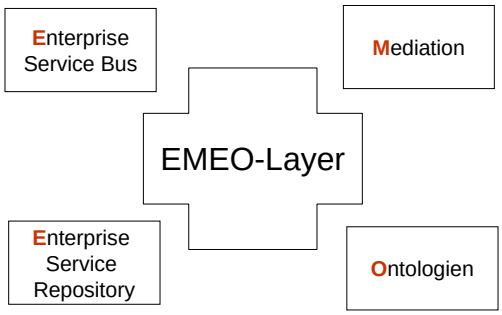

Abbildung 6.1: Technologien und Konzepte im EMEO-Layer

Die Abbildung 6.1 ist der Grafik 2.10 WSMO-Kernelemente auf Seite 44 ähnlich und definiert die logische Schicht EMEO-Layer. Der EMEO-Layer besteht aus den Ablaufumgebungen **E**nterprise Service Bus (siehe Kapitel 2.1.7 auf Seite 32), **E**nterprise Service Repository (siehe Kapitel 2.1.8 auf Seite 34), sowie dem Entwurfsmuster **M**ediation (siehe Kapitel 2.9 auf Seite 38) und der Definition von Konzepten in einer **O**ntologie (siehe Kapitel 2.3.1 auf Seite 39). Im Gegensatz zur in Kapitel 3.2.3 auf Seite 66 beschriebenen Ablaufumgebung, findet der Enterprise Service Bus und das Enterprise Service Repository hier seine Anwendung. Der Enterprise Service Bus ermittelt via Lookup-Komponente Informationen über Unverfügbarkeit der Service-Provider und ruft (bei Unverfügbarkeit) äquivalente Services in der Ablaufumgebung auf. Das Enterprise Service Repository verwaltet technische und fachliche Informationen in der Entwicklungs- und Ablaufumgebung.

Analog zu Kapitel 4.2 auf Seite 79 wird die Top-Down-Vorgehensweise TD+ aus dem Kapitel 3.3 auf Seite 68 zu Grunde gelegt. Somit beschreiben Schritt I. die Analysephase, die Schritte II. – V. die Designphase, die Schritte VI. – VIII. die Entwicklungsphase sowie die Schritte IX. und X. das Deployment. In der folgenden Darstellung nehmen wir an, die Analyse- und Designphase (Schritte I. – V.) wurden bereits

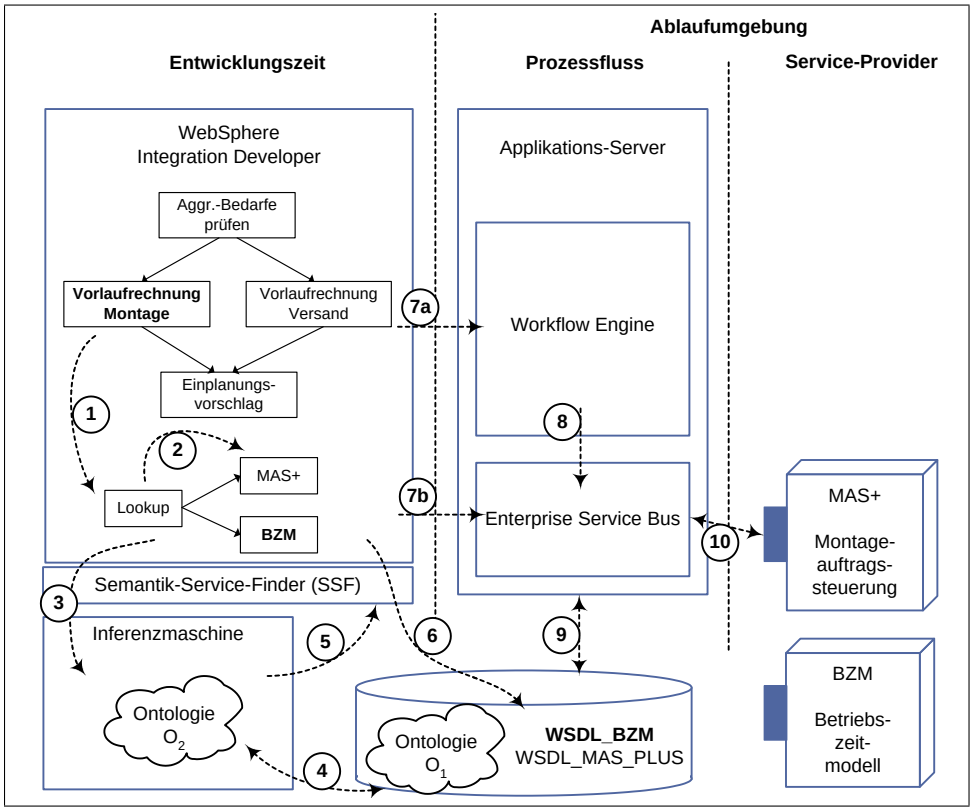

Abbildung 6.2: Semiautomatisch: Entwicklungs- und Deploymentphase

realisiert, womit alle durch Services realisierte Funktionen bereits im eEPK-Modell mit der plattformneutralen Service-Beschreibung verbunden sind. Ferner nehmen wir an, das Modell wurde bereits mit allen notwendigen technischen Informationen (Schritt VI. und partiell Schritt XII.) angereichert. Die *Lookup-Komponente*, bereits in Kapitel 4.2 eingeführt, ermöglicht eine Ermittlung des Service-Endpunkts in der Ablaufumgebung. Die folgende Darstellung erweitert das bereits in der Top-Down-Vorgehensweise TD+ (siehe Kapitel 4.2 auf Seite 79) vorgestellte Verbinden der *Lookup-Komponente* mit dem Service `WSDL_MAS_PLUS` durch die Mediation. Die dadurch verbesserte Flexibilisierung, bei Unverfügbarkeit (*Service-Provider MAS+*) in der Ablaufumgebung, schließt *Schritt VII.* ab. Die *Schritte VIII. – X.* weichen nicht von TD+ ab. Im Folgenden wird der Einbau und das Verhalten der *Mediation* in der Entwicklungs- und Ablaufumgebung beschrieben.

6.1 Entwicklung und Deployment

1. Die Verbindung zwischen Serviceaufruf *Vorlaufrechnung Montage* und Service `WSDL_MAS_PLUS` wird entfernt. Eine neue Verbindung zwischen Serviceaufruf *Vorlaufrechnung Montage* und *Lookup-Komponente* wird hergestellt.

2. Die *Lookup-Komponente* spaltet den Serviceaufruf in zwei unabhängige Pfade (P_1 und P_2). Pfad P_1 wird mit dem Service `WSDL_MAS_PLUS` verbunden.

3. Eingabe- und Ausgabeparameter sowie Vor- und Nachbedingung werden als `IOPE` *(engl. Input, Output, Precondition, Effect)* bezeichnet. Im Enterprise Service Repository (ESR) sind alle Serviceoperationen in der Form Ein- (`I`) und Ausgaben (`O`) sowie Vor- (`P`) und Nachbedingungen (`E`) mit semantischen Informationen ausgezeichnet. Das Fachwissen in der Domäne Automotive wurde bereits durch Modellierung externalisiert und in Form einer Ontologie O_1 gespeichert (erweiterte Randbedingungen durch Verfügbarkeit der Ontologie siehe Kapitel 6.6 auf Seite 94). Die IOPEs des Service `WSDL_MAS_PLUS` werden als Eingabeparameter für die Inferenzmaschine genutzt, um einen oder mehrere äquivalente Services zu finden (Architektur und Algorithmus Semantik-Service-Finder (SSF) siehe Kapitel 6.2 und 6.4 auf den Seiten 88 und 91).

4. Durch die Inferenz der Inferenzmaschine wird die abgeleitete/gefolgerte Ontologie O_2 erstellt. O_2 wird zur Auswertung im Arbeitsspeicher der Inferenzmaschine bereitgestellt.

5. Die Suche nach äquivalenten Services kann drei Ergebnisse (a, b, c) zurückliefern: (a) keinen äquivalenten Service gefunden, (b) genau einen äquivalenten Service gefunden oder (c) mehr als einen äquivalenten Service gefunden. Ergebnis (a) wird aufgrund der erweiterten Randbedingungen (Service Redundanz (siehe Kapitel 6.7 auf Seite 97)) ausgeschlossen. Ergebnis (c) verhält sich analog zu Ergebnis (b), jedoch mit der Ergänzung der Lookup-Komponente durch weitere unabhängige Pfade. Im Folgenden wird der Eintritt des Ergebnisses (b) als an Sicherheit grenzende Wahrscheinlichkeit angenommen. Durch den SSF und das Schließen der Inferenzmaschine (siehe Kapitel 6.2 auf Seite 88) wurde die Äquivalenz der beiden Services `WSDL_MAS_PLUS` und `WSDL_BZM` semiautomatisch ermittelt (Ergebnis (b) ist eingetreten).

6. Der Service `WSDL_BZM` wird mit dem zweiten unabhängigen Pfad P_2 der Lookup-Komponente verbunden. Das Abbilden *(engl. mapping)* des Serviceaufrufs *Vorlaufrechnung Montage* auf die Schnittstelle `WSDL_BZM` schließt den *Schritt VII.* komplett ab.

7. Ein ablauffähiger Code wird generiert. (a) der Prozessfluss wird in die Workflow-Engine ausgeliefert *(engl. deployed)*, (b) die Lookup-Komponente sowie die *Mediation* werden in den Enterprise Service Bus ausgeliefert *(engl. deployed)*. (Vgl. Kapitel 3.3 Schritte *VIII.* – *X.*)

8. In der Ablaufumgebung findet der Aufruf *Vorlaufrechnung Montage* des Service `WSDL_ MAS_PLUS` nicht direkt, sondern über die *Lookup-Komponente* im Enterprise Service Bus statt.

9. Die *Lookup-Komponente* (im Enterprise Service Bus) ermittelt in der Ablaufumgebung den Service-Endpunkt über das Enterprise Service Repository.

10. Der ESB ruft den Service `WSDL_MAS_PLUS` (bereitgestellt durch das System *MAS+*) mit dem soeben ermittelten Service-Endpunkt auf und übergibt das Geschäftsobjekt `MAS_PLUS_IN` dem Service-Provider. Der Service-Provider verarbeitet das Geschäftsobjekt und gibt das Geschäftsobjekt `MAS_PLUS_OUT` an den Enterprise Service Bus zurück. Der Enterprise Service Bus liefert das Geschäftsobjekt `MAS_PLUS_OUT` in die Workflow-Engine zurück.

6.2 Architektur Semantik-Service-Finder (SSF)

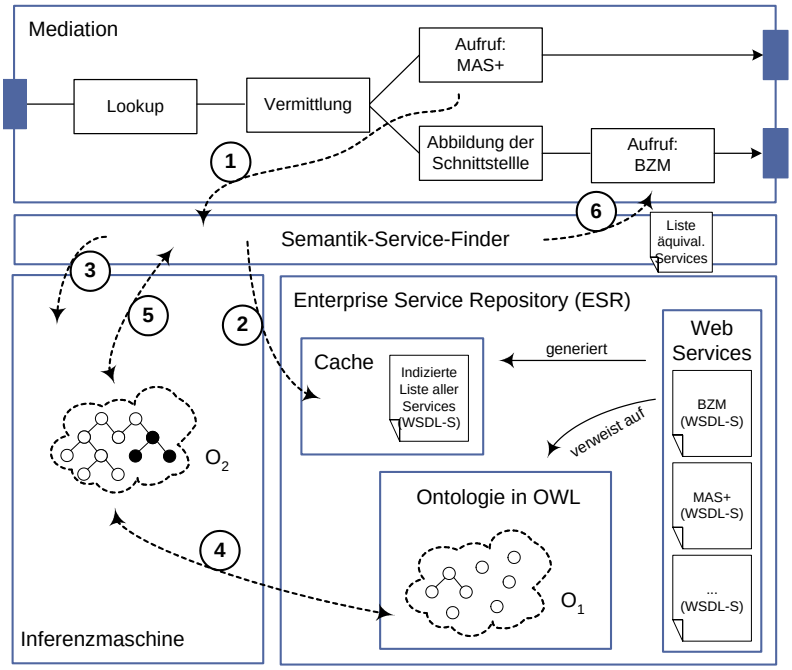

Abbildung 6.3: Architektur des Semantik-Service-Finders (SSF)

Die Abbildung 6.3 wird als Architektur des Semantik-Service-Finders (SSF) bezeichnet und konkretisiert die Schritte 4. bis 6. aus der Abbildung 6.2 auf Seite 86. Aufgezeigt werden der Einfluss auf die Mediation im ESB, die Inferenzmaschine als autonome *(engl. stand-alone)* Anwendung und die Ontologie (beschrieben in OWL) sowie den Web Services (beschrieben in WSDL-S) im ESR. Bevor Schritt 1. ausgeführt werden kann, muss durch Generierung im ESR eine indizierte Liste aller verfügbaren Services im Cache verfügbar sein.

1. Der mit semantischen Informationen angereicherte Service `WSDL_MAS_PLUS` stellt die Eingabe des SSF dar. Der SSF extrahiert die IOPEs und speichert sie in den lokalen Hash `mapMASPlusgetSollTermin`.

2. Der SSF liest die bereits durch das ESR erstellte, indizierte Liste aller verfügbaren Services in den lokalen Hash `mapServicesInESR` ein.

3. Der SSF verbindet sich via DIG-Schnittstelle (DL Implementation Group) mit der Inferenzmaschine (bspw. RACER) über eine beliebige URL (bspw. http://localhost:8080) und den Port 8080.

4. Der SSF steuert die Inferenzmaschine durch Kommunikation via DIG-Schnittstelle und löst *(engl. triggered)* das Laden der Ontologie O_1 (gespeichert im ESR) sowie die Anwendung der Inferenz auf die Ontologie O_1 aus. In Abbildung 6.3 auf Seite 88 werden Konzepte als nicht ausgefüllte Kreise dargestellt. Kanten zeigen die Relationen der Konzepte untereinander auf. Die Inferenzmaschine stellt die aus Ontologie O_1 gefolgerte Ontologie O_2 im Arbeitsspeicher zur Verfügung. Die ausgefüllten Kreise und die neu erstellten Verbindungen symbolisieren den neuen Erkenntnisgewinn, repräsentiert durch die gefolgerte Ontologie O_2.

5. Der Standard WSDL-S ermöglicht bei Input (`I`) und Output (`O`), im Gegensatz zu Precondition (`P`) und Effect (`E`), die Anreicherung mehrerer semantischer Konzepte. Somit extrahiert der SSF-Algorithmus (siehe Kapitel 6.4 auf Seite 91) Input und Output des Service `WSDL_MAS_PLUS` in weitere einzelne Fragmente, iteriert über alle Services (passend zu den extrahierten Fragmenten der `IOPE`) des lokalen Hash `mapServicesInESR` und bewertet den Abstand der einzelnen Fragmente. Die Grundlage bildet die Bewertung (bei einem passenden Fragment) der Klassenmethoden subClass() und superClass() mit 0.5 sowie der Klassenmethode equivalenceClass() mit 1.0. Nach der vollständiger Iteration über den lokalen Hash `mapServicesInESR` wird eine Liste der äquivalenten Services erstellt.

6. Der SSF übergibt die Liste äquivalenter Services, sortiert nach dem Abstand zwischen den einzelnen Fragmenten, zur Verifizierung und semiautomatischer Auswahl an den Entwickler. Der Entwickler wählt nach Rücksprache (QoS, Security etc.) mit dem Service-Portfolio-Management (siehe Kapitel 2.1.9 auf Seite 35) einen äquivalenten Service aus und bildet die Schnittstelle auf die Funktion *Vorlaufrechnung Montage* ab.

Der SSF nutzt die Inferenzmaschine RACER (siehe Punkt 4.), um aus der Ontologie O_1 die gefolgerte Ontologie O_2 im Arbeitsspeicher zur Verfügung zu stellen. Dabei wird von der Randbedingung ausgegangen, dass das vorgestellte Gedankenexperiment (siehe Kapitel 6.6 auf Seite 94) für jedes Konzept und jede Relation in der Ontologie angewandt wurde. Zusätzlich wird davon ausgegangen, dass nicht jede durch Inferenz ermittelte Erkenntnis notwendigerweise vom Fachbereich anerkannt wird (Diskussion mit Jens Lehmann und Sebastian Hellmann am 31.10.2007 via Telefonkonferenz). Die Ontologie O_1 liegt für den Fachbereich in logischer, die Ontologie O_2 liegt für den Algorithmus SFF in auswertbarer Form vor. Die Inferenz erkennt keine Abhängigkeiten zu WSDL-S, womit die Anwendung des Algorithmus SSF zur Bestimmung äquivalenter Services (Kandidaten) notwendig ist.

6.3 Anreicherung semantischer Informationen

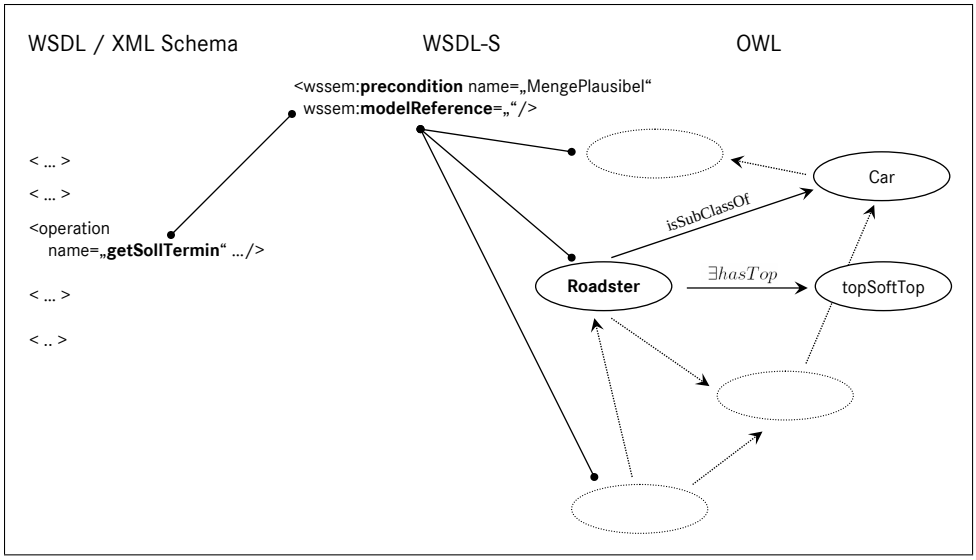

Abbildung 6.4: Art und Weise der Anreicherung semantischer Informationen

Die Architektur des beschriebenen Algorithmus SSF (siehe Kapitel 6.2 auf Seite 88) setzt eine bestimmte Art und Weise der Anreicherung semantischer Informationen voraus, die im Folgenden beschrieben wird. Die Abbildung 6.4 illustriert drei Bereiche: WSDL, WSDL-S und OWL. Im linken Teil wird ein kleiner Ausschnitt (Serviceoperation) der Schnittstellenbeschreibung des Service `MASPlus` dargestellt. Der mittlere Teil stellt das Element `<wssem:precondition>` sowie das erweiternde Attribut `<wssem:modelReference>` des Standards WSDL-S dar. Die Linien mit den ausgefüllten Punkten demonstrieren die technische Verbindung zwischen Elementen aus dem Standard WSDL, den Elementen aus dem Standard WSDL-S und den Konzepten aus dem Standard OWL. Im rechten Teil wird exemplarisch eine Ontologie in OWL dargestellt. Hier illustrieren die benannten Ellipsen definierte Konzepte, wie bspw. `Roadster`, `Car` und `topSoftTop`, und Pfeile (gerichtete Kanten) die Relationen wie bspw. $\exists hasTop$ und `isSubClassOf`. Die gestrichelten unbenannten Ellipsen sollen weitere Konzepte, die gestrichelten Pfeile weitere Relationen in einer Ontologie darstellen und die Ausdrucksmächtigkeit einer Ontologie, im Gegensatz zu einer Taxonomie, verdeutlichen. In allen drei Teilen wird die Aussage der Grafik durch die Auszeichnung in halbfetter Schrift abgebildet: Die Operation `getSollTermin` wird über das Element `<wssem:precondition>`, das erweiternde Attribut `<wssem:modelReference>`, das Konzept `Roadster` und *weiteren Konzepten* mit semantischen Informationen angereichert. Somit ist in dieser Arbeit das Verbinden des Elementes `<wssem:modelReference>` mit mehreren Konzepten in einer Ontologie (basierend auf OWL) möglich (vlg. Semantic Annotations for WSDL (SAWSDL) in Kapitel 2.3.5 auf Seite 46).

6.4 Semantik-Service-Finder (SSF-Algorithmus)

Durch den im Folgenden präsentierten Pseudocode wird, als ein Konglomerat aus der natürlichen Sprache und einer höheren Programmiersprache, der Semantik-Service-Finder-Algorithmus (SSF-Algorithmus) beschrieben. In Kapitel 7.4 auf Seite 116 werden Code-Fragmente der Implementierung und im Anhang L auf Seite 189 der komplette Quellcode der Implementierung gelistet.

Pseudocode 1 Hauptprogramm SSF

Require: IOPE of a WSDL-Service-Operation-1
Ensure: List of equivalent WSDL-Service-Operations ordered by ranking (HashMap)
1: extract IOPE from input and map structured activity to HashMap-1
2: connect to Reasoner and doInference
3: init ordered List (HashMap)
4: **if** get First WSDL-Service-Operation-2 from ESR **then**
5: **repeat**
6: extract IOPE from WSDL-Service-Operation-2
7: map structured activity to HashMap-2
8: **for** (IOPE) **do**
9: rankService(HashMap-1, HashMap-2, IOPE) add to List
10: **end for**
11: **until** has Next WSDL-Service-Operation-2 from ESR
12: **end if**
13: return List (HashMap)

Das Hauptprogramm (Pseudocode 1) erwartet semantische Informationen (Konzepte), in der Form: Eingabe (`I`), Ausgabe (`O`), Vor- (`P`) und Nachbedingung (`E`) als Eingabe und gibt eine sortierte Liste möglicher, äquivalenter Serviceoperationen (Kandidaten) zurück. Zeile `01` extrahiert die Eingabe in den `HashMap-1`. Zeile `02` verbindet den SSF mit einer Inferenzmaschine und führt die Inferenz aus (unter der Voraussetzung, dass die Inferenzmaschine in die Infrastruktur integriert ist und Zugriff auf die Ontologie besitzt). Zeile `03` initialisiert eine lokale Liste zur Speicherung der Rückgabe (`HashMap`). Zeile `04` liest die erste Serviceoperation aus dem Enterprise Service Repository (unter der Voraussetzung, dass der SSF in die Infrastruktur integriert ist und Zugriff auf das ESR besitzt). In Zeile `05` beginnt der Schleifenrumpf mit der logischen Operation als Abbruchbedingung am Ende (`11`). Da in Zeile `04` das Einlesen mit einem logischen IF verbunden ist, wird der Schleifenkörper ab Zeile `05` nur bei einer positiven Rückgabe durchlaufen. Zeile `06` extrahiert die Eingabe aus dem ESR in den `HashMap-2`. Zeile `07` strukturiert `HashMap-2` in der Form SERVICE, OPERATION, I, O, P und E. Die Zeilen `08`, `09` und `10` beschreiben eine Zählschleife, die pro Serviceoperation viermal (IOPE) durchlaufen wird und bei jedem Durchlauf das Unterprogramm `rankService(HashMap-1, HashMap-2, IOPE)` aufruft. WSDL-S ermöglicht die Anreicherung der Serviceoperationen mit semantischen Informationen, dabei erweitert der EMEO-Layer das Verbinden der Eingabe (`I`), Ausgabe (`O`), Vor- (`P`) und Nachbedingung (`E`) mit mehreren unterschiedlichen Konzepten. Ein zugeordnetes Konzept wird im Folgenden als Fragment bezeichnet. Bei jedem Aufruf des Unterprogramms `rankService(HashMap-1, HashMap-2,`

IOPE) werden alle Fragmente (IOPE) übergeben. Die Rückgabe des Unterprogramms wird in der lokalen Liste (HashMap) gespeichert. Zeile 11 ist die logische Abbruchbedingung für den Schleifenkörper und liest solange die Serviceoperationen aus dem ESR, bis keine weiteren mehr vorhanden sind. Zeile 12 beendet das logische IF aus Zeile 04. Zeile 13 gibt eine sortierte Liste möglicher äquivalenter Serviceoperationen (Kandidaten) zurück.

Pseudocode 2 Unterprogramm rankService(HashMap-1, HashMap-2, IOPE)

```
Require: HashMap-1, HashMap-2, IOPE
Ensure: partial List of equivalent WSDL-Service-Operations ordered by ranking
 1: extract IOPE from HashMap-1 and map structured activity to Fragment-1
 2: extract IOPE from HashMap-2 and map structured activity to Fragment-2
 3: init partial List pList (HashMap)
 4: for (i=0; i < Fragment-1.length(); i++) do
 5:    for (j=0; j < Fragment-2.length(); j++) do
 6:       if ( Fragment-1[i] == Fragment-2[j] ) then
 7:          add IOPE + Fragment-1[i] + 1.0 to pList
 8:          calc percent IOPE and to pList
 9:          return pList
10:       else if ( Fragment-1[i] isSuperClassOf(Fragment-2[j]) ) then
11:          add IOPE + Fragment-1[i] + 0.5 to pList
12:          calc percent IOPE and to pList
13:          return pList
14:       else if ( Fragment-1[i] isSubClassOf(Fragment-2[j]) ) then
15:          add IOPE + Fragment-1[i] + 0.5 to pList
16:          calc percent IOPE and to pList
17:          return pList
18:       else
19:          return empty pList
20:       end if
21:    end for
22: end for
```

Das Unterprogramm (Pseudocode 2) wird aus Zeile 09 im Hauptprogramm (Pseudocode 1) auf Seite 91 aufgerufen, erwartet HashMap-1, HaspMap-2 sowie IOPE und gibt die ermittelte, sortierte Liste einer Serviceoperation (Kandidaten) zurück. Die Zeilen 01 und 02 strukturieren HashMap-1 und HashMap-2 nach ihren Fragmenten. Zeile 03 initialisiert eine lokale Liste zur Speicherung der Rückgabe (HashMap). Die Zeilen 04-22 beschreiben zwei Zählschleifen sowie drei zu unterscheidende Bedingungen (IF). Die erste Zählschleife (Zeile 04) iteriert über jedes Fragment, bestehend aus der bekannten Serviceoperation. Die zweite Zählschleife (Zeile 05) iteriert über jedes Fragment, bestehend aus der zu bewertenden Serviceoperation (Kandidat). Die erste zu unterscheidende Bedingung (Zeilen 06-09) prüft auf Äquivalenz der Fragmente (Zeile 06), bewertet bei Übereinstimmung mit 1.0 (Zeile 07), legt die Bewertung in der lokalen Liste HashMap ab (Zeile 08) und gibt die lokale Liste an das Hauptprogramm zurück (Zeile 09). Die beiden weiteren Unterscheidungen sind in der Prüfung auf Ober- (Zeile 10) bzw. Unterklasse (Zeile 14) sowie in der Bewertung mit 0.5 bei Übereinstimmung (Zeile 11 bzw. 15) dokumentiert. In Zeile 19 wird eine leere Liste bei fehlender Übereinstimmung zurückgeben. Die Zeilen 21 und 22 schließen die beiden Zählschleifen syntaktisch ab.

6.5 Verhalten bei Unverfügbarkeit

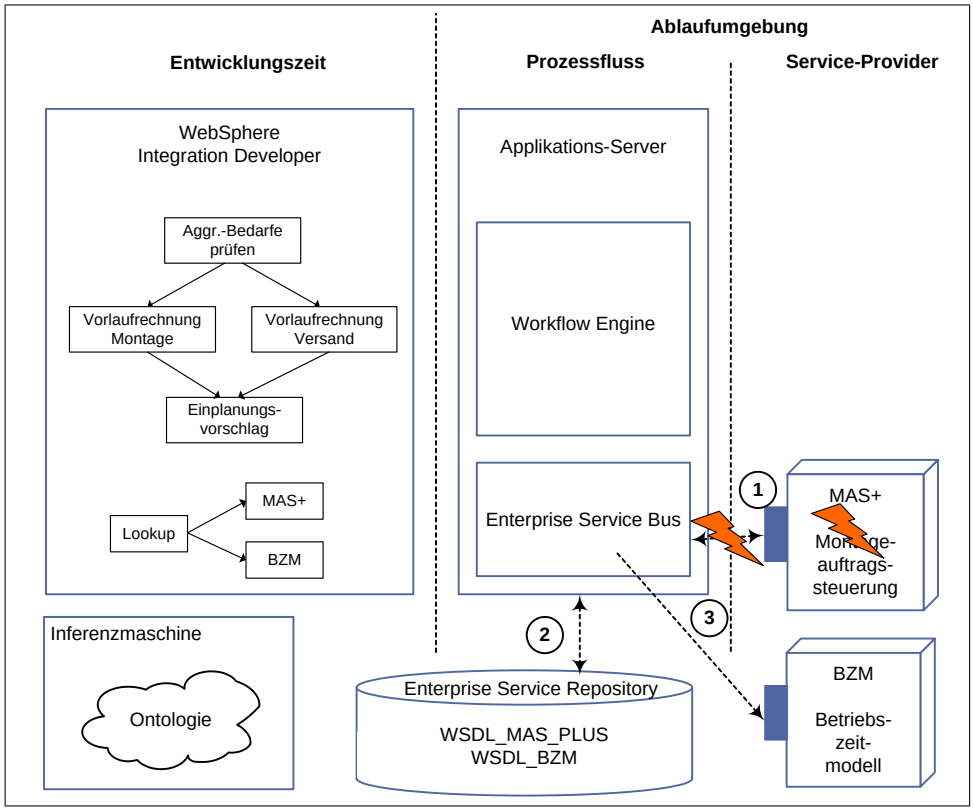

Abbildung 6.5: Mediator verhindert implizierten Prozess-Stop

Im Folgenden wird das verbesserte Verhalten (vgl. Kapitel 5 auf Seite 81) in der Ablaufumgebung bei Unverfügbarkeit des *Service-Provider-1 MAS+* illustriert.

1. Der *Service-Provider-1 MAS+* ist zu einem bestimmten Zeitpunkt nicht verfügbar, wodurch die Kommunikation mit dem Provider-1 nicht mehr möglich ist. Somit wird der Prozess beim Prozess-Schritt *Vorlaufrechnung Montage*, für den Anwender nicht wahrnehmbar unterbrochen.

2. Das in Abbildung 5.1 notwendige menschliche Eingreifen der Personengruppe A ist nicht mehr nötig, da das Enterprise Service Repository über den zurzeit gewarteten *Service-Provider-1 MAS+* informiert wurde. Die *Lookup-Komponente* liefert die Information der Unverfügbarkeit in der Ablaufumgebung an den Enterprise Service Bus.

3. Der Enterprise Service Bus ruft den äquivalenten *Service-Provider-2 BZM* in der Ablaufumgebung auf und übergibt das Geschäftsobjekt `BZM_IN` an den Provider-2. Dieser gibt das Geschäftsobjekt `BZM_OUT` zurück und der Prozess wird automatisch fortgesetzt.

6.6 Domänenspezifische Ontologie

In Kapitel 6.4 auf Seite 91 wird ein Baustein des EMEO-Layers, der Algorithmus SSF, mit dem Ziel der Bewertung äquivalenter Konzepte vorgestellt. Das folgende Gedankenexperiment soll zu ersten Erkenntnissen in der Modellierung des minimal anzunehmenden, deklarativen Wissens der Personengruppe A führen. Um in diesem Kapitel Konzepte visuell von Relationen unterscheiden zu können, beginnen die Konzepte, im Gegensatz zu den Relationen, mit einem Hash (`#`). Die Konzepte `#leftFrontDoor`, `#rightFrontDoor`, `#leftSeat`, `#rightSeat`, `#topSoftTop` und `#Car` sowie die Relationen `hasDoorway` (Datentyp-Eigenschaft) und `hasSeat` (Datentyp-Eigenschaft) werden als gegeben vorausgesetzt. Zur Durchführung des Gedankenexperiments nehmen wir an: die Vorbedingung (`P`) der Serviceoperation `ermittleSollTermin` des Service `BZM` wurde mit dem Konzept `#Roadster` ausgezeichnet. Im Gegensatz dazu wurde die Vorbedingung (`P`) der Serviceoperation `getSollTermin` des Service `MASPlus` mit dem Konzept `#SLK` ausgezeichnet. Jeder `#SLK` ist ein `#Roadster`, aber nicht jeder `#Roadster` ein `#SLK`, somit folgt `#SLK` ist ein Teilkonzept von `#Roadster`.

Damit die Inferenzmaschine das Konzept `#SLK` als Teilkonzept des Konzeptes `#Roadster` durch Inferenz ermitteln kann, müssen beide Konzepte definiert werden. Im Folgenden werden die beiden Konzepte `#Roadster` und `#SLK` durch Anwendung der iterativen Methode On-To-Knowledge (OTKM) [154] definiert sowie durch Inferenz verifiziert. OTKM definiert einen Meta-Prozess mit den Phasen Identifizierung des Problems bzw. Bestimmung des Ziels, Projektstart *(engl. kickoff)*, Erfassung und Codierung der Ontologie, Evaluierung und Evolution (vgl. Grafik in G auf Seite 165). Dies geschieht mit dem Ziel, das minimal angenommene, spezifische Wissen der Personengruppe A in Form von Konzepten, zentral in einer Ontologie abzulegen und durch die iterative Methode On-To-Knowledge, in Zusammenarbeit mit dem Fachbereich, zu validieren (siehe Kapitel 7.2 auf Seite 105).

X und Y sind Personengruppen, die eine hinreichende Expertise haben, um fachliches Wissen über einen Roadster bzw. einen SLK beschreiben zu können. Die erste Iteration mit der Personengruppe X ergab die Definition: *„Ein Roadster ist ein Auto mit genau zwei Türen und genau zwei Sitzen“*, und wird somit als notwendiges und hinreichendes Konzept wie folgt definiert:

$$Roadster \equiv Car \ \sqcap (= 2\ hasDoorway) \ \sqcap (= 2\ hasSeat) \tag{6.1}$$

Die Methode OTKM schlägt die Überprüfung der Definition 6.1 durch eine weitere Personengruppe Y vor. Bei der Überprüfung der Definition qualifizierten sich der `#SLK`, der `#Crossfire` und der `#SLR` als Teilkonzepte des Konzeptes Roadster. `#Crossfire` und `#SLR` existieren in beiden Ausprägungen Roadster und Coupe, womit die (neuen) Konzepte `#Crossfire_Roadster` und `#SLR_Roadster` sowie `#Crossfire_Coupe` und `#SLR_Coupe` eingeführt wurden. Als Alleinstellungsmerk-

mal für einen Roadster wurde die Relation `hasTop` (Konzepteigenschaft) mit dem Wert `topSoftTop` identifiziert und definiert. Folglich qualifizierten sich die Konzepte `#Crossfire_Coupe` und `#SLR_Coupe` nicht als Roadster. Die zweite Iteration der Personengruppe Y ergab die Definition: *„Ein Roadster ist ein Auto mit genau zwei Türen und genau zwei Sitzen und genau einem Faltdach“*. Die zweite Iteration erweitert somit die Definition 6.1 um einen Existenzausdruck ($\exists$) und wird als notwendiges und hinreichendes Konzept wie folgt definiert:

$$\begin{aligned} Roadster \equiv Car \ &\sqcap (= 2\ hasDoorway) \ \sqcap (= 2\ hasSeat) \\ &\sqcap \exists hasTop.topSoftTop \end{aligned} \tag{6.2}$$

Die iterative Methode OTKM schlägt die Verifizierung der Definition 6.2 durch eine exemplarische Implementierung vor. Dabei ergab sich für einen SLK die Beschreibung: *„Ein SLK ist ein Auto mit genau einer rechten Tür, genau einer linken Tür, genau einem rechten Sitz, genau einem linken Sitz und genau einem Faltdach“*, und erfüllt somit als Konzept die folgende Bedingung:

$$\begin{aligned} SLK \sqsubseteq \ &= 1\ hasDoorway.leftFrontDoor \sqcap \\ &= 1\ hasDoorway.rightFrontDoor \sqcap \\ &= 1\ hasSeat.leftSeat \sqcap \ = 1\ hasSeat.rightSeat \sqcap \\ &\exists hasTop.topSoftTop \ \sqcap \ Car \end{aligned} \tag{6.3}$$

Die Bedingung $SLK \sqsubseteq Roadster$, genau dann wenn, alle Instanzen von SLK Instanzen von *Roadster* sind. Nach der Inferenz wurde die Zuordnung des Konzeptes `#SLK` als Teilkonzept des Konzeptes `#Roadster` durch die Inferenzmaschine erwartet. Obwohl die rechte Tür und die linke Tür sowie der rechte Sitz und der linke Sitz disjunkt sind, wird das Konzept `#SLK` nicht als Unterklasse des Konzeptes `#Roadster` erkannt. Im Folgenden wird die Bedingung 6.3 in die Logik erster Stufe transformiert, wobei für $(\exists y)((hasDoorway(x,y) \wedge leftFrontDoor(y)) \wedge (\forall y_1)(\forall y_2)(hasDoorway(x,y_1) \wedge hasDoorway(x,y_2) \Rightarrow (y_1 = y_2)))$ (mindestens einen $\wedge$ höchstens einen $\Leftrightarrow$ genau einen) in der Definition 6.4 die Kurzschreibweise $(\exists!y)(hasDoorway(x,y) \wedge leftFrontDoor(y))$ verwendet wird und die Definition 6.4 zu lesen ist als: Wenn ein Objekt x ein SLK ist, dann hat x genau eine rechte Tür und genau eine eine linke Tür, hat genau einen linken Sitz und genau einen rechten Sitz, hat x ein Dach (Faltdach), und für alle y, die Dach von x sind, ist y ein Faltdach, und ist x ein Auto.

$$
\begin{aligned}
SLK(x) \Rightarrow\ & ((\exists!y)(hasDoorway(x,y) \wedge leftFrontDoor(y))) \wedge \\
& ((\exists!y)(hasDoorway(x,y) \wedge rightFrontDoor(y))) \wedge \\
& ((\exists!y)(hasSeat(x,y) \wedge leftSeat(y))) \wedge \\
& ((\exists!y)(hasSeat(x,y) \wedge rightSeat(y))) \wedge \\
& ((\exists y)(hasTop(x,y) \wedge topSoftTop(y))) \wedge \\
& (Car(x))
\end{aligned} \tag{6.4}
$$

Nach der logischen Analyse fällt auf, dass die Spezifikation des Konzepts `#SLK` (Bedingung 6.3) um weitere Türen wie bspw. $rightBackDoor$ (linke Hintertür) oder $leftBackDoor$ (rechte Hintertür) erweitert werden kann. Um diese (ungewünschte) Erweiterung auszuschließen, wird die Spezifikation des Konzepts `#SLK` (Bedingung 6.3) wie folgt durch zwei abschließende Axiome *(engl. closure axioms)* erweitert:

$$
\begin{aligned}
SLK \sqsubseteq\ & = 1\ hasDoorway.leftFrontDoor \sqcap \\
& = 1\ hasDoorway.rightFrontDoor \sqcap \\
& = 1\ hasSeat.leftSeat \sqcap\ = 1\ hasSeat.rightSeat \sqcap \\
& \forall hasDoorway\ (leftFrontDoor \sqcup rightFrontDoor) \sqcap \\
& \forall hasSeat\ (leftSeat \sqcup rightSeat) \sqcap \\
& \exists hasTop.topSoftTop\ \sqcap\ Car
\end{aligned} \tag{6.5}
$$

Nun kann durch logische Analyse leicht gezeigt werden, dass die Spezifikation des Konzepts `#SLK` (rechte Seite der Definition 6.5) als Teilkonzept der Spezifikation des Konzepts `#Roadster` (rechte Seite der Definition 6.2) impliziert. Hierbei setzen wir voraus, dass in der TBox das Axiom $\bot = leftFrontDoor \wedge rightFrontDoor$ (Disjunktion) enthalten ist. Durch erneute Inferenz wurde die Zuordnung des Konzeptes `#SLK` als Teilkonzept des Konzeptes `#Roadster` durch die Inferenzmaschine erkannt und somit die logische Analyse verifiziert:

$$SLK \sqsubseteq Roadster \tag{6.6}$$

Zum besseren Verständnis der Fachlichkeit wird im Anhang H.1 auf Seite 169 ein SLK (Roadster), ein SLR (Coupe) und zwei Crossfire (Roadster und Coupe) illustriert. Die Erkenntnisse aus dem soeben dargestellten Gedankenexperiment sind

- Einführung notwendiger Iterationen bei der Wissensmodellierung
- Definition erwarteter Ergebnisse nach der Anwendung der Inferenz („Testfälle“)
- Notwendiges Handhaben der Herausforderung der Offenen-Welt-Annahme

6.7 Verfügbarkeit der Service-Provider durch Redundanz

Der Prozess *Auftragsplanung Getriebe durchführen* (siehe Abbildung 4.3) wurde, aus historischen und organisatorischen Gründen (getrennte Linienproduktion der Motoren, Getriebe, Achsen etc.), mehrfach in unterschiedliche IT-Systeme auf verschiedenen strategischen Plattformen implementiert. Die verschiedenen Plattformen ergeben sich dabei durch das Ziel jeder Organisationseinheit, homogene Systeme zu betreiben. Jedes dieser IT-Systeme muss die aggregatsbezogene Kalkulation durchführen, somit nehmen wir eine Redundanz der Services im Faktor größer zwei an. Dies bedeut, dass jeder Service mindestens zwei Mal in der Infrastruktur vorhanden ist. Somit werden die Schnittstellenbeschreibungen *(engl. Interface)* der Services des *Service-Provider-1 MAS+* sowie des *Service-Provider-2 BZM* plattformneutral durch `WSDL`-Dateien im Enterprise Service Repository zur Verfügung gestellt.

Im Bereich Informationstechnologie Management (ITM) Mercedes-Benz Cars wurden ca. 1.500 Applikationen und ca. 2.500 Instanzen der Applikationen quantifiziert [106]. Die Anzahl der möglichen Services in der Infrastruktur bei ITM Mercedes-Benz Cars ist nicht zu ermitteln. Das Unternehmen Credit Suisse [57] beziffert die Anzahl ihrer Services auf über 50.000. Diese Angabe wurde von Vertretern der Software AG und telefonisch von Vertretern der Credit Suisse bestätigt. Die Anzahl an Services beziffert alle verfügbaren Versionen [119] [25]. Somit nehmen wir ebenfalls eine Anzahl von 50.000 Services an. Diese Services sind plattformneutral beschrieben und logisch zentral in einem Enterprise Service Repository (ESR) (siehe Kapitel 2.1.8 auf Seite 34) abgelegt. Alle IT-Systeme, die an der Auftragsabwicklung (siehe Abbildung 4.4 auf Seite 76 und Abbildung 4.5 auf Seite 77) beteiligt sind, wurden bereits im ESR dokumentiert.

Kapitel 7

EMEO-Layer: Validierung

Der vorgeschlagene Lösungsansatz EMEO-Layer wurde in Kapitel 6 darstellt. Zur Validierung wird das notwendige exemplarische Szenario (bestehend aus zwei Teilelementen) sowie ein lauffähiger Prototyp (bestehend aus drei Teilelementen) vorgestellt. Die beiden Teilelemente sind notwendige Voraussetzungen für die Erstellung des lauffähigen Prototypen und werden im Folgenden kurz beschrieben:

1. Teilelement I: illustriert exemplarisch an den beiden Services `MASPlus` und `BZM` die Anreicherung der Serviceoperationen durch semantische Auszeichnungen (siehe Kapitel 7.1 auf Seite 100).

2. Teilelement II: illustriert einen Ansatz zur domänenspezifischen Ontologie mit dem Ziel der Externalisierung des Know-hows der Personengruppe A (siehe Kapitel 7.2 auf Seite 105).

Auf diesen beiden Teilelementen aufbauend, wird der lauffähige Prototyp, bestehend aus drei Teilelementen, im Folgenden kurz beschrieben:

1. Teilelement I: der produktneutral entwickelte Mediator demonstriert die lose Kopplung von Objekten in Java (siehe Kapitel 7.3 auf Seite 112).

2. Teilelement II: der Semantik-Service-Finder demonstriert die semantische semiautomatische Identifizierung äquivalenter Services zur Entwicklungszeit. Dabei kommen das Open-Source-Werkzeug Protégé und die Inferenzmaschine RACER zum Einsatz. (siehe Kapitel 7.4 auf Seite 116).

3. Teilelement III: die exemplarische Implementierung in den Produkten WebSphere Integration Developer, WebSphere Enterprise Service Bus sowie WebSphere Process Server demonstrieren die Lookup-Komponente und den Mediator im ESB in der Entwicklungs- bzw. Ablaufumgebung (siehe Kapitel 7.5 auf Seite 124).

Abschließend wird in Kapitel 7.6 auf Seite 130 die neue, abgeleitete Vorgehensweise vorgestellt, die als TD++ bezeichnet wird. Hierzu wird in Kapitel 7.7 auf Seite

131, basierend auf echten Werten, der rechnerischen Nachweis zur Minimierung der Unverfügbarkeit dokumentiert.

7.1 Anreicherung semantischer Informationen

Die Funktion *Vorlaufrechnung Montage* wird durch die Operationen `getSollTermin` bzw. `ermittleSollTermin` realisiert und durch die Services `MAS_PLUS` bzw. `BZM` bereitgestellt. Ziel ist es, nach der Anreicherung mit teilweise unterschiedlichen semantischen Informationen, die Distanz zwischen den beiden Serviceoperationen zu ermitteln (Inferenz in Kombination mit einem Algorithmus). Die Operation `getSollTermin` erwartet das Geschäftsobjekt `MAS_PLUS_IN` als Eingabeparameter und gibt das Geschäftsobjekt `MAS_PLUS_OUT` als Ausgabeparameter zurück. Die Operation `ermittleSollTermin` erwartet das Geschäftsobjekt `BZM_IN` als Eingabeparameter und gibt das Geschäftsobjekt `BZM_OUT` als Ausgabeparameter zurück. Die komplette Beschreibung der Services in WSDL-S wird im Anhang P auf Seite 219 gelistet.

```
01 xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
02 xmlns:DAI="http://www.daimler.com/xmlns/XMLSchema/DAI"
03 xmlns:AutomotiveDAI="http://www.daimler.com/ontologies/DAI.owl#">
```

Die abgebildeten Zeilen `01-03` sind in beiden WSDL-Dateien identisch und definieren durch die Referenzierung auf eine URI die `Namespaces` für den WSDL-S Standard (`01`), das daimlerspezifische Datenschema (`02`) und die daimlerspezifische Ontologie (`03`).

7.1.1 MAS_PLUS

XML Schema Definition (URI-Liste)

```
01 <resource-lists xmlns="MASPlus-P-uri-list">
02    <list>
03       <entry uri="AutomotiveDAI#MengePlausibel" />
04       <entry uri="AutomotiveDAI#Gear" />
05       <entry uri="AutomotiveDAI#SLK" />
06    </list>
07 </resource-lists>
```

Der Vorbedingung (`P`) wurden drei Konzepte in Form einer URI-Liste in der Ontologie zugewiesen. In Zeile `01` wird ein eindeutiger Bezeichner der URI-Liste definiert. In den Zeilen `02` und `06` wird der Beginn bzw. das Ende der Liste markiert. Die Zeilen `03`, `04` und `05` definieren die URI-Liste, bestehend aus den Konzepten `#MengePlausibel`, `#Gear` und `#SLK`. Die Zeile `07` schließt die Definition der URI-Liste ab. Die URI-Liste findet ihre Anwendung in Kapitel Serviceoperation MAS_PLUS auf Seite 102 in Zeile `24`.

Geschäftsobjekt MAS_PLUS_IN

```
04 <xsd:complexType name="MAS_PLUS_IN">
05   <xsd:sequence>
06      <xsd:element name="Sachnummer" nillable="false" type="xsd:string"
07       wssem:modelReference="AutomotiveDAI#DAISachnummer"/>
08   </xsd:sequence>
09   <xsd:sequence><xsd:element name="Menge" nillable="false"
10       type="xsd:integer" wssem:modelReference="AutomotiveDAI#Menge"/>
11   </xsd:sequence>
12   <xsd:sequence><xsd:element name="Liefertermin" nillable="false"
13       type="xsd:date" wssem:modelReference=
14                          "AutomotiveDAI#DAILiefertermin"/>
15   </xsd:sequence>
16 </xsd:complexType>
```

Das Geschäftsobjekt `MAS_PLUS_IN` wird in Form eines `<complexType>` realisiert. In Zeile `06, 09` und `13` werden alle benötigten Variablen definiert. Der fachliche Datentyp *Sachnummer* in Zeile `06` wird um `Not Null` erweitert und als primitiver technischer Datentyp `xsd:string` definiert. Der fachliche Datentyp *Menge* in Zeile `09` wird ebenfalls um `Not Null` erweitert und ebenfalls als primitiver technischer Datentyp `xsd:integer` definiert. Zusätzlich wird in Zeile `07` auf das Konzept `#DAISachnummer` referenziert und somit semantisch eindeutig definiert. Analog zur Erweiterung der *Sachnummer* verhält es sich zu *Menge* in Zeile `09` und `10` sowie zu *Liefertermin* in Zeile `13` und `14`.

Geschäftsobjekt MAS_PLUS_OUT

```
17 <xsd:simpleType name="MAS_PLUS_OUT">
18    <xsd:restriction base="xsd:date"
19     wssem:modelReference="AutomotiveDAI#DAISollTermin">
20   </xsd:restriction>
21 </xsd:simpleType>
```

Das Geschäftsobjekt `MAS_PLUS_OUT` wird in Form eines `<simpleType>` realisiert. Der fachliche Datentyp wird in Zeile `18` als technischer Datentyp `datetime` definiert. Zusätzlich wird in Zeile `19` auf das Konzept `#DAISollTermin` referenziert und somit semantisch eindeutig definiert.

Serviceoperation MAS_PLUS

```
22 <operation name="getSollTermin" pattern=wsdl:in-out
23   <wssem:precondition name="MengePlausibel"
24    wssem:modelReference="DAI:MASPlus-P-uri-list">
25   <wssem:effect name="SollTerminErmittelt"
26    wssem:modelReference="AutomotiveDAI#DAISollTerminVerfuegbar"/>
27 </operation>
```

Die Operation `getSollTermin` wird mit den Bezeichnern `MengePlausibel` als Vorbedingung und `SollTerminErmittelt` als Nachbedingung ausgezeichnet. In Zeile `24` wird die Vorbedingung mit der URI-Liste `MASPlus-P-uri-list` (siehe Kapitel 7.1.1 auf Seite 100) und in Zeile `26` die Nachbedingung mit dem Konzept `#DAISollTerminVerfuegbar` verbunden.

7.1.2 BZM

XML Schema Definition (URI-Liste)

```
01 <resource-lists xmlns="BZM-P-uri-list">
02    <list>
03       <entry uri="AutomotiveDAI#AmountPlausible" />
04       <entry uri="AutomotiveDAI#Aggregate" />
05       <entry uri="AutomotiveDAI#Roadster" />
06    </list>
07 </resource-lists>
```

Der Vorbedingung (`P`) wurden drei Konzepte, in Form einer URI-Liste, aus der Ontologie zugewiesen. In Zeile `01` wird ein eindeutiger Bezeichner der URI-Liste definiert. In den Zeilen `02` und `06` wird der Beginn bzw. das Ende der Liste markiert. Die Zeilen `03`, `04` und `05` definieren die URI-Liste, bestehend aus den Konzepten `#AmountPlausible`, `#Aggregate` und `#Roadster`. Die Zeile `07` schließt die Definition der URI-Liste ab. Die URI-Liste findet ihre Anwendung in Kapitel Serviceoperation BZM auf Seite 103 in Zeile `24`.

Geschäftsobjekt BZM_IN

```
04 <xsd:complexType name="BZM_IN">
05   <xsd:sequence>
06      <xsd:element name="SNR" nillable="false" type="xsd:string"
07       wssem:modelReference="AutomotiveDAI#DAISachnummer"/>
08   </xsd:sequence>
09   <xsd:sequence><xsd:element name="Amount" nillable="false"
10       type="xsd:integer" wssem:modelReference="AutomotiveDAI#Amount"/>
11   </xsd:sequence>
```

```
12   <xsd:sequence><xsd:element name="DateTime" nillable="false"
13       type="xsd:date" wssem:modelReference=
14                           "AutomotiveDAI#DAILiefertermin"/>
15   </xsd:sequence>
16 </xsd:complexType>
```

Das Geschäftsobjekt `BZM_IN` wird in Form eines `<complexType>` realisiert. In Zeile `06, 09` und `13` werden alle benötigten Variablen definiert. Der fachliche Datentyp *Sachnummer (SNR)* in Zeile `06` wird um `Not Null` erweitert und als primitiver technischer Datentyp `xsd:string` definiert. Der fachliche Datentyp *Amount* in Zeile `09` wird ebenfalls um `Not Null` erweitert und ebenfalls als primitiver technischer Datentyp `xsd:integer` definiert. Zusätzlich wird in Zeile `07` auf das Konzept `#DAISachnummer` referenziert und somit semantisch eindeutig definiert. Analog zur Erweiterung der *Sachnummer (SNR)* verhält es sich zu *Amount* in Zeile `09` und `10` sowie zu *DateTime* in Zeile `13` und `14`.

Geschäftsobjekt BZM_OUT

```
17 <xsd:simpleType name="BZM_OUT">
18    <xsd:restriction base="xsd:datetime"
19     wssem:modelReference="AutomotiveDAI#DAISollTermin">
20   </xsd:restriction>
21 </xsd:simpleType>
```

Das Geschäftsobjekt `BZM_OUT` wird in Form eines `<simpleType>` realisiert. Der fachliche Datentyp wird in Zeile `18` als technischer Datentyp `datetime` definiert. Zusätzlich wird in Zeile `19` auf das Konzept `#DAISollTermin` referenziert und somit semantisch eindeutig definiert.

Serviceoperation BZM

```
22 <operation name="ermittleSollTermin" pattern=wsdl:in-out
23   <wssem:precondition name="AmountPlausible"
24    wssem:modelReference="DAI:BZM-P-uri-list">
25   <wssem:effect name="SollTerminCalculated"
26    wssem:modelReference="AutomotiveDAI#DAISollTerminVerfuegbar"/>
27 </operation>
```

Die Operation `ermittleSollTermin` wird mit den Bezeichnern `AmountPlausible` als Vorbedingung und `SollTerminCalculated` als Nachbedingung ausgezeichnet. In Zeile `24` wird die Vorbedingung mit der URI-Liste `DAI:BZM-P-uri-list` (siehe Kapitel 7.1.2 auf Seite 102) und in Zeile `26` die Nachbedingung mit dem Konzept `#DAISollTerminVerfuegbar` verbunden.

7.1.3 Gegenüberstellung der Serviceoperationen

Die Möglichkeiten zur Anreicherung semantischer Informationen wurden für die beiden Serviceoperationen `MASPLUS.getSollTermin` und `BZM.ermittleSollTermin` in den Kapiteln 7.1.1 und 7.1.2 auf den Seiten 100 und 102 beschrieben.

	`MASPLUS.getSollTermin`	`BZM.ermittleSollTermin`
`I`	→ `#DAISachnummer` (Sachnummer)	→ `#DAISachnummer` (SNR)
	→ `#Menge` (Menge)	→ `#Amount` (Amount)
	→ `#DAILiefertermin` (Liefertermin)	→ `#DAILiefertermin` (DateTime)
`O`	→ `#DAISollTermin` („SimpleType“)	→ `#DAISollTermin` („SimpleType“)
`P`	→ `#MengePlausibel`	→ `#AmountPlausible`
	→ `#Gear`	→ `#Aggregate`
	→ `#SLK`	→ `#Roadster`
`E`	→ `#DAISollTerminVerfügbar`	→ `#DAISollTerminVerfügbar`

Tabelle 7.1: Gegenüberstellung der Serviceoperationen

In Tabelle 7.1 wurden die semantischen Informationen aus den beiden Schnittstellenbeschreibungen MASPlus.wsdl (siehe Anhang P.1 auf Seite 219) und BZM.wsdl (siehe Anhang P.2 auf Seite 221) extrahiert. Für die Anwendung des Algorithmus SSF (siehe Kapitel 6.2 auf Seite 88) ist die Extrahierung semantischer Informationen aus den beiden Dateien (MASPlus.wsdl, BZM.wsdl) notwendig. Die technischen Informationen, wie bspw. Sachnummer, Menge und Liefertermin etc. werden ausschließlich zur besseren Orientierung (in Klammern) aufgelistet. Unter dem Terminus technicus Liefertermin kann bspw. ein exakter Zeitpunkt oder eine Kalenderwoche verstanden werden – so kann man unter mehreren unterschiedlichen technischen Informationen wie bspw. SNR-1 und SNR-2 den identischen Terminus technicus Sachnummer verstanden. Folglich ist eine Ontologie in Kombination mit Inferenz und einem Algorithmus (bspw. der SSF) notwendig, um die Distanz zwischen den beiden Schnittstellenbeschreibungen zu ermitteln.

Generell ist zwischen Ein- (`I`) und Ausgabeparameter (`O`) sowie Vor- (`P`) und Nachbedingung (`E`) der unterschiedlichen Serviceoperationen `MASPLUS.getSollTermin` und `BZM.ermittleSollTermin` zu unterscheiden. Ein- und Ausgabeparameter: Der Algorithmus SSF schlägt eine Abbildung mit dem Ziel einer technischen Realisierung und der Bewertung der Distanz zwischen `MASPLUS.getSollTermin` und `BZM.ermittleSollTermin` vor. Vor- und Nachbedingungen: Im Gegensatz zu Ein- und Ausgabeparameter wird die vorgeschlagene Abbildung hier ausschließlich zur Bewertung der Distanz benötigt. Dabei spielt die Reihenfolge der semantischen Auszeichnung keine Rolle. Bspw. wird die Nachbedingung (`P`) der Serviceoperation `BZM.ermittleSollTermin` mit folgenden semantischen Informationen in der Reihenfolge `#AmountPlausible`, `#Aggregate` und `#Roadster` genau gleich bewertet wie in der Reihenfolge `#Roadster`, `#AmountPlausible` und `#Aggregate`.

7.2 Ansatz zur domänenspezifischen Ontologie

Der Terminus technicus Ontologie und die Technologie OWL wurden bereits in den Kapiteln 2.3.1 und 2.3.2 eingeführt. Der vorgeschlagene Lösungsansatz EMEO-Layer verfolgt das primäre Ziel der Identifizierung äquivalenter Services durch semantische semiautomatische Unterstützung in der Entwicklungsumgebung. Zur primären Zielerreichung ist die notwendige Voraussetzung eine Externalisierung des Know-hows der Personengruppe A. Zur Erfüllung der notwendigen Voraussetzung wird eine Ontologie erstellt, die die beiden sekundären Ziele, allgemein gültiges Verständnis der Domäne Automotive und Unterstützung der Mensch-Maschine Kommunikation verfolgt.

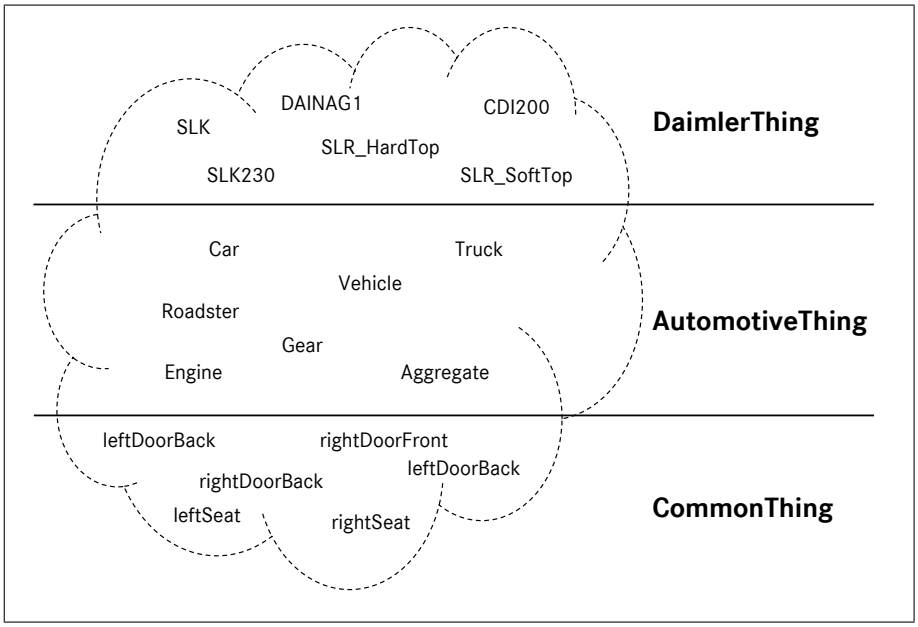

Abbildung 7.1: Konzepte in einer domänenspezifischen Ontologie

Zur Externalisierung des Know-hows der Personengruppe A wird zur praktischen Anwendung eine Aufteilung in die drei Bereiche *CommonThing*, *AutomotiveThing* und *DaimlerThing* vorgeschlagen (siehe Abbildung 7.1). Die drei Bereiche werden in Form abstrakter Konzepte (*CommonThing*, *AutomotiveThing* und *DaimlerThing*) modelliert, sind nicht disjunkt und dienen zur Kategorisierung des Know-hows der Personengruppe A. Die Konzepte und Relationen im Bereich *CommonThing* stellen das Basiswissen der Domäne dar und sind disjunkt definiert. Sie werden auch als primitive Konzepte bzw. primitive Begriffe bezeichnet. Der Bereich *AutomotiveThing* beinhaltet Konzepte und Relationen, die ein gemeinsames Verständnis in der Domäne Automotive definieren und somit das gemeinsame Verständnis aller Automobilhersteller und Lieferanten repräsentieren. Der Bereich *DaimlerThing* stellt das interne spezifische Wissen dar und repräsentiert somit das Wissen des Fachbereichs der Daimler AG.

Alle drei abstrakte Konzepte stellen in der Summe das minimal angenommene spezifische Wissen der Personengruppe A dar, das notwendig dafür ist, die potentiellen Gefahrenquellen (siehe Kapitel 5 auf Seite 81) zu eliminieren. Abbildung 7.2 illustriert die Hierarchie der drei Bereiche als Teilkonzepte von `owl:Thing` ($\top$).

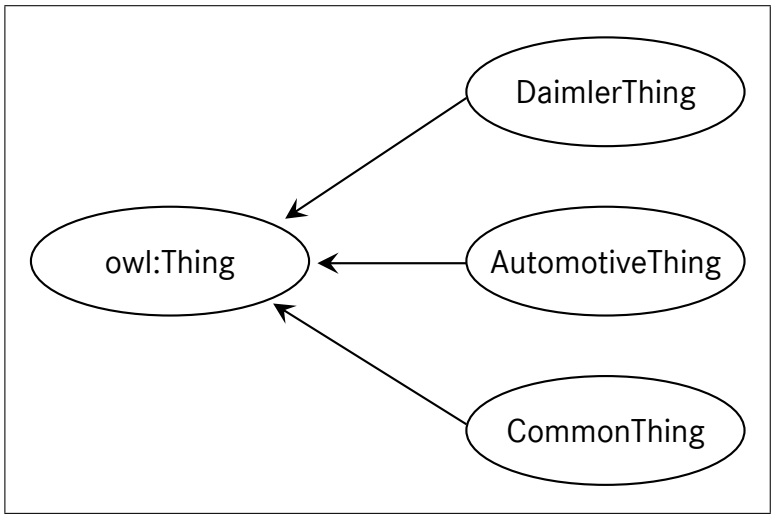

Abbildung 7.2: Hierarchie der drei Bereiche

Kapitel 7.1 auf Seite 100 beschreibt die semantische Anreicherung der WSDL-Dokumente durch IOPEs und die Klassifizierung der Service-Schnittstellen. Zur Erfüllung des primären Ziels, die Identifizierung äquivalenter Services durch semantische semiautomatische Unterstützung in der Entwicklungsumgebung, musste die minimale Breite und die minimale Tiefe des angenommenen spezifischen, deklarativen Wissens der Personengruppe A ermittelt werden. Das minimal angenommene spezifische Wissen der Personengruppe A wurde bereits in Form von Konzepten, zentral in einer Ontologie abgelegt und durch die iterative Methode On-To-Knowledge (OTKM) in Zusammenarbeit mit dem Fachbereich validiert (siehe Kapitel 6.6 auf Seite 94).

In diesem Kapitel werden alle notwendigen und hinreichenden Konzepte der drei Bereiche *CommonThing* (siehe Kapitel 7.2.2 auf Seite 107), *AutomotiveThing* (siehe Kapitel 7.2.3 auf Seite 108) und *DaimlerThing* (siehe Kapitel 7.2.4 auf Seite 109) formal in $\mathcal{SHOIN}(\mathcal{D})$ beschrieben. Die Konzepte und Relationen werden in Anlehnung an die terminologische Abbildung der Beschreibungslogik in OWL aus Tabelle 2.3 auf Seite 42 definiert. Ein *unbestimmtes Konzept (engl. primitive concept)* wird durch eine oder mehrere *notwendige Bedingungen* ($\sqsubset$, $\sqsubseteq$) definiert. So wird ein *bestimmtes Konzept (engl. defined concept)* durch mindestens eine *notwendige und hinreichender Bedingung* ($\equiv$) definiert.

7.2.1 Relationen

Die in Abbildung 7.2 auf Seite 106 vorgestellte Einteilung soll eine mögliche exemplarische Kategorisierung darstellen, die Anwendung bei der Definition der Konzepte findet. Im Gegensatz dazu werden Relationen nicht kategorisiert. Im Folgenden werden den Relationen `hasDoorway`, `hasTop` und `hasSeat` die Bereiche *(engl. range)* `DoorValuePartition`, `TopValuePartition`, `SeatValuePartition` zugewiesen.

$$\top \sqsubseteq \forall hasDoorway.DoorValuePartition \tag{7.1}$$
$$\top \sqsubseteq \forall hasTop.TopValuePartition \tag{7.2}$$
$$\top \sqsubseteq \forall hasSeat.SeatValuePartition \tag{7.3}$$

Die Relationen `hasEngine` und `isValid` werden als funktional definiert.

$$\top \sqsubseteq (\leq 1\ hasEngine) \tag{7.4}$$
$$\top \sqsubseteq (\leq 1\ isValid) \tag{7.5}$$

Die Relation `matchesTo` wird als symmetrisch und die Relationen `isAssambledIn` und `isSameAS` werden als transitiv definiert.

$$matchesTo \equiv matchesTo^{-} \tag{7.6}$$
$$isAssembledIn^{+} \in \Re_{+} \tag{7.7}$$
$$isSameAs^{+} \in \Re_{+} \tag{7.8}$$

7.2.2 Das abstrakte Konzept *CommonThing*

Die beiden Konzepte `Amount` und `Menge` werden als Synonym durch die transitive Relation `isSameAs` in Form notwendiger und hinreichender Bedingungen definiert.

$$Amount \equiv \exists isSameAs.Menge \tag{7.9}$$
$$Amount \sqsubseteq DataValuePartition \tag{7.10}$$
$$Menge \equiv \exists isSameAs.Amount \tag{7.11}$$
$$Menge \sqsubseteq DataValuePartition \tag{7.12}$$

Die Konzepte `leftDoorBack`, `rightDoorBack`, `leftDoorFront` und `rightDoorFront` werden als disjunkt definiert.

$$\bot \equiv leftDoorBack \sqcap rightDoorBack \tag{7.13}$$
$$\bot \equiv leftDoorBack \sqcap leftDoorFront \tag{7.14}$$
$$\bot \equiv leftDoorBack \sqcap rightDoorFront \tag{7.15}$$
$$\bot \equiv rightDoorBack \sqcap leftDoorFront \tag{7.16}$$

$$\bot \equiv rightDoorBack \sqcap rightDoorFront \tag{7.17}$$

$$\bot \equiv leftDoorFront \sqcap rightDoorFront \tag{7.18}$$

Die Konzepte `backSeat`, `leftSeat` und `rightSeat` werden als disjunkt definiert.

$$\bot \equiv backSeat \sqcap leftSeat \tag{7.19}$$

$$\bot \equiv backSeat \sqcap rightSeat \tag{7.20}$$

$$\bot \equiv leftSeat \sqcap rightSeat \tag{7.21}$$

Die Konzepte `topHardTop` und `topSoftTop` werden als disjunkt definiert.

$$\bot \equiv topHardTop \sqcap topSoftTop \tag{7.22}$$

7.2.3 Das abstrakte Konzept *AutomotiveThing*

Das Konzept `Aggregate` wird durch die Konzepte `Vehicle` und `Gear` sowie durch die Relationen `isAssembledIn` und `matchesTo` in Form notwendiger und hinreichender Bedingungen definiert.

$$Aggregate \equiv \exists isAssembledIn.Vehicle \sqcup \exists matchesTo.Gear \tag{7.23}$$

Das Konzept `Engine` wird durch das Konzept `Gear` sowie durch die Relation `matchesTo` in Form notwendiger und hinreichender Bedingungen definiert.

$$Engine \equiv \exists matchesTo.Gear \tag{7.24}$$

Das Konzept `Gear` wird durch das Konzept `Vehicle` sowie durch die Relation `isAssembledIn` in Form notwendiger und hinreichender Bedingungen definiert.

$$Gear \sqsubseteq \exists isAssembledIn.Vehicle \tag{7.25}$$

Das Konzept `Roadster` wird durch die Konzepte `topSoftTop` und `Car`, die Relation `hasTop` sowie durch die Datentyp-Relationen `hasDoorway` und `hasSeat` in Form notwendiger und hinreichender Bedingungen definiert. Alternativ ist die Definition der Relationen `nrOfDoorways` und `nrOfSeats` anstelle von `hasDoorway` und `hasSeat` möglich.

$$\begin{aligned} Roadster \equiv\ & (hasDoorway = 2) \sqcap (hasSeat = 2) \sqcap \\ & \exists hasTop.topSoftTop \sqcap Car \end{aligned} \tag{7.26}$$

7.2.4 Das abstrakte Konzept *DaimlerThing*

Das Konzept `SLK` wird durch die Konzepte `leftFrontDoor`, `rightFrontDoor`, `leftSeat`, `rightSeat`, `topSoftTop` und `DAICar` sowie durch die Relationen `hasTop`, `hasDoorway` und `hasSeat` in Form notwendiger Bedingungen definiert (vgl. Diskussion in Kapitel 6.6 auf Seite 94).

$$\begin{aligned} SLK \sqsubseteq\ & =1\ hasDoorway.leftFrontDoor \sqcap \\ & =1\ hasDoorway.rightFrontDoor \sqcap \\ & =1\ hasSeat.leftSeat \sqcap\ =1\ hasSeat.rightSeat \sqcap \\ & \forall hasSeat\ (rightSeat \sqcup leftSeat) \sqcap \\ & \forall hasDoorway\ (rightDoorFront \sqcup leftDoorFront) \sqcap \\ & \exists hasTop.topSoftTop\ \sqcap\ DAICar \end{aligned} \tag{7.27}$$

Die Konzepte `SLK200` und `SLK230` werden durch die Konzepte `SLK`, `Compressor200` und `Compressor230` sowie durch die Relation `hasEngine` in Form notwendiger Bedingungen definiert.

$$SLK200 \equiv \forall hasEngine.Compressor200 \sqcap SLK \tag{7.28}$$

$$SLK230 \equiv \forall hasEngine.Compressor230 \sqcap SLK \tag{7.29}$$

Das Konzept `SLRSoftTop` wird als Konzept SLK (`rightFrontDoor`, `leftSeat`, `rightSeat`, `topSoftTop`, `DAICar`, `hasTop`, `hasDoorway` und `hasSeat`) und durch die Relation `hasEngine.V8` in Form notwendiger Bedingungen definiert.

$$SLRSoftTop \equiv SLK \sqcup \exists hasEngine.V8 \tag{7.30}$$

Das Konzept `DAIEngine` wird durch das Konzept `DAICar` sowie durch die Relation `matchesTo` in Form notwendiger und hinreichender Bedingungen definiert.

$$DAIEngine \equiv \forall matchesTo.DAICar \tag{7.31}$$

Das Konzept `DAIGear` wird durch das Konzept `DAICar` sowie durch die Relation `isAssembledIn` in Form notwendiger Bedingungen definiert.

$$DAIEngine \sqsubseteq \forall isAssembledIn.DAICar \tag{7.32}$$

Das Konzept `DAINAG1` wird durch die Konzepte `DAIGear`, `SLK200` und `SLK230` sowie durch die Relation `isAssembledIn` in Form notwendiger Bedingungen definiert.

$$\begin{aligned} DAINAG1 \sqsubseteq (& \exists isAssembledIn.SLK200 \sqcup \\ & \exists isAssembledIn.SLK230\ ) \\ & \sqcap DAIGear \end{aligned} \tag{7.33}$$

Die Konzepte `DAILiefertermin`, `DAISachnummer` und `DAISollTermin` werden als disjunkt definiert.

$$\bot \equiv DAILiefertermin \sqcap DAISachnummer \tag{7.34}$$

$$\bot \equiv DAILiefertermin \sqcap DAISollTermin \tag{7.35}$$

$$\bot \equiv DAISachnummer \sqcap DAISollTermin \tag{7.36}$$

Das Konzept `DAISollTerminVerfuegbar` wird durch die Konzepte `DAISystem` und `DAISollTermin` sowie durch die Relation `isValid` in Form notwendiger und hinreichender Bedingungen definiert.

$$DAISollTerminVerfuegbar \equiv \exists isValid.DAISollTermin \sqcap DAISystem \tag{7.37}$$

Das Konzept `AmountPlausible` wird durch die Konzepte `Amount` und `BZM` sowie durch die Relation `isValid` in Form notwendiger und hinreichender Bedingungen definiert.

$$AmountPlausible \equiv \exists isValid.Amount \sqcap BZM \tag{7.38}$$

Das Konzept `MengePlausibel` wird durch die Konzepte `Menge` und `MASPlus` sowie durch die Relation `isValid` in Form notwendiger und hinreichender Bedingungen definiert.

$$MengePlausibel \equiv \exists isValid.Menge \sqcap MASPlus \tag{7.39}$$

7.2.5 Implementierung

In den Kapiteln 7.2.1, 7.2.2, 7.2.3 und 7.2.4 ab der Seite 107, wird das minimal anzunehmende deklarative Wissen der Personengruppe A formal beschrieben. Bei der Implementierung in OWL findet bei den formalen Definitionen in 7.26, 7.27 und 7.2.4 eine Abweichung statt. Grund für die Abweichung ist die fehlende Unterstützung der Datentypen-Eigenschaften in der aktuellen DIG-Schnittstelle (DL Implementation Group) Version 1.1. Somit wurden die beiden Konzepte `topHardTop` und `topSoftTop` als Objekte des Konzeptes `TopValuePartition` definiert.

Die beiden Grafiken im Anhang L.4 auf Seite 196 illustrieren die modellierte Ontologie O_1, die durch Inferenz erzeugte neue Ontologie O_2 sowie alle Aktionen, um O_1 auf O_2 abzubilden (vgl. Kapitel 7.4 ab Seite 116).

Im Folgenden werden die ausschlaggebenden bzw. verursachenden Axiome der erzeugten Ontologie O_2 beschrieben. Die Konzepte `Menge` und `Amount` werden als äquivalent ($\equiv$) erkannt, da beide Konzepte via transitiver Relation `isSameAs` zueinander in Form notwendiger und hinreichender Bedingungen definiert wurden.

Folglich werden die beiden Konzepte `AmountPlausible` und `MASPlus` als äquivalent ($\equiv$) erkannt, da beide Konzepte durch die funktionale Relation `isValid` mit den Konzepten `Amount` bzw. `Menge` in Form notwendiger und hinreichender Bedingungen definiert wurden. Das Konzept `Aggregate` wurde als Oberkonzept der beiden Konzepte `Engine` und `Gear` erkannt, da das Konzept `Aggregate` mit der Relation `isAssembledIn` bzw. `matchesTo` via Existenzquantor ($\exists$) und das Konzept `Gear` mit der Relation `isAssambledIn` sowie `Engine` mit der Relation `matchesTo` in Form notwendiger und hinreichender Bedingungen definiert wurden. Die ausschlaggebenden bzw. verursachenden Axiome zur Ermittlung des Konzeptes `Roadster` als Oberkonzept des Konzeptes `SLK` wurde in Kapitel 6.6 auf Seite 94 ausführlich diskutiert. Für die Bewertung der semantische Informationen (Konzepte) in der Form Eingabe (`I`), Ausgabe (`O`), Vor- (`P`) und Nachbedingung (`E`) ist die Inferenz nicht ausreichend. Somit wird die Erstellung einer sortierten Liste möglicher äquivalenter Serviceoperationen (Kandidaten) in Kapitel 6.4 auf Seite 91 theoretisch erarbeitet (Semantik-Service-Finder) und in Kapitel 7.4 auf Seite 116 validiert. Im Anhang N auf Seite 203 wird die Ontologie O_1 in OWL illustriert.

7.3 Mediator prototypisch in Java

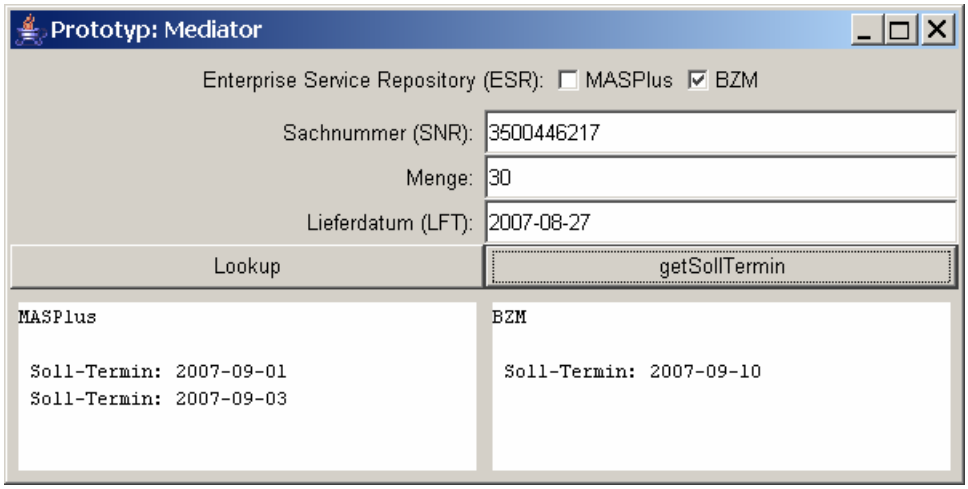

Abbildung 7.3: Prototyp Mediator in Java

Die im EMEO-Layer vorgeschlagene Mediation wurde prototypisch in Java implementiert (der komplette Code wird im Anhang K auf Seite 183 dokumentiert). Die Kernelemente der Mediation (Förderung der losen Kopplung durch Generalisierung der Unterklassen `WidgetBZM` und `WidgetMASPlus` in Form einer abstrakten Klasse `Widget` sowie die gekapselte Steuerung der Kommunikation in der Klasse `Mediator`) werden im Folgenden beschrieben. Die Abbildung 7.3 stellt die grafische Oberfläche zur Dateneingabe bzw. -ausgabe und zur Steuerung der Mediation dar. Der Prototyp besteht aus zwei Checkboxen (`MASPlus`, `BZM`) sowie zwei Schaltflächen *(engl. Buttons)* (`Lookup`, `getSollTermin`) zur Steuerung, drei Textfelder (`SNR`, `Menge`, `LFT`) zur Eingabe und zwei Memofelder (`MASPlus`, `BZM`) zur Ausgabe. Die Eingaben der Sachnummer (SNR), Menge und Lieferdatum (LFT) liegen korrekt in den Textfeldern vor. Mit der Auswahl der Checkboxen werden dem Service `getSollTermin` die aktuell verfügbaren Service-Provider ((a) nur `MASPlus` oder (b) nur `BZM` oder (c) `MASPlus` und `BZM` oder (d) kein Provider verfügbar) über das Enterprise Service Repository (ESR) mitgeteilt. Falls das Ereignis (d) eintritt, wird auf der Console `Lookup ESR: none.` ausgegeben und kein Service aufgerufen. Falls das Ereignis (a) oder (b) eintritt, wird auf der Console `Lookup ESR: MASPlus` bzw. `Lookup ESR: BZM` ausgegeben und der Service `MASPlus` bzw. `BZM` aufgerufen. Falls das Ereignis (c) eintritt, wird auf der Console `Lookup ESR: MASPlus` (Standardwert) ausgegeben und der Service `MASPlus` aufgerufen. Über die Schaltfläche `Lookup` wird der aktuell verfügbare Service ermittelt. Über die Schaltfläche `getSollTermin` werden die Eingabedaten an der Service übergeben. Bei Auslösen der Schaltfläche `getSollTermin` wird automatisch die Schaltfläche `Lookup` zur Ermittlung der verfügbaren Services ausgeführt. Die Console gibt folgende Werte aus:

```
Lookup ESR: MASPlus
Lookup ESR: MASPlus
Lookup ESR: BZM
Lookup ESR: none.
```

Bei der Implementierung der Mediation ist die Implementierung der abstrakten Klasse `Vermittler` obsolet. Die Notwendigkeit, eine abstrakte Klasse `Vermittler` einzuführen, ist nur durch eine gegebene Kommunikation mit weiteren abstrakten Vermittler-Klassen sinnvoll (vgl. Kapitel 2.9).

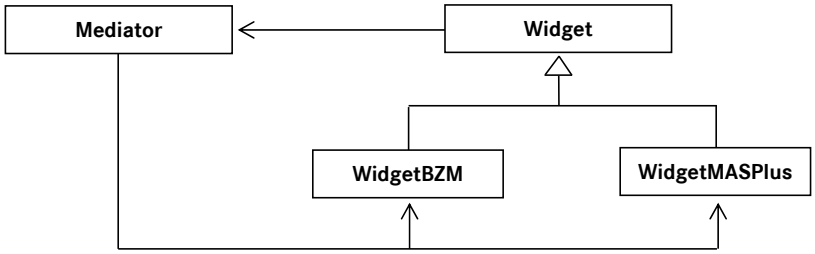

Abbildung 7.4: Mediator Klassendiagramm

Das Klassendiagramm (siehe Abbildung 7.4) illustriert grafisch die Abhängigkeiten der Klassen untereinander. Das Widget *(dt. Komponente einer Benutzeroberfläche)* stellt die Generalisierung der Unterklassen `WidgetBZM` und `WidgetMASPlus` dar. Der Mediator kennt, koordiniert und verwaltet die Klasseninstanzen der beiden Widgets `WidgetBZM` und `WidgetMASPlus` durch die Implementierung des Gesamtverhaltens. Die beiden Widgets kommunizieren ausschließlich über den Mediator und nicht mit sich selbst. Der Mediator koordiniert die Kommunikation zwischen den Anfragen und den Widgets, indem er beim Senden und Empfangen die Anfragen an die Objekte vermittelt. Bei der folgenden exemplarischen Darstellung werden nur Code-Fragmente aufgezeigt, die direkt für die Implementierung des Mediator Verhaltensmusters nötig sind. Somit werden keine Imports, Packages, Panels etc. dargestellt. Der komplette Quellcode wird im Anhang K auf Seite 183 gelistet.

Die folgende Beschreibung der Code-Fragmente stellt die Klassen sowie Methoden kursiv in Grundschrift (bspw. *Mediator(), calc()*) und die Objekte in der nichtproportionalen Schriftart Courier (bspw. `med`) dar. Die Code-Fragmente selbst werden komplett in der nichtproportionalen Schriftart Courier dargestellt. Im Konstruktor der Klasse *Main()* werden die Klassen *Mediator()* (`med`), *WidgetMASPlus()* (`widgetMASPlus`) und *WidgetBZM()* (`widgetBZM`) instanziiert.

```
Mediator      med           = new Mediator();
WidgetMASPlus widgetMASPlus = new WidgetMASPlus(med, MASPLUS);
WidgetBZM     widgetBZM     = new WidgetBZM(med, BZM);
```

Die Klasse *Widget()* definiert die abstrakte Methode *calc()* und registriert alle Unterklassen im Konstruktor.

```
public abstract void calc(Calendar cal, Integer imenge, Long isnr);

public Widget (Mediator m, String s) {
   append(s + "\n");
   m.register(this);
}
```

Die beiden Klassen *WidgetMASPlus()* und *WidgetBZM()* symbolisieren die Service-Provider `MASPlus` und `BZM`. Beide Klassen implementieren die Methode *calc()* zur Berechnung des Soll-Termins. Die Signatur der Methoden *WidgetMASPlus.calc()* und *WidgetBZM.calc()* sowie der errechneten Rückgabe `Liefertermin` sind syntaktisch und semantisch äquivalent. Die unterschiedliche Implementierung der beiden Methoden wird, durch die Anwendung des Kommutativgesetzes, vereinfacht skizziert. Im Folgenden wird die Klasse *WidgetMASPlus()* dargestellt. Im Konstruktor wird das Objekt `med` und der Objektname übergeben. Die Implementierung der Methode *calc()* berechnet den Liefertermin.

```
public WidgetMASPlus(Mediator med, String s) {
   super(med, s);
}

public void calc(Calendar cal, Integer imenge, Long isnr) {
   cal.add(Calendar.DAY_OF_WEEK,
   (int) (((imenge.intValue()) / 40)* -1 *
           (isnr.doubleValue() % 7)));
}
```

Im Folgenden wird die Klasse *WidgetBZM()* dargestellt. Im Konstruktor wird das Objekt `med` und der Objektname übergeben. Die Implementierung der Methode *calc()* berechnet den Liefertermin. Die unterschiedliche Implementierung der beiden Methoden wird, durch die Anwendung des Kommutativgesetzes, vereinfacht skizziert.

```
public WidgetBZM(Mediator med, String s) {
   super(med, s);
}

public void calc(Calendar cal, Integer imenge, Long isnr) {
   cal.add(Calendar.DAY_OF_WEEK,
   (int) ((-1 * (imenge.intValue()) / 40) *
          (isnr.doubleValue() % 7)));
}
```

Die Klasse *Mediator()* kapselt die komplett abzubildende Logik in den Methoden *setLookUpValue()*, *getWidget()* und *call()*. Die beiden Checkboxen `masPlus` und `bzm` simulieren den Eintrag im ESR. Die private Methode *setLookUpValue()* ermittelt das aktuell gültige Widget zur Laufzeit.

```
private void setLookUpValue() {
   if (masPlus.isSelected()) {
      lookUpValue = MASPLUS;
   } else if (bzm.isSelected()) {
      lookUpValue = BZM;
   } else
      lookUpValue = "none.";
}
```

Die private Methode *getWidget()* ermittelt, mit Hilfe der Klasse *Iterator()*, die aktuelle Instanz (`widgetMASPlus` oder `widgetBZM`) des zurzeit gültigen Widget.

```
private Widget getWidget(String lookUpValue) {
   Iterator iter = widgets.iterator();
   while (iter.hasNext()) {
      Widget widget = (Widget) iter.next();
      if (lookUpValue.equals(MASPLUS) &&
          widget instanceof WidgetMASPlus) {
         return widget;
      } else if (lookUpValue.equals(BZM) &&
                  widget instanceof WidgetBZM) {
         return widget;
      }
   }
   return null;
}
```

Die private Methode *call()* ruft die Methode *calc()* beim aktuell gültigen Widget (`widgetMASPlus` oder `widgetBZM`) auf und gibt den ermittelten Liefertermin aus.

```
private void call(Calendar c, Integer imenge,
                  Long lsnr, Widget widget) {
   widget.calc(c, imenge, lsnr);
   widget.append("\n Soll-Termin: " + df.format(c.getTime()));
}
```

7.4 Semantik-Service-Finder (SSF) in Java

Die folgende exemplarische Darstellung zeigt nur Code-Fragmente auf, die direkt für die Implementierung des SSF nötig sind, womit keine Imports und Packages etc. dargestellt werden. Der komplette Quellcode wird im Anhang L auf Seite 189 gelistet. Die folgende Beschreibung der Code-Fragmente stellt die Klassen sowie Methoden kursiv in Grundschrift (bspw. *RankService(), connectToReasoner()*) und die Objekte in der nichtproportionalen Schriftart Courier (bspw. `mapServicesInESR`) dar. Die Code-Fragmente selbst werden komplett in der nichtproportionalen Schriftart Courier dargestellt. Der SSF besteht aus den drei Klassen *Main()*, *ESRServices()* und *RankService()*. Die Klasse *Main()* verwaltet die semantischen Informationen des Service `MASPlus` in der HashMap `mapMASPlusgetSollTermin` und steuert die Kommunikation zwischen den beiden Klassen *ESRServices()* und *RankService()*. Die Klasse *RankService()* kommuniziert mit der Inferenzmaschine via URL `http://localhost:8080` und bewertet, nach dem Schließen der Inferenzmaschine, den Abstand zwischen den Konzepten in der Wissensbasis. Die Klasse *ESRServices()* verwaltet alle verfügbaren Services aus dem ESR in der HashMap `rankService`. Das Lesen aus dem ESR wird über die Methode *hasNextServiceOperation()* simuliert. Im Konstruktor der Klasse *Main()* werden zuerst beide HashMaps (`mapMASPlusgetSollTermin`, `mapServicesInESR`) instanziiert. Ebenso werden zur Verwaltung semantischer Informationen der Serviceoperation `MASPlus.getSollTermin` alle verfügbaren Serviceoperationen aus dem ESR und die Klassen *ESRServices()* (`serviceESR`) bzw. *RankService()* (`rankService`) instanziiert. In der Klasse *ESRServices()* werden die als Zugriffsschlüssel auf die beiden HashMaps in Form von `SERVICE` zur Speicherung der Service-Bezeichnung, `OP` zur Speicherung des Operation, `I` zur Speicherung des Inputs, `O` zur Speicherung des Outputs, `P` zur Speicherung der Preconditions und `E` zur Speicherung des Effects als konstante Variablen definiert.

```
HashMap mapMASPlusgetSollTermin = new HashMap();
HashMap mapServicesInESR        = new HashMap();
ESRServices serviceESR          = new ESRServices();
RankService rankService         = new RankService();

mapMASPlusgetSollTermin.put(serviceESR.SERVICE,"MASPlus");
mapMASPlusgetSollTermin.put(serviceESR.OP,     "getSollTermin");
mapMASPlusgetSollTermin.put(serviceESR.I,
                            "DAISachnummer;Menge;DAILiefertermin");
mapMASPlusgetSollTermin.put(serviceESR.O,      "DAISollTermin");
mapMASPlusgetSollTermin.put(serviceESR.P,      "MengePlausibel");
mapMASPlusgetSollTermin.put(serviceESR.E,
                            "DAISollTerminVerfuegbar");
```

Nach der Instanziierung der beiden Klassen und der beiden HashMaps werden die Methoden *connectToReasoner()* und *doInference()*, zur Verbindung mit der Inferenzmaschine und die Ausführung der Inferenz, aufgerufen.

```
rankService.connectToReasoner();
rankService.doInference();
```

Nach der Inferenz werden in einer Schleife alle Service-Operationen aus dem ESR gelesen und via Klasseninstanz `rankService` bewertet sowie das Ergebnis der Bewertung ausgegeben (Console). Die Methoden *printFooter()* und *printRanking()* übergeben eine kumulierte Bewertung der Service-Operation sowie am Schluss eine bewertete Liste aller Service-Operationen an die Console.

```
do {
   rankService.printHeader(mapMASPlusgetSollTermin,
                           mapServicesInESR);

   rankService.detectEquivalence(serviceESR.I,
                                 mapMASPlusgetSollTermin,
                                 mapServicesInESR);
   rankService.detectEquivalence(serviceESR.O,
                                 mapMASPlusgetSollTermin,
                                 mapServicesInESR);
   rankService.detectEquivalence(serviceESR.P,
                                 mapMASPlusgetSollTermin,
                                 mapServicesInESR);
   rankService.detectEquivalence(serviceESR.E,
                                 mapMASPlusgetSollTermin,
                                 mapServicesInESR);

   rankService.printFooter(serviceESR.I,
                           serviceESR.O,
                           serviceESR.P,
                           serviceESR.E);

   mapServicesInESR = serviceESR.getNextServiceOperation();

}  while (serviceESR.hasNextServiceOperation());

rankService.printRanking();
```

Die Klasse *ESRServices()* definiert in Form von Konstanten die Werte der Service-Bezeichnung, der Operation, des Inputs, des Outputs, der Preconditions und Effects.

```
final String SERVICE = "Service";
final String OP      = "Operation";
final String I       = "INPUT";
final String O       = "OUTPUT";
final String P       = "PRECONDITION";
final String E       = "EFFECT";
```

Die private Eigenschaft `hashCount` der Klasse *ESRServices()* verwaltet durch die privaten Methoden *getHashCount()* und *setHashCount()* die Anzahl der eingelesenen Serviceoperationen. Die private Methode *addServiceToHashMap()* verwaltet alle Fragmente (IOPE) der Serviceoperationen in einer HashMap. Dieses Konstrukt bildet das Lesen der Serviceoperationen aus dem ESR ab (Simulation).

```
private void addServiceToHashMap() {
   if (getHashCount() == 1) {
      hashMap.clear();
      hashMap.put(SERVICE, "BZM");
      hashMap.put(OP,      "uebergebeDaten");
      hashMap.put(I,       "DAILachnummer;Amount;DAILiefertermine");
      hashMap.put(O,       "DAIIstTermin;Xmount");
      hashMap.put(P,       "AmountPlausibles");
      hashMap.put(E,       "DAISollTerminVerfuegbarer");
   }
}
```

Die Klasse *RankService()* definiert in Form von Konstanten den Pfad zur Ontologie und die URI zur Inferenzmaschine.

```
final String ONTOLOGY_URL = "D:/Projekte/_Promotion/owl/DAI.owl";
final String REASONER_URL = "http://localhost:8080";
```

Die beiden Methoden *connectToReasoner()* und *doInference()* der Klasse *RankService()* stellen die Verbindung zur Inferenzmaschine her und führen die Inferenz aus. Nach der Durchführung der Inferenz ist die neue Ontologie O_2 im Arbeitsspeicher der Inferenzmaschine und somit über die DIG-Schnittstelle (DL Implementation Group) verfügbar.

```
public boolean connectToReasoner() throws FileNotFoundException {
   model = ProtegeOWL.createJenaOWLModelFromReader(new BufferedReader(
      new FileReader(ONTOLOGY_URL)));
   ReasonerManager reasonerManager = ReasonerManager.getInstance();
   reasoner = reasonerManager.getReasoner(model);
   reasoner.setURL(REASONER_URL);
   return true;
}
```

```
public boolean doInference() throws DIGReasonerException {
   if(reasoner.isConnected()) {
      DIGReasonerIdentity reasonerIdentity = reasoner.getIdentity();
      System.out.print("Connected to " +
                       reasonerIdentity.getName() +
                       ". Classifying ...");
      reasoner.classifyTaxonomy(null);
      System.out.println("  done.\n");
      return true;
   } else return false;
}
```

WSDL-S ermöglicht die Anreicherung der Serviceoperationen mit semantischen Informationen. Dabei erweitert der EMEO-Layer das Verbinden der Eingabe (`I`), Ausgabe (`O`), Vor- (`P`) und Nachbedingung (`E`) mit mehreren unterschiedlichen Konzepten. Somit besteht, bei der Ermittlung der Äquivalenz, die Notwendigkeit, alle zugeordneten Konzepte berücksichtigen zu müssen. Ein zugeordnetes Konzept wird im Folgenden als Fragment bezeichnet. Die Grafiken im Anhang L.4 auf Seite 196 untergliedern die Ontologie in zwei Bereiche (siehe Abbildung L.1) und illustrieren die Inferenz via Inferenzmaschine RACER (siehe Abbildung L.2), visualisiert in Protégé. Der linke Bereich zeigt die modellierte Ontologie O_1, die in Kapitel 7.2 formal beschrieben wird. Der rechte Bereich zeigt die, durch Inferenz gefolgerte Ontologie O_2. Abbildung L.2 stellt alle Aktionen dar, um O_1 auf O_2 abzubilden. Generell sind beim Vorgang der Inferenz drei Aktionen (*Add()*, *Move()*, *Remove()*) auf Konzept-Ebene möglich. *Move()* stellt die Konzept-Hierarchie eines Konzepts um, *Add()* ergänzt ein bereits bestehendes Konzept an einer anderen Stelle und *Remove()* entfernt ein bestehendes Konzept aus der Konzept-Hierarchie. Die Methode *detectEquivalence()* der Klasse *RankService()* extrahiert die einzelnen Fragmente und bewertet in der Form *inferredEquivalentClasses()* mit 1.0, *inferredSuperClasses()* und *inferredSubclasses()* mit 0.5. Die Methode *calcPercent()* errechnet das prozentuale Verhältnis zwischen *übereinstimmende Fragmente* und *Anzahl der Fragmente*.

```
   public void detectEquivalence(final String sIOPE, HashMap mapService,
               HashMap mapServiceESR) throws DIGReasonerException {
      this.mapServiceESR = mapServiceESR;
      this.sIOPE = sIOPE;

      System.out.println(sIOPE);
      String sInput = (String) mapService.get(sIOPE);
      String sSplit[] = sInput.split(DELIMITER);

      setServiceFragmentCount(sSplit.length);
      setServiceFragment(0);
```

```
    for (int i=0 ; i < sSplit.length; i++ ) {
       OWLNamedClass namedClass             =
          model.getOWLNamedClass(sSplit[i]);

       Collection inferredSubclasses        =
          namedClass.getInferredSubclasses();
       Collection inferredEquivalentClasses =
          namedClass.getInferredEquivalentClasses();
       Collection inferredSuperClasses      =
          namedClass.getInferredSuperclasses();

       setFragmentRanked(false);
       printCollection(inferredEquivalentClasses, sSplit[i],
          "inferredEquivalentClasses", 1);
       printCollection(inferredSuperClasses,      sSplit[i],
          "inferredSuperClasses     ", 0.5);
       printCollection(inferredSubclasses,        sSplit[i],
          "inferredSubclasses       ", 0.5);

       System.out.print("\n");
    }
    calcPercent(sIOPE);
}
```

Die private Methode *rankFrag()* prüft via Methode *isFragmentRanked()* auf Doppelbewertungen der Fragmente, bewertet (im negativen Fall) das Fragment und übergibt das Ergebnis an die Console.

```
private void rankFrag(double ranking, String curClass, String frag) {
    String sInputESR   = (String) this.mapServiceESR.get(sIOPE);
    String sSplitESR[] = sInputESR.split(DELIMITER);

    if ( ! isFragmentRanked() ) {
       for (int j=0 ; j < sSplitESR.length; j++ ){
       if (curClass.equals(sSplitESR[j]) ||
          frag.equals(sSplitESR[j])) {
          addRank(ranking);
          System.out.print("(ADD " + ranking + ")");

          setFragmentRanked(true);
          j = sSplitESR.length;
       }
    }
```

Die Methode *calcPercent()* errechnet das prozentuale Verhältnis zwischen *übereinstimmende Fragmente* und *Anzahl der Fragmente*, formatiert und speichert das Ergebnis in einer HashMap.

```
private void calcPercent(String sIOPE) {
   String sFrag      = String.valueOf(getRankServiceFragment());
   String sFragCount = String.valueOf(getServiceFragmentCount());
   String sKey       = (String) mapServiceESR.get(SERVICE) + "." +
                       (String) mapServiceESR.get(OP);

   Integer iErgebnis = new Integer ( (int)
                       Math.round(( Double.parseDouble(sFrag) /
                       Double.parseDouble(sFragCount) ) * 100) );

   rankServiceOperation.put(sIOPE, sFrag + DELIMITER +
                            sFragCount + DELIMITER + iErgebnis);
   this.mapServiceESR.put(RANKING, rankServiceOperation);

      if (rankedServices.get(sKey) == null) {
         rankedServices.put( sKey, "0");
      }

      double dRanking =  Double.parseDouble( (String)
                         rankedServices.get(sKey) ) +
                         Double.parseDouble( (String)
                         iErgebnis.toString() );
      String sRanking = myFormatter.format(dRanking);
      rankedServices.put( sKey, sRanking);
   }
```

Die folgende Darstellung zeigt die Ausgabe (Auszug) durch den SSF auf der Console. Zeile (01) dokumentiert die Abbildung der Serviceoperation *MASPlus.getSollTermin()* auf die Serviceoperation *BZM.ermittleSollTermin()*. Die Zeile (02) dokumentiert die Prüfung semantischer Konzepte der Eingabeparameter (INPUT). Die Zeile (03) dokumentiert drei quantitativ zugewiesene Fragmente durch die Ziffer 3. equalClasses und ermittelt, aus den drei Fragmenten, die Existenz des Fragments DAISachnummer (falls vorhanden). Die Zeichenkette (ADD 1.0) dokumentiert die Existenz eines gleichen Fragments, bewertet mit 1.0. Die Zeilen (04-06) dokumentieren die Aufrufe der Methoden *inferredEquivalentClasses()*, *inferredSuperClasses()* und *inferredSubclasses()* mit dem Ziel der Ermittlung der Existenz äquivalenter Klassen, Oberklassen oder Unterklassen. Äquivalente und durch Schließen ermittelte, äquivalente Klassen werden mit 1.0, Oberklassen sowie Unterklassen mit 0.5 bewertet. Die Zeile (07) fasst die Ergebnisse der Abbildung Serviceoperation *MASPlus.getSollTermin()* auf die Serviceoperation *MASPlus.getSollTermin()* zusammen: *Anzahl übereinstimmende Fragmente (a)* ; *Anzahl Fragmente (b)* ; Quotient aus der Division (a)/(b) in Prozent. Die Zeilen (08-15) illustrieren einen Vorschlag, in Form einer Auswahlliste, aufsteigend sortiert nach der Bewertung. Somit wurde die Identifizierung äquivalenter Services durch semantische semiautomatische Unterstützung prototypisch belegt. Dies stellt eine signifikante Verbesserung der in Kapitel 5 auf Seite 81 vorgestellten, quasi-statischen Bindung des Prozessschrittes an den Service dar.

```
01  MASPlus.getSollTermin ==> BZM.ermittleSollTermin
01a
02  INPUT
03  equalClasses (candidates) of DAISachnummer(s):  3 (ADD 1.0)
04  inferredEquivalentClasses of DAISachnummer(s):  0
05  inferredSuperClasses      of DAISachnummer(s):  1
06  inferredSubclasses        of DAISachnummer(s):  0
...
07  I=3.0;3.0;100%  O=1.0;1.0;100%  P=2.0;3.0;67%  E=1.0;1.0;100%
...
08  Äquivalente Serviceoperationen (Kandidaten):
09  ============================================
09a
10  Bewertung   Serviceoperation
11  --------------------------------------------
12  0367        BZM.ermittleSollTermin
13  0100        CrtPlusService.getCustomer
14  0017        BZM.uebergebeDaten
15  0000        CrtService.getPartOwner
```

Die folgende Darstellung illustriert einen Vorschlag zur Abbildung der Schnittstellen (durch den SSF) bei den Eingabe- (I) und Ausgabeparametern (O) sowie die partielle Abbildung der Konzepte bei Vorbedingung (P) und Nachbedingung (E). Der Vorschlag zur Abbildung wird exemplarisch an der Serviceoperation BZM.ermittleSollTermin aufgezeigt, da die Serviceoperation mit einem Wert von 367 am Höchsten bewertet wurde. Die Zeilen (01) und (06) dokumentieren den Beginn sowie das Ende der Abbildung der Serviceoperation MASPlus.getSollTermin auf BZM.ermittleSollTermin. Abbildungen der Konzepte werden durch den doppelten Pfeil ====>, Abbildungsvorschläge durch den einfachen Pfeil ----> illustriert. Die Zeile (02) zeigt die Abbildung der Eingabeparameter (I) MASPlus.getSollTermin.DAISachnummer auf BZM.ermittleSollTermin.DAISachnummer, den Abbildungsvorschlag MASPlus.getSollTermin.Menge auf BZM.ermittleSollTermin.Amount (in Zeile (03)) und die Abbildung MASPlus.getSollTermin.DAILieferTermin auf BZM.ermittleSollTermin.DAILieferTermin (in Zeile (04)). Die Zeile (05) zeigt die Abbildung des Ausgabeparameters (O) MASPlus.getSollTermin.DAISollTermin auf BZM.ermittleSollTermin.DAISollDatum. Die Zeilen (06) und (11) dokumentieren den Beginn sowie das Ende der Abbildung semantischer Konzepte bei Vor- (P) bzw. Nachbedingung (E) der beiden Serviceoperationen MASPlus.getSollTermin und BZM.ermittleSollTermin. Die Zeile (07) zeigt die Abbildung zugewiesener Konzepte der Vorbedingung (P) MengePlausibel auf MengePlausibel, die partielle Abbildung Gear auf Aggregate (in Zeile (08)) sowie SLK auf Roadster (in Zeile (09)). Die Zeile (10) zeigt die Abbildung des zugewiesenen Konzeptes der Nachbedingung (E) DAISollTerminVerfuegbar auf DAISollTerminVerfuegbar.

```
01  ----- Vorschlag zur Abbildung der Schnittstellen -------------
02  I: | DAISachnummer ===> DAISachnummer |
03     | Menge ===> Amount |
04     | DAILiefertermin ===> DAILiefertermin |
05  O: | DAISollTermin ===> DAISollTermin |
06  ----- Abbildung der Konzepte --------------------------------
07  P: | MengePlausibel ===> AmountPlausible |
08     | Gear ---> Aggregate |
09     | SLK ---> Roadster |
10  E: | DAISollTerminVerfuegbar ===> DAISollTerminVerfuegbar |
11 ----------------------------------------------------------------
```

Die Abbildung und partielle Abbildung zugewiesener Konzepte der Vor- (P) und Nachbedingung (E) unterstützt die Selektion äquivalenter Services zur Entwicklungszeit. Der Vorschlag zur Abbildung der Schnittstellen durch die semantische semiautomatische Unterstützung wurde prototypisch belegt und stellt eine signifikante Verbesserung der in Kapitel 3.2.2 auf Seite 66 vorgestellten, manuellen Abbildung der Schnittstellen bzw. Geschäftsobjekte dar. Die komplette Ausgabe der Console wird im Anhang M auf Seite 197 dargestellt.

7.5 Mediator im Enterprise Service Bus

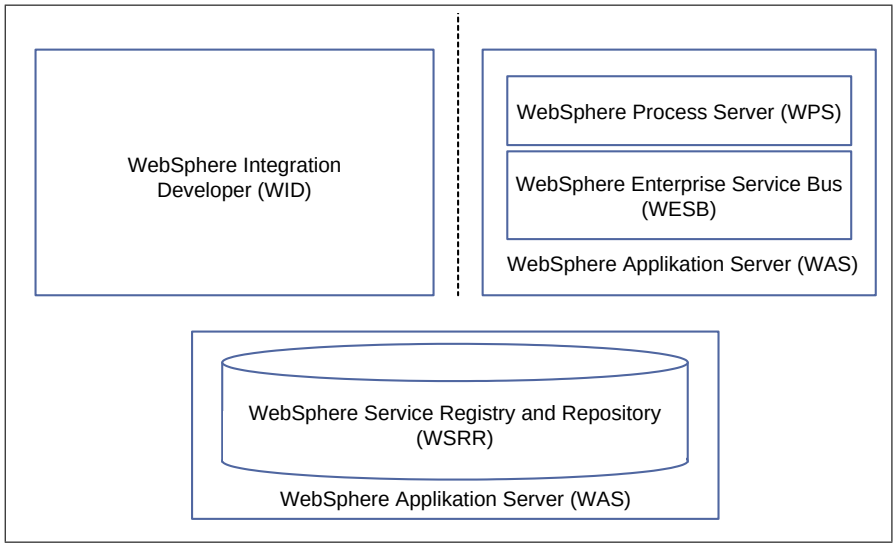

Abbildung 7.5: Produkt-Suite im Überblick

Das Verhaltensmuster Mediation (siehe Kapitel 2.9 auf Seite 38) wurde bereits dargestellt sowie Prinzipien der Objekt-orientierten Software-Entwicklung auf plattformneutraler Ebene der Services diskutiert. Die tragenden, allgemeingültigen Konzepte einer SOA, der Enterprise Service Bus und das Enterprise Service Repository wurden bereits in den Kapiteln 2.1.7 bis 2.1.8 auf den Seiten 32 und 34 illustriert. In diesem Kapitel wird der EMEO-Layer partiell mit den heute am Markt verfügbaren Werkzeugen exemplarisch implementiert. Dabei wird die Vorgehensweise zur Entwicklungszeit und das Verhalten in der Ablaufumgebung aufgezeigt. Die Abbildung 7.5 illustriert die genutzte Infrastruktur im Labor bei der Daimler AG im Mercedes-Benz Werk Sindelfingen. Der WebSphere Integration Developer (WID) unterstützt den Software-Entwicklungsprozess durch die grafische Modellierung von Geschäftsprozessen, die Möglichkeit der technischen Anreicherung des Modells und die Generierung der Java-Klassen. Der WebSphere Process Server (WPS) interpretiert die generierten Java-Klassen und stellt somit die Ablaufumgebung der modellierten Geschäftsprozesse dar. Technologisch betrachtet basiert der WPS auf offenen Standards, die ihn in die Lage versetzen, Web Services sowie proprietäre Formate (durch Adapter) in die Infrastruktur einzubinden. Der WPS stellt sicher, dass Geschäftsprozesse transaktional (ACID) durchgeführt werden und die Prozess-Historie persistiert wird. Der WebSphere Enterprise Service Bus (WESB) ist eine Komponente zur Untersützung (Formatierung der Nachrichten via HTTP/SOAP sowie Abbilden der Geschäftsobjekte) der WPS-Funktionalität. Das WebSphere Service Registry und Repository (WSRR) ist eine Anwendung, die Informationen über die Services zur Entwicklungszeit und in der Ablaufumgebung bereitstellt. Der WebSphere Applikation Server (WAS) ist die Laufzeitumgebung der Produkte WPS,

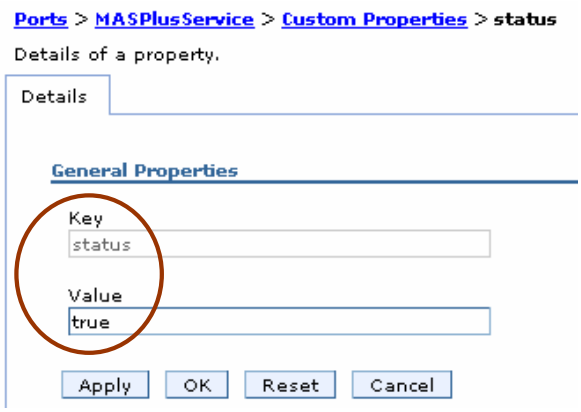

Abbildung 7.6: Benutzerdefinierte Eigenschaft *status=true*

ESB und WSRR. Der WAS stellt eine auf Java basierende Messaging-Engine zur Verfügung, die als Kommunikationsinfrastruktur zwischen den Komponenten dient. Um einen dynamischen Lookup mit dem Ziel der Mediation zwischen zwei Services realisieren zu können, müssen beide Services im WSRR publiziert und durch eine benutzerdefinierte Eigenschaft *(engl. Custom Property)* ergänzt werden. Im Folgenden wird die Publizierung der WSDL-Datei `MASPlus.wsdl` (siehe Anhang P.1 auf Seite 219) und die Erweiterung der benutzerdefinierten Eigenschaft `status` im WSRR beschrieben.

Die Voraussetzung für den Zugriff via Web-Oberfläche ist ein bereits gestarteter WAS. Nach erfolgreicher Authentifizierung am WSRR via URL `http://localhost:9080/ServiceRegistry/` kann der Service `MASPlus.wsdl` registriert werden (siehe Abbildung im Anhang O.1 auf Seite 213). Über *Service Documents → Load Document*, der Schaltfläche `Browse`, dem Dokument-Typ *(engl. document type)*, der Beschreibung *(engl. document description)* und der Versionsnummer *(engl. document version)* wird durch die Bestätigung der Schaltfläche `OK` der Service `MASPlus.wsdl` in das WSRR geladen (siehe Abbildung O.2 auf Seite 213). Während des Speichervorgangs wird die WSDL-Datei nach `port types`, `bindings`, `services` und `custom properties` strukturiert. Die Abbildung in Anhang O.3 auf Seite 214 illustriert einen Teil der Strukturierung durch das WSRR.

Über *Service Metadata → WSDL → Ports → MASPlusService → Custom Properties*, der Schaltfläche `New`, dem Schlüsselwort *(engl. Key)* und dem Wert *(engl. Value)* wird durch die Bestätigung der Schaltfläche `OK` dem Service `MASPlus.wsdl` eine neue Eigenschaft zugewiesen. Diese Eigenschaft steuert die Mediation via Lookup-Komponente abhängig vom Wert `true` bzw. `false`. Abbildung 7.6 illustriert das Schlüsselwort `status` mit dem Wert `true`, verwaltet durch das WSRR. Die Abbildung 7.7 wurde aus drei verschiedenen Bildausschnitten zu einem neuen Gesamtbild zusammengesetzt (Fotomontage) und illustriert drei Bereiche des WebSphere Integration Developers. Der linke Bereich *Business Integration* zeigt die Projektstruktur in einer hierarchischen Darstellung. Das Modul *BZMSimulator*, das

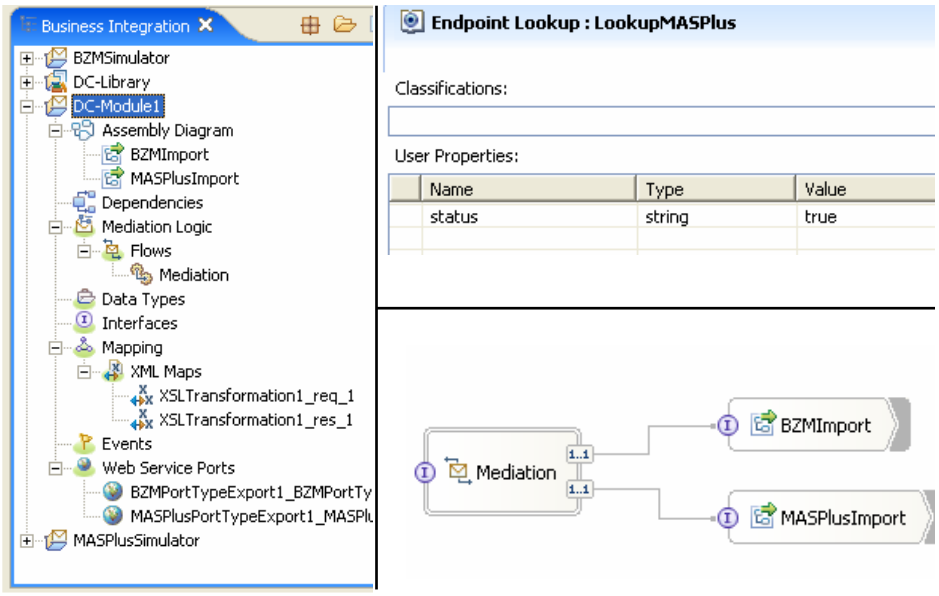

Abbildung 7.7: WebSphere Integration Developer

Modul *MASPlusSimulator* und die Bibliothek *DC-Library* verwalten die Informationen der Services `BZM` und `MASPlus`. Das Modul *DC-Module1* wird in der Abbildung komplett illustriert und verwaltet alle Informationen über die Mediation. *DC-Module*1 → *Mediation Logic* → *Flows* → *Mediation* verwaltet die grafische Darstellung der Mediation in Richtung Anfrage *(engl. request)* (siehe Abbildung 7.8) und Antwort *(engl. response)* (siehe Anhang O.4 auf Seite 214). *DC-Module*1 → *Mapping* → *XML Maps* verwaltet beide Abbildungen der Geschäftsobjekte `MASPlus` und `BZM`. Die Abbildung in Anhang O.5 auf Seite 215 illustriert die Abbildung der Eingabeparameter des Service `BZM` auf Service Message Objekts im ESB (siehe Kapitel 2.1.7 auf Seite 32). Der obere rechte Bereich *Endpoint Lookup* illustriert die im WSRR ergänzend definierte Eigenschaft `status` und den zugewiesenen Wert `true`. Unterstützt durch die grafische Oberfläche im WID ist keine Programmierung notwendig, um den zugewiesenen Wert zur Steuerung abzufragen. Der untere rechte Bereich illustriert alle Verbindungen des Moduls *DC-Module1* in Form eines Verbindungsdiagramm *(engl. assembly-diagrams)* durch das Aufzeigen aller Referenzen auf externe Ressourcen, wie das der beiden Module *BZMSimulator* und *MASPlusSimulator*. Die Erweiterung der beiden Module *BZM* und *MASPlus* durch den Term *Simulator* dokumentiert die Simulation der beiden Service-Provider durch eine Mediation im WID.

Die Grafik 7.8 illustriert im oberen Bereich das Abbilden der Operation `getTermin` auf die beiden Operationen `ermittleSollTermin` und `getSollTermin` der beiden Service-Provider `BZM` und `MASPlus`. Der untere Bereich der Grafik 7.8 illustriert den modellierten Prozessfluss (Mediation) in Form einer Anfrage an den Service-Provider und stellt somit die Implementierung der in Kapitel 6.1 auf Seite

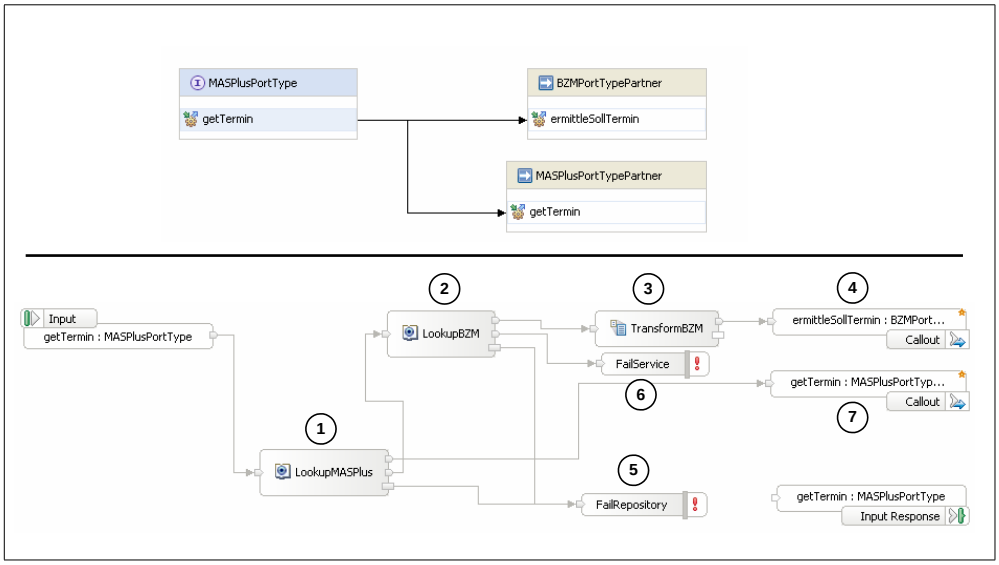

Abbildung 7.8: Prozessfluss (Mediation) Anfrage

86 angenommenen Erweiterung dar. Im Folgenden wird die Vorgehensweise zur Erstellung der Mediation im WID zur Entwicklungszeit beschrieben. (Die Grafik 7.8 wird im Anhang O.10 auf Seite 218 vergrößert dargestellt.)

1. Via drag-and-drop *(dt. (rüber)ziehen und ablegen)* wird eine *Lookup-Komponente* erstellt und durch die Eingabe `LookupMASPlus` eindeutig bezeichnet. Sie besteht aus einem Eingang (E_1) und drei Ausgängen (A_1, A_2, A_3), ermöglicht das Auslesen der Wertzuweisung (`true`) (der benutzerdefinierten Eigenschaft `status` des Service `MASPlus` aus dem WSRR) und kann ggf. durch die Programmiersprache Java funktional erweitert werden. Die Funktion *Vorlaufrechnung Montage* wird durch einen gerichteten Pfeil mit E_1 verbunden. A_1 repräsentiert ein positives Ergebnis (`true`), A_2 repräsentiert ein negatives Ergebnis (`not true`) und A_3 repräsentiert den Fehlerfall der benutzerdefinierten Eigenschaft `status`.

2. Somit spaltet die *Lookup-Komponente* den Serviceaufruf in drei voneinander unabhängige Pfade ($P1_1$, $P1_2$, $P1_3$). Via drag-and-drop wird die zweite *Lookup-Komponente* erstellt (produktspezifische Abweichung zur in Kapitel 6.1 auf Seite 86 angenommenen Vorgehensweise) und durch die Eingabe `LookupBZM` eindeutig bezeichnet sowie das Auslesen der benutzerdefinierten Eigenschaft `status` des Service `BZM` aus dem WSRR ermöglicht. Analog zur ersten *Lookup-Komponente* `LookupMASPlus` besteht die zweite aus einem Eingang und drei Ausgängen und somit aus drei voneinander unabhängigen Pfade ($P2_1$, $P2_2$, $P2_3$).

3. Die *Transformation-Komponente* wird via drag-und-drop erstellt und durch die Eingabe `TransformBZM` eindeutig bezeichnet. Sie bildet die Eingabeparameter der Funktion *Vorlaufrechnung Montage* auf die Eingabeparameter des Service `BZM` ab und wird mit $P2_1$ verbunden.

4. Die Operation `ermittleSolltermin` des Service BZM wird mit der *Transformation-Komponente* verbunden und ruft in der Ablaufumgebung den Service-Provider auf.

5. $P1_3$ und $P2_3$ repräsentieren den Fehlerfall und werden direkt mit der *Fehler-Komponente* `FailRepository` verbunden. Diese Komponente gibt beim Eintreten einer der beiden Ereignisse bzw. Pfade eine benutzerdefinierte Fehler-Meldung `WSRR nicht verfügbar` auf der Console aus.

6. $P2_2$ repräsentiert keinen verfügbaren Service und wird direkt mit der *Fehler-Komponente* `FailService` verbunden. Diese Komponente gibt beim Eintreten des Ereignisses bzw. des Pfades eine benutzerdefinierte Fehler-Meldung `Kein Service verfügbar` auf der Console aus.

7. A_1 repräsentiert das positives Ergebnis der Abfrage nach der benutzerdefinierten Eigenschaft `status`, wird direkt mit der Operation `getSollTermin` des Service `MASPlus` verbunden und ruft in der Ablaufumgebung den Service-Provider auf.

Die grafische Darstellung der Mediation in Form einer Antwort vom Service-Provider wird im Anhang O.4 auf Seite 214 illustriert und schließt die Modellierung der Mediation komplett ab. Die beiden Service-Provider `MASPlus` und `BZM` sind IT-Systeme im Intranet der Daimler AG und werden vom European-Data-Center (EDC) betrieben. Das EDC agiert als Cost-Center (eigenständige Unternehmenseinheit ohne Gewinnermittlung) und ist für den ausfallsicheren Betrieb der IT-Systeme verantwortlich. Der ausfallsicherere Betrieb wird u.a. durch hohe Sicherheitsanforderungen gewährleistet, womit ein Zugriff aus dem Labor nicht möglich ist. Folglich wurden die beiden Service-Provider `MASPlus` und `BZM` im WID simuliert. Beide Service-Provider haben drei Eingabeparameter und einen Ausgabeparameter (siehe Kapitel 4.1.2 auf Seite 78), wobei jeder der beiden Service-Provider ein Datenfeld vom Typ `xsd:date` als Eingabe- und Ausgabeparameter besitzt. Bei der Simulation der beiden Service-Provider wurde die einfachste Ausprägung der Mediation genutzt, indem der Eingabeparameter auf den Ausgabeparameter abgebildet wurde (pro Service). Im Anhang O.9 auf Seite 217 wird die Simulation der beiden Service-Provider illustriert.

Die Abbildung 7.9 illustriert im linken Bereich Ablauf-1 und Ablauf-2 der Mediation; Ablauf-1 mit verfügbarem Service `MASPlus` (`status=true`) und Ablauf-2 mit unverfügbarem Service `MASPlus` (`status=false`). Im rechten Bereich werden Eigenschaften wie Modulname `DC-Module1`, Komponente `Mediation`, Schnittstelle `MASPlus` und Name der Operation `getSollTermin` sowie der Ausgabeparameter

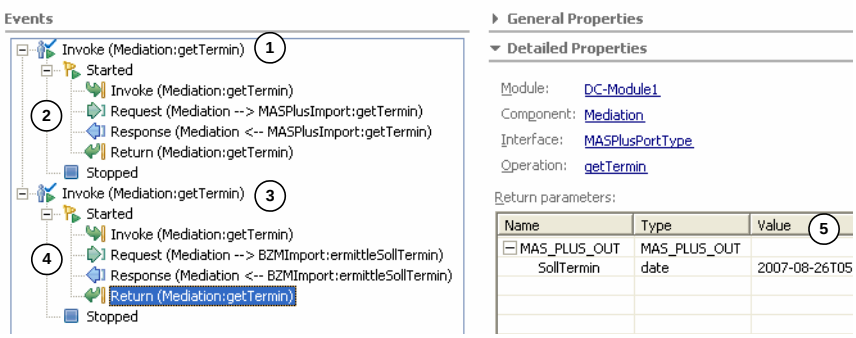

Abbildung 7.9: Ablaufumgebung Mediation *status=false*

`SollTermin` des Ablauf-2 abgebildet. Im Folgenden wird das Verhalten der Mediation in der Ablaufumgebung beschrieben (Vgl. Kapitel 6.5 auf Seite 93).

1. Die Mediation wird über die Oberfläche, durch die Eingabe der Parameter `Sachnummer=3500446217`, `Menge=30` und `Liefertermin=2007-08-27` sowie der Bestätigung der Schaltfläche `Continue`, gestartet. Vgl. Anhang O.7 auf Seite 216.

2. Aufgrund der benutzerdefinierten Eigenschaft und dem gesetzten Wert (`status=true`) im WSRR wird der Service `MASPlus` aufgerufen. Die Simulation des Service-Providers (siehe Anhang O.9 auf Seite 217) wird gestartet und durch die Rückgabe des Ausgabeparameters `SollTermin=2007-08-26` beendet. Vgl. Anhang O.8 auf Seite 216.

3. Der Wert der benutzerdefinierten Eigenschaft im WSRR wird auf `false` gesetzt (vgl. Anhang O.6 auf Seite 215) und die Mediation über die Oberfläche erneut (inkl. Eingabe aller Parameter) gestartet.

4. Aufgrund der benutzerdefinierten Eigenschaft und dem gesetzten Wert (`status=false`) im WSRR wird der Service `BZM` aufgerufen. Die Simulation des Service-Providers (siehe Anhang O.9 auf Seite 217) wird gestartet.

5. Durch die Rückgabe des Ausgabeparameters `SollTermin=2007-08-26` wird die Mediation beendet.

Im WSRR wurde jedem der beiden Services `MASPlus` und `BZM` die benutzerdefinierte Eigenschaft `status` zugewiesen. Das in Abbildung 7.9 illustrierte Verhalten in der Ablaufumgebung beschränkt sich auf die Veränderung (`status=false`) der dem Service `MASPlus` zugewiesene benutzerdefinierte Eigenschaft. Eine Veränderung der dem Service `BZM` zugewiesenen Eigenschaft hat beim Eintritt der Ereignisse `MASPlus.status=false` und `BZM.status=false` eine Ausgabe auf der Console (Fehler-Meldung `Kein Service verfügbar`) und keinen Serviceaufruf zur Folge. *Der EMEO-Layer wurde mit Software-Architekten der IBM im Detail diskutiert. Dabei wurde die Absicht signalisiert, das Konzept in die nächste Release-Planung aufzunehmen.*

7.6 Abgeleitete generische Vorgehensweise TD++

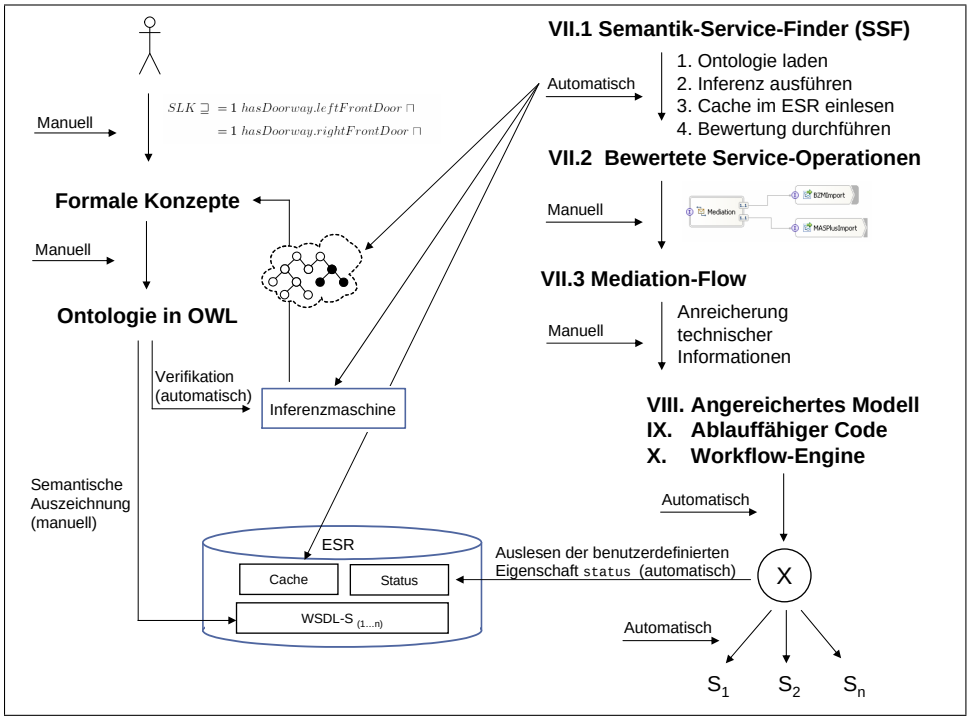

Abbildung 7.10: Generische Vorgehensweise (Top-Down++)

TD++ (vgl. Abbildung 7.10) ist ein Akronym gebildet aus dem englischen Term „Top-Down“, wobei das erste Plus (+) die Durchgängigkeit von der Modellierung bis zur Codegenerierung symbolisiert. Das zweite Plus (+) symbolisiert die im Folgenden beschriebenen Erweiterungen der bereits vorgestellten, abgeleiteten, generischen Vorgehensweise TD+ (vgl. Kapitel 3.3 auf Seite 68).

In der dortigen Abbildung 3.9 auf Seite 68 beschreibt Schritt I. die Analysephase, die Schritte II. – V. die Designphase, die Schritte VI. – VIII. die Entwicklungsphase sowie die Schritte IX. und X. das Deployment. Die im Folgenden dargestellte Erweiterung TD++ sieht keine Ergänzungen in den Schritten *I. Prozessbeschreibung* bis *VI. Dateisystem* sowie *VIII. Angereichertes Modell* bis *X. Workflow-Engine* vor. Als TD++ bezeichneten Erweiterungen betreffen ausschließlich Schritt *VII. Entwicklungsumgebung* und das Verhalten in der Ablaufumgebung.

Abbildung 7.10 illustriert auf der linken Seite die manuelle Erstellung des minimal angenommenen, deklarativen Wissens einer Personengruppe in Form **formaler Konzepte**, die manuelle Erstellung einer **Ontologie in OWL**, die iterative Vorgehensweise zur **Validierung** der Konzepte durch eine Inferenzmaschine (vgl. Kapitel 6.6 auf Seite 94) sowie die manuelle **semantische Auszeichnung** der Konzepte durch WSDL-S. Im unteren Bereich der Abbildung 7.10 wird das Enterprise Service

Repository (ESR) illustriert. Das ESR generiert eine indizierte Liste aller verfügbaren Services (Cache), verwaltet die mit semantischer Information angereicherten Services (WSDL-S) und die benutzerdefinierte Eigenschaft `status`. Der rechte Bereich der Abbildung 7.10 illustriert die im Folgenden beschriebenen Erweiterungen.

VII.1 Semantik-Service-Finder (SSF). Der entwickelte Algorithmus SSF lädt in der ersten Phase die Ontologie in den Arbeitsspeicher der Inferenzmaschine. Nach durchgeführter Inferenz in der zweiten Phase wird in der dritten Phase die indizierte Liste aller verfügbaren Services aus dem Cache des Enterprise Service Repositories gelesen. In der abschließenden vierten Phase werden die Serviceoperationen bewertet und eine Liste äquivalenter Services erstellt (vgl. Kapitel 6.4 auf Seite 91).

VII.2 Bewertete Service-Operation. Der SSF übergibt die Liste äquivalenter Services, aufsteigend sortiert nach der höchsten Bewertung, zur Verifizierung und semiautomatischer Auswahl an den Entwickler. Der Entwickler wählt nach Rücksprache mit dem Service-Portfolio-Management (siehe Kapitel 2.1.9 auf Seite 35) einen oder mehrere äquivalente Services aus und bildet die Schnittstelle ab.

VII.3 Mediation-Flow. Die Mediation spaltet den Serviceaufruf in zwei oder mehrere voneinander unabhängige Pfade. Der Entwickler ergänzt die Pfade um die ermittelten äquivalenten Services zu einem Flow. Nach dem Anreichern technischer Informationen wird aus dem angereicherten Modell ablauffähiger Code (Java, EJB) generiert und in die Workflow-Engine bzw. in den Enterprise Service Bus ausgeliefert (vgl. Kapitel 3.3 auf Seite 68).

X. Workflow-Engine. In der Ablaufumgebung ermittelt der Enterprise Service Bus automatisch den Service-Endpunkt sowie den Wert x der benutzerdefinierten Eigenschaft `status` aus dem Enterprise Service Repository. Abhängig vom Wert x ruft der Enterprise Service Bus automatisch Service S_1, S_2 oder S_n auf und gibt die Antwort des aufgerufenen Service zurück (vgl. Kapitel 7.5 auf Seite 124).

7.7 Nachweis verbesserter Verfügbarkeit

Definition 5 *Unter **maximaler Ausfallzeit** versteht man die Zeit bis zur Wiederherstellung der Grundfunktionalität aus Anwendersicht [Daimler AG].*

Definition 6 ***Verfügbarkeit V** ist die Wahrscheinlichkeit, ein System zu einem vorgegebenen Zeitpunkt t in einem funktionsfähigen Zustand anzutreffen [DIN 40042].*

Hochverfügbarkeit wird durch Cluster auf Software-Ebene im WAS [84] und auf Hardware-Ebene durch zSeries [69] realisiert. Es ist nicht davon auszugehen, dass die komplette Infrastruktur bei der Daimler AG hochverfügbar ist. In einer SOA ist bei der Auswahl der Service-Provider in der internen Infrastruktur auf die Verfügbarkeit zu achten. Generell wird zwischen *geplanter Ausfallzeit*, wie bspw. Einspielen von

Patches, Auf- bzw. Umrüsten von Hardware, Software-Update, Release-Update etc. und *ungeplanter Ausfallzeit* wie bspw. Hardware- sowie Software-Fehler, menschliches Fehlverhalten, Netzwerkprobleme etc. unterschieden. Messungen bezogen auf menschliches Fehlverhalten werden durch den Betriebsrat ausgeschlossen. Die Eintrittswahrscheinlichkeit der voneinander unabhängigen Ereignisse, welche die folgende IEEE-Studie aufzeigt, ist mit der Infrastruktur bei der Daimler AG vergleichbar und wird in Tabelle 7.2 dargestellt [82].

Fehlerursache	$\mathbf{P}(A_i)$
Geplante Ausfallzeit	**0.30**
Einspielen BS-Patch	(0.10)
Auf-/Umrüsten Hardware	(0.05)
Software-Updates	(0.15)
Ungeplante Ausfallzeit	**0.70**
Menschliches Fehlverhalten	(0.15)
Software-Fehler	(0.40)
Hardware-Fehler	(0.10)
Ablaufumgebung	(0.05)

Tabelle 7.2: Fehlerursachen geplanter und ungeplanter Ausfallzeiten

Die IT-Systeme in der Daimler AG werden in vier Verfügbarkeitsklassen eingeteilt (siehe Tabelle 7.3). Die hier abgebildete Tabelle stellt alle Klassen und ihre maximal zu akzeptierenden Ausfallzeiten dar. Die maximale Ausfallzeit von *Null* Stunden/Jahr ist ein geforderter imaginärer Wert und nicht erreichbar. Bspw. wird bei der Daimler AG durch den Einsatz von z/OS Sysplex eine Verfügbarkeit von 99,999%[1] erreicht. Der Ausfall eines IT-Systems der Verfügbarkeitsklasse XH impliziert einen Stopp der Produktion und führt somit zu direkten Umsatzeinbußen.

Code	XH	H	S	N
Bezeichnung	höchstverfügbar	hochverf.	schnellverf.	normalverf.
Max. Ausfallzeit	0	1 Std.	8 Std.	72 Std.

Tabelle 7.3: Verfügbarkeitsklassen

[1] Dieser Wert entspricht rund 5 Minuten Ausfallzeit pro Jahr.

Errechenbare Verfügbarkeit

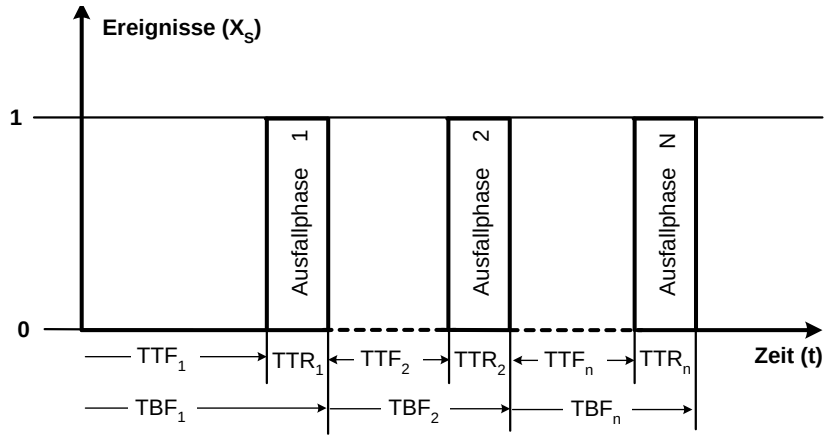

Abbildung 7.11: Ausfall- und Reparaturzeiten reparierbarer Systeme

Eine Verfügbarkeit der Services von 100 % in einer gewachsenen Infrastruktur ist nicht realisierbar. Gründe hierfür sind u.a. die wachsende Komplexität der Systemlandschaften sowie die permanante Weiterentwicklung der Technologien. Die Durchschnittskosten für eine Stunde Unverfügbarkeit werden auf bis zu 150.000 $ im Medienumfeld und auf bis zu 6.500.000 $ im Finanzwesen beziffert [129]. Ziel eines Unternehmens ist es, die Unverfügbarkeit zu minimieren und *ungeplante Ausfallzeiten* zu eliminieren. In der Praxis werden die *geplanten Ausfallzeiten* auf Feiertage oder auf das Wochenende terminiert. Dringende *geplante Ausfallzeit* wie beispielsweise ein Sicherheitsupdate, nicht während des laufenden Betriebes durchgeführt werden kann sowie ein Update der Applikation oder Datenbank, die nach erfolgreichem Update ausführliche, zeitintensive Test über mehrere Stunden benötigen, sind sehr zeitnah zu realisieren. Der sehr praxisnahe Ansatz, die *geplanten Ausfallzeiten* auf die arbeitsfreie Zeit zu legen, ist unbefriedigend, da beispielsweise in der Automobilindustrie auch an Feiertagen und Wochenenden produziert wird. Generell sind Abweichungen von Dienstleistungsverträgen *(engl. Service Level Agreements (SLAs))* zu messen. Im Folgenden wird die mathematische Grundlage, zur Ermittlung des evtl. vorhandenen Einsparungspotentials, kurz erläutert. Die Eintrittswahrscheinlichkeiten der Fehlerursachen bezogen auf die Ausfallzeit sind in den meisten Fällen voneinander unabhängig, somit gilt:

$$\sum_{i=1}^{n} P(A_i) = 1$$

In der abgebildeten Grafik 7.11 auf Seite 133 werden Zeitspanne (Time To Failure FFT), Ausfalldauer (Time To Repair TTR) und Zeitdauer (Time Between Failure TBT) zwischen den Ausfällen dargestellt. Es können genau zwei Ereignisse, System ist intakt ($X_S = 0$) und System ist defekt ($X_S = 1$), eintreten. Die Ausfallphase

beginnt mit dem Zeitpunkt des Systemausfalls und beinhaltet alle Reparaturzeiten, die notwendig sind, das System wieder zu betreiben. Im Folgenden wird der Zusammenhang zwischen Ausfall- und Reparaturzeit definiert. Unverfügbarkeit U ist das Komplement der Verfügbarkeit V zu 1, somit gilt:

$$U = 1 - V \tag{7.40}$$

Für die mittlere ausfallfreie Zeitspanne (Mean Time To Failure MTTF), mittlere Ausfalldauer (Mean Time To Repair MTTR) und mittlere Zeitdauer zwischen den Ausfällen (Mean Time Between Failure (MTBF) gilt:

$$MTBF = MTTF + MTTR \tag{7.41}$$

Für die Ausfallrate $\lambda(\lambda = 1/MTTF)$, die Reparaturrate $\mu(\mu = 1/MTTR)$ und die mittlere Fehlerhäufigkeit bzw. Fehlerrate ν gilt:

$$\nu = \lambda * \mu/(\lambda + \mu) \tag{7.42}$$

Mit Hilfe der mittleren Zeit bis zum Ausfall (MTTF) sowie der mittleren Zyklusdauer aus Betriebsintervall und der Ausfallphase (MTBF) lassen sich die Verfügbarkeit V und die Unverfügbarkeit U ermitteln. Invers beschreiben V und U das Systemverhalten eines reparierbaren Systems genau dann, wenn sich Betriebsintervalle (System = intakt, $X_S = 0$) und Ausfallphasen (System = defekt, $X_S = 1$) abwechseln. Somit gilt:

$$\begin{aligned} V &= MTTF/MTBF \\ &= MTTF/(MTTF + MTTR) \\ &= V(t \to \infty) \equiv P(X_S = 0) \end{aligned} \tag{7.43}$$

$$\begin{aligned} U &= MTTR/MTBF \\ &= MTTR/(MTTF + MTTR) \\ &= 1 - V \\ &= U(t \to \infty) \equiv P(X_S = 1) \end{aligned} \tag{7.44}$$

Um eine Abschätzung der wirtschaftlichen Bedeutung vorzunehmen, wird in Tabelle 7.4 die Verfügbarkeit der IT-Systeme im Werk Sindelfingen aufgezeigt. Dabei beziehen sich die relativen Werte auf gefertigte Fahrzeuge in einem Zeitraum von 12 Monaten. Das arithmetische Mittel wird somit mit **99,95 %** beziffert.

01	02	03	04	05	06	07	08	09	10	11	12
99,94	100	99,97	99,80	99,86	100	100	99,96	100	100	99,95	99,95

Tabelle 7.4: Monatliche Verfügbarkeit der IT-Systeme in %

Verfügbarkeit SOA (Prozess Bestellanforderung)

Eine grobe Schätzung der Verfügbarkeit V innerhalb eines Jahres beträgt 99,95 %, somit gilt aus (7.43) und (7.44):

$$\begin{aligned} 0,9995\ (99,95\,\%) &= MTTF/525.600 \\ MTTF &= 525.337,2 \end{aligned} \tag{7.45}$$

$$\begin{aligned} U &= 1 - 0,9995 \\ &= 0,0005\ (0,05\,\%) \end{aligned} \tag{7.46}$$

Die mittlere ausfallfreie Zeitspanne MTTF beträgt 525.337 Minuten und 12 Sekunden, somit gilt aus (7.41) und (7.43):

$$\begin{aligned} MTTR &= 525.600 - 525.337,2 \\ &= 262,8 \end{aligned} \tag{7.47}$$

Verfügbarkeit SOA + EMEO-Layer (Prozess Auftragsplanung Getriebe)

Die Ausfallzeit untergliedert sich in „geplante Ausfallzeit“ und „ungeplante Ausfallzeit“ siehe 7.2 auf Seite 132. Bei „geplanter Ausfallzeit“ **(0.3)** wird eine Verbesserung um **2/3** angenommen. Bei „ungeplanter Ausfallzeit“ wird bei menschlichen Fehlverhalten **(0.15)** ebenfalls eine Verbesserung um **2/3** angenommen. Somit ergibt sich eine verbesserte Eintrittswahrscheinlichkeit von **30 %** (0.3 * 2/3 + 0.15 * 2/3). Das mindert die mittlere Ausfalldauer MTTR von 262 Minuten und 48 Sekunden wie folgt:

$$\begin{aligned} MTTR &= 262,8 * 0,7 \\ &= 183,96 \end{aligned} \tag{7.48}$$

Nach rechnerischer Eliminierung der Fehlerursachen ergibt sich die neue mittlere Ausfalldauer MTTR von 183 Minuten und 57,6 Sekunden. Somit erhöht sich die mittlere ausfallfreie Zeitspanne MTTF wie folgt:

$$\begin{aligned} MTTF &= 525.600 - 183,96 \\ &= 525.416,04 \end{aligned} \tag{7.49}$$

Nach Minderung der mittleren Ausfalldauer MTTR und Steigerung der mittleren ausfallfreien Zeitspanne MTTF ergibt sich die folgende Verfügbarkeit V und Unverfügbarkeit U:

$$\begin{aligned} V &= 525.416,04/525.600 \\ &= 0,99965(99,965\,\%) \end{aligned} \tag{7.50}$$

$$\begin{aligned} U &= 1 - 0,99965 \\ &= 0,00035\ (0,035\,\%) \end{aligned} \tag{7.51}$$

Ermittlung und Bewertung der Ergebnisse

	SOA	SOA + EMEO-Layer	Differenz
MTTR	262,8 Min./Jahr	183,96 Min./Jahr	+78,84 Min./Jahr
U	0,0005 (0,050 %)	0,00035 (0,035 %)	+0,00015 (0,015 %)

Tabelle 7.5: Ermittelte Ergebnisse

In Tabelle 7.5 werden die ermittelten Ergebnisse dargestellt. Um die Ergebnisse besser bewerten zu können, findet im Folgenden eine monetäre Bewertung statt. Unter Berücksichtigung der Verteilung der unterschiedlichen Baureihen sowie der unterschiedlichen Netto-Verkaufspreise der Fahrzeuge nehmen wir einen Umsatz von 45.000 Euro pro Fahrzeug an. Das Werk Sindelfingen produziert in Summe 2.000 Fahrzeuge am Tag. Somit ergibt sich ein möglicher Tagesumsatz von 90.000.000 Euro. Abhängig von der Kritikalität eines Systems ist der Umsatzverlust durch Unverfügbarkeit an einem Tag (SOA ohne und mit EMEO-Layer) wie folgt monetär zu bewerten:

Umsatz in Euro/Tag	Unverfügbarkeit (U)	Umsatzverlust in Euro/Tag
90.000.000	0,00050	45.000
90.000.000	0,00035	31.500
		13.500

Tabelle 7.6: Monetäre Bewertung

Der EMEO-Layer erreicht eine Verbesserung des Umsatzverlustes um täglich **13.500 Euro** und erhöht die Verfügbarkeit auf **99,96 %**. Auch wenn die hier dargestellte Berechnung eine **Abschätzung** der wirtschaftlichen Bedeutung darstellt, wird dennoch das eventuell **vorhandene Einsparungspotential** deutlich.

Hierbei wurden Einsparungen der Personalkosten (Gehälter/Löhne, Sonderzahlungen, Fortbildungsmaßnahmen etc.) durch die Eliminierung menschlichen Eingreifens (Personengruppe A) nicht berücksichtigt.

Kapitel 8

Fazit

8.1 Schlussfolgerungen

8.1.1 Kernelemente

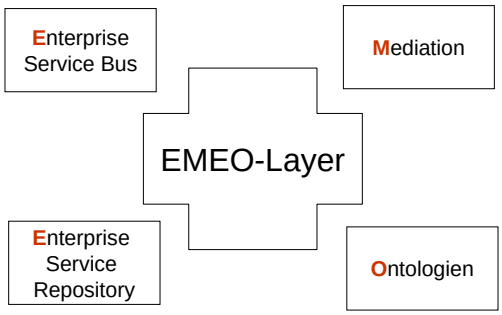

Abbildung 8.1: Kernelemente des EMEO-Layers

Das primäre Ziel der vorliegenden Dissertation ist die Flexibilisierung der Schnittstellenbeschreibungen und die Verbesserung der Verfügbarkeit von Prozessabläufen bei geplanter Ausfallzeit. Der hier vorgeschlagene EMEO-Layer setzt Ontologien zur unterstützten Suche äquivalenter Services (Kandidaten) ein, wurde prototypisch implementiert und durch aktuelle Realisierungsmöglichkeiten in einer Service-orientierten Architektur validiert. Der EMEO-Layer definiert eine logische Schicht durch Kombination der vier Kernelemente:

- Entwurfsmuster Mediation,
- definierte Konzepte in einer Ontologie sowie den beiden Ablaufumgebungen
- Enterprise Service Bus und
- Enterprise Service Repository.

Im Folgenden werden die vier Kernelemente des EMEO-Layers dokumentiert.

Mediation. In der Ablaufumgebung ermöglicht die Lookup-Komponente die Ermittlung der bereits bestimmten Schnittstellenbeschreibungen und Service-Endpunkte. Die Mediation nutzt die redundante Industrialisierung der Services, um sich bei Unverfügbarkeit eines Service-Providers mit einem äquivalenten Service-Provider zu verbinden.

Ontologie. In der Ontologie wird das Wissen der Know-how-Träger externalisiert. Durch den Standard WSDL-S werden Konzepte in der Ontologie mit Eingabe (`I`), Ausgabe (`O`), Vor- (`P`) und Nachbedingung (`E`) der Services verbunden. Nach Anwendung der Inferenz ermittelt der Algorithmus Semantik-Service-Finder eine bewertete Liste äquivalenter Services (Kandidaten).

Enterprise Service Bus. Der Enterprise Service Bus ist die Ablaufumgebung der Mediation. Von dort wird die benutzerdefinierte Eigenschaft *status* abgefragt und die Mediation gesteuert.

Enterprise Service Repository. Diese Komponente ist während der Entwicklungszeit und in der Ablaufumgebung ein Kernelement der logischen Schicht EMEO-Layer. Das Enterprise Service Repository verwaltet die benutzerdefinierte Eigenschaft *status* zur Entwicklungszeit und besitzt eine grafische Oberfläche zur Administration. In der Ablaufumgebung wird ein definierter Zugriff auf den Wert der benutzerdefinierten Eigenschaft *status* sichergestellt.

8.1.2 Unterstützende Elemente

Unterstützt durch externalisiertes Wissen in einer Ontologie und der Inferenz in Kombination mit dem Algorithmus Semantik-Service-Finder zeigt die vorliegende Arbeit die Generierung einer bewerteten Liste äquivalenter Services (Kandidaten) in der Entwicklungszeit. Nach der Auswertung der bewerteten Liste werden unabhängige Pfade der Service-Kandidaten via Mediation manuell erstellt und in die Ablaufumgebung ausgeliefert. Bei geplanter Ausfallzeit eines Service-Providers wird die benutzerdefinierte Eigenschaft *Service-Provider.status* auf `false` gesetzt. Somit wählt die Mediation automatisch den nächsten verfügbaren Pfad und ruft den damit verbunden Service-Provider auf. Im Folgenden werden diese unterstützenden Elemente des EMEO-Layers dokumentiert.

Vorgehensweise TD+. Die generische Vorgehensweise TD+ hat das Ziel fachliche Anforderungen (eEPKs) in ein ablauffähiges Modell (BPEL) zu transformieren. In Verbindung mit der praktischen Anwendung Prozess „Bestellanforderung“ (Verfügbarkeitsklasse N, nomalverfügbar) erwiesen sich die Schritte I. bis X. als anwendbar und praktikabel (siehe Kapitel 4.2 auf Seite 79). In der erneuten praktischen Anwendung Prozess „Auftragsplanung Getriebe“ (Verfügbarkeitsklasse XH,

höchstverfügbar) erwies sich TD+ als nicht ausreichend. Mit der Erweiterung der Schritte VII. und VIII. durch den Algorithmus Semantik-Service-Finder, der Bewertung der Serviceoperationen und die manuelle Erstellung der Mediation, wird die Vorgehensweise TD++ definiert (siehe Kapitel 7.6 auf Seite 130).

Semantik-Service-Finder. Zur Bestimmung äquivalenter Services (Kandidaten) ist die Inferenz notwendig aber nicht hinreichend. Der Algorithmus „Semantik-Service-Finder“ ist eine hinreichende Bedingung, um äquivalente Services (Kandidaten) zu ermitteln. Die gefolgerte Ontologie O_2 wird zur Generierung einer indizierten Liste (durch den Semantik-Service-Finder) genutzt. Input (`I`), Output (`O`), Precondition (`P`) und Effect (`E`) jeder Serviceoperation wird mit semantischen Informationen (Konzepten) ausgezeichnet. Input (`I`) und Output (`O`) können mit einem oder mehreren Konzepten ausgezeichnet werden. Eine konkrete Auszeichnung wird in dieser Arbeit als Fragment bezeichnet. Die Grundlage dafür bildet die Bewertung der Fragmente in Form von Ober- und Unterkonzepten durch 0.5 sowie die Äquivalenz der Konzepte durch 1.0 (siehe Kapitel 6.2 auf Seite 88).

8.1.3 Entwicklungs- und Ablaufumgebung

In dieser Arbeit wird strikt zwischen Entwicklungs- und Ablaufumgebung unterschieden. Der EMEO-Layer findet seine Anwendung in der Entwicklungsumgebung und hat positive Auswirkungen auf die Ablaufumgebung. In der Entwicklungsumgebung wird die Äquivalenz der Services (Kandidaten) ermittelt sowie die Mediation erstellt. Nach einer erfolgreichen Auslieferung des Codes in die Ablaufumgebung, wird, abhängig von externalisierten Informationen im Enterprise Service Repository, ein äquivalenter Service aufgerufen. Somit ist die Serviceschnittstelle bei geplanter Ausfallzeit flexibel.

8.2 Vor- und Nachteile

Die positiven Auswirkungen des EMEO-Layers auf eine Service-orientierte Architektur sind, die verbesserte Verfügbarkeit der Prozessabläufe (bei geplanter Ausfallzeit) und die flexible Schnittstellenverarbeitung in der Ablaufumgebung. Der EMEO-Layer eliminiert partiell die fixe Schnittstellenverarbeitung einer Service-orientierten Architektur, ohne die bereits vorhandenen Vorteile negativ zu beeinflussen. Durch die Eliminierung potentieller Gefahrenquellen wie Übertragungsfehler an IT-Systemgrenzen und Unverfügbarkeit der Know-how-Träger ist das folgende Verhalten des EMEO-Layers als positiv zu bewerten:

- Ein Protokollwechsel von bspw. SOAP/HTTP auf SOAP/JMS sowie ein Wechsel des Service-Endpunkts (Schnittstelle, IP-Adresse, Port) ist in der Ablaufumgebung komplett möglich.
- Die Schnittstellenbeschreibung in der Ablaufumgebung ist lose gekoppelt, variabel und bleibt flexibel.

- Die Abbildung aller äquivalenten Service-Schnittstellen in der Entwicklungszeit wird ermöglicht, womit die vorhandenen Ressourcen (äquivalente Services) in der Infrastruktur maximal genutzt werden können.
- Die Mediation (bei geplanter Ausfallzeit) von äquivalenten Services in der Ablaufumgebung erlaubt eine hohe Flexibilität.
- Fachliche Funktionen der IT-Systeme werden durch Services repräsentiert und lösen somit die monolithische Darstellungen der IT-Systeme auf. Die Ermittlung redundant implementierten Services kann zur Kostenreduktion genutzt werden.

Diese Form der losen Kopplung unterstützt die Flexibilisierung der IT bezogen auf geplante Ausfallzeit, mangelt aber bei ungeplanter Ausfallzeit. Somit sind die folgenden Eigenschaften des EMEO-Layers als negativ zu bewerten:

- Schnittstellenbeschreibungen müssen zum Entwicklungszeitpunkt bekannt sein.
- Bei ungeplanter Ausfallzeit ist die Mediation zwischen äquivalenten Services und folglich das dynamische Finden, Auswählen und Binden (in der Ablaufumgebung) nicht möglich.
- Modelliertes Know-how der Personengruppe A wird in der Ontologie als statisch betrachtet. Bei Änderungen der Ontologie ist ein iterativer Prozess vorgesehen. Dieser Prozess wurde in Verbindung mit Dynamik hier nicht näher untersucht.

Die vorliegende Arbeit wurde bei dem Wettbewerb „Bestes SOA-Industrieprojekt 2007“ vom Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (BITKOM) in der Kategorie „Cross-Vendor“ mit dem ersten Preis ausgezeichnet. Der Preis wurde am 15.11.2007 von Herrn Dr. Andreas Goerdeler (Bundesministerium für Wirtschaft und Technologie) während des Jahresabschluss-Events der BITKOM-Hauptvorstands Taskforce SOA in Berlin überreicht (siehe Anhang I.13 auf Seite 179).

8.3 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden die Grundlagen der Service-orientierten Architektur, das Entwurfsmuster Mediator und semantische Konzepte zusammengefasst und diskutiert. Die Top-Down-Vorgehensweise TD+ fand die praktische Anwendung im fachlichen Prozess Bestellanforderung (BANF) der Verfügbarkeitsklasse N (normalverfügbar). In der Anwendung (Modellierung bis hin zum ablauffähigen Prototyp) erwies sich TD+ als positiver Beitrag und ohne Defizite. Die erneute Anwendung an einem fachlichen Prozess der höchsten Verfügbarkeitsklasse XH (höchstverfügbar) setzte eine verbesserte Verfügbarkeit der Service-Provider voraus. Die quasi-statische Bindung sowie die Ergänzung der Lookup-Komponente im

Enterprise Service Bus zeigte Defizite bei potentiellen Gefahrenquellen, wie bspw. Unverfügbarkeit der Service-Provider aus und erwies sich somit als sub-optimal.

In dieser Arbeit wurde der EMEO-Layer vorgeschlagen, um durch einen semantischen semiautomatischen Ansatz (in der Entwicklungsumgebung) die Defizite der Service-orientierten Architektur bei geplanter Unverfügbarkeit der Service-Provider zu minimieren. Der EMEO-Layer definiert eine logische Schicht aus den Ablaufumgebungen: Enterprise Service Bus und Enterprise Service Repository, sowie dem Entwurfsmuster Mediation und der Definition von Konzepten in einer Ontologie. Dabei wurde basierend auf WSDL-S ein Algorithmus (Semantik-Service-Finder) entwickelt, der in der Entwicklungsumgebung eine bewertete Liste äquivalenter Service-Kandidaten zurückliefert. In Anwendung des EMEO-Layers wurde durch ein exemplarisches Szenario und durch Implementierung eines lauffähigen Prototypen die Verbesserung der Verfügbarkeit von Prozessabläufen durch eine flexible Schnittstellenverarbeitung nachgewiesen. Das für die Validierung notwendige exemplarische Szenario bestand aus den beiden Teilelementen: Anreicherung der Serviceoperationen durch semantische Auszeichnungen sowie Externalisierung des Know-hows der Wissensträger. Der lauffähige Prototyp besteht aus den drei Teilelementen: lose Kopplung durch produktneutrale Mediation, Anwendung des Algorithmus „Semantik-Service-Finder“ zur Identifizierung äquivalenter Service-Kandidaten und Implementierung der Mediation (inkl. Lookup-Komponente) in den Produkten IBM WebSphere Integration Developer, Enterprise Service Bus, Process Server und Enterprise Service Repository. Die neue, abgeleitete Top-Down-Vorgehensweise TD++ wurde gegenüber TD+ um den EMEO-Layer in der Entwicklungsumgebung ergänzt. Im Folgenden werden weitere Untersuchungen aufgezeigt, um diesen Ansatz industrialisieren zu können:

- Eine Automotive-Ontologie muss abgestimmt mit dem Verband der Automobilindustrie erarbeitet, diskutiert und realisiert werden.
- Das Unternehmen Ontoprise GmbH legte im Juni 2007 der Daimler AG ein „Whitepaper“ mit dem Titel „Semantic Service Finder“ vor, um die notwendige Erstellung der Ontologie zu konkretisieren und zu realisieren. Der Projektstart war Mitte Januar 2008.
- Die WebSphere Business Service Fabric stellt heute bereits Ontologien in OWL für die Bereiche Finanzen und Banken zur Verfügung. Ein weiterer zukunftsweisender Schritt ist die Aufnahme einer abgestimmten Ontologie in die WebSphere Business Service Fabric. Der EMEO-Layer wird bei der kommenden Release-Planung des WebSphere Integration Developer als ergänzende Funktionalität in Betracht gezogen.
- Das Forschungsprojekt *Semantic Annotations for WSDL (SAWSDL)* untersuchte die Anreicherung der Qualitätsmerkmale von Serviceoperationen in einer ähnlichen Art und Weise. Der EMEO-Layer ist auf die fachliche Anrei-

cherung (`IOPE`) fokussiert und betrachtet Qualitätsmerkmale nicht. Die Kombination des EMEO-Layers mit SAWSDL verspricht ein Potential, das in einer Forschungsinitiative untersucht werden sollte.

- Der Semantik-Service-Finder bewertet die äquivalente Serviceoperation `BZM.ermittleSollTermin` mit 367 (siehe Kapitel 7.4 auf Seite 122). Dabei werden die Konzepte `AmountPlausible`, `Aggregate` und `Roadster` als Vorbedingung (`P`) berücksichtigt. Wird das Konzept `Aggregate` durch das Konzept `DAINAG1` ersetzt, so bewertet der Semantik-Service-Finder die Serviceoperation `BZM.ermittleSollTermin` ebenfalls mit 367. Das Konzept `DAINAG1` repräsentiert in der Realität das Getriebe DAINAG1. Fachlich ist das DAINAG1 aber nur eine Teilmenge der durch das Konzept `Aggregate` geforderten Getriebe und somit kein Servicekandidat. Hier sollte der Algorithmus verfeinert werden.

- Sollten bei der iterativen Vorgehensweise zur Modellierung der Ontologie Konflikte in Form von fachlichen Widersprüchen entdeckt werden, so muss jede Auszeichnung der Serviceoperationen mit einem oder mehreren Konzepten, erneut überprüft werden. Hier sollte ein automatisierter Mechanismus erarbeitet werden.

Die Kombination der logischen Ebene EMEO-Layer mit der bereits heute zur Verfügung stehenden Ontologie aus der WebSphere Business Fabric stellt zukünftig eine sehr gute Basis für die Realisierung höchstverfügbarer Prozesse (Verfügbarkeitsklasse XH) dar. Forschungsprojekte wie bspw. OWL-S und WSMO, die eine flexible Schnittstellenverarbeitung in der Ablaufumgebung untersuchen, partizipieren von den Erkenntnissen dieser Arbeit und von der logischen Schicht EMEO-Layer. Basierend auf dem EMEO-Layer ist die Validierung der Ergebnisse der Forschungsprojekte im Bereich Semantik Web Services ein weiterer signifikanter Fortschritt.

Akronyme

ABox	Assertional Box
ACID	atomar, konsistent, isoliert, dauerhaft
AI	Artificial Intelligence
ARIS	Architektur integrierter Informationssysteme
BAM	Business Activity Monitoring
BANF	Bestellanforderung
BAPI	Business Application Programming Interface
BITKOM	Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V.
Blob	Binary Large Object
BPEL4WS	Business Process Execution Language for Web Services
BPM	Business Process Management
BPMI	Business Process Modeling Initiative
BPML	Business Process Modeling Language
BPMN	Business Process Modeling Notation
BZM	Betriebszeitmodell
CCTS	Core Component Technical Specification
CMMI	Capability Maturity Model Integration
COBOL	Common Business Oriented Language
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CVD	Commercial Vehicle Division
DAML	DARPA Agent Markup Language
DAML-OIL	DAML Ontology Interchange Language
DARPA	Defense Advanced Research Projects Agency
DCOM	Distributed Component Object Model
DERI	Digital Enterprise Research Institute
DIG	DL Implementation Group
DL	Description Logic
ebXML	Elektronic Business XML
EDA	Event-driven Architecture
EDC	European-Data-Center
eEPK	Erweiterte Ereignisgesteuerte Prozessketten

EIC ..... Eco-System Industrie Standards Gruppe
ERM ..... Entity-Relationship-Modell
ESB ..... Enterprise Service Bus
ESR ..... Enterprise Service Repository
ETTK ..... Emerging Technologies Toolkit
FaCT ..... Fast Classification of Terminologies
GoF ..... Gang of Four
GUI ..... Grafisches User Interface
HT ..... Human Task
HTML ..... Hypertext Markup Language
IDL ..... Interface Definition Language
IDS ..... Instandhaltung-, Ersatzteile- und Dokumentationssystem
IOPE ..... Input, Output, Precondition, Effect
ITM ..... Informationstechnologie Management
J2SE ..... Java 2 Standard Edition
JBI ..... Java Business Integration
JCo ..... Java-Connector
JSF ..... JavaServer Faces
JSP ..... JavaServer Pages
JVM ..... Java Virtueller Maschinen
KAON ..... KArlsruhe ONtologie
KI ..... Künstliche Intelligenz
KPI ..... Key Performance Indikatoren
KR ..... Knowledge Representation
MAS+ ..... Montageauftragssteuerung Plus
MBD ..... Material-Bestandsführung und Disposition
MES ..... Material-Einkauf-System
NACOS ..... Neues Accounting- und Controlling-System
NDA ..... Non-Disclosure Agreement
NFA ..... Nicht-funktionale Anforderungen
OASIS ..... Advancement of Structured Information Standards
OMG ..... Object Management Group
ORB ..... Object Request Broker
OTKM ..... On-To-Knowledge Methodology
OWL ..... Web Ontology Language
OWL-S ..... Semantic Markup for Web Services
P4 ..... Process Four
PAI ..... Pro-Aktive-Infrastruktur
PLZ ..... Produktionsleitzentrale
QoS ..... Quality of Service
Quasar ..... Quality Software Architecture
RACER ..... Renamed Abox and Concept Expression Reasoner
RDF ..... Resource Description Framework

RDF(S) ..... Resource Description Framework (RDF) & RDF-Schema
RMI ..... Remote Method Invocation
RPC ..... Remote Procedure Call
SAP-GDT ..... SAP-Global-Datatypes
SAWSDL ..... Semantic Annotations for WSDL
SDO ..... Service Data Objects
SGMM ..... SOA Governance & Management Methode
SMO ..... Service Message Objects
SOA ..... Service-orientierte Architektur
SOA MM ..... SOA Maturity Model
SOFA ..... SOA for Automotive
SSF ..... Semantik-Service-Finder
SWS ..... Semantischen Web Services
SWSF ..... Semantic Web Services Framework
TBE ..... Teilebedarfsermittlung
TBox ..... Terminological Box
TD+ ..... Top-Down Plus
TD++ ..... Top-Down PlusPlus (Erweiterung TD+)
UDDI ..... Universal Description, Discovery and Integration
UML ..... Unified Modeling Language
UN/CEFACT .... United Nations Centre for Trade Fac. and Electronic Business
URI ..... Uniform Resource Identifier
USWS ..... Ubiquitous Semantic Web Services
VDA ..... Verband der Automobilindustrie
W3C ..... World Wide Web Consortium
WAS ..... WebSphere Applikation Server
WB ..... Wissensbasis
WES ..... Wareneingang und -abgleich-System
WESB ..... WebSphere Enterprise Service Bus
WfMC ..... Workflow Management Coalition
WID ..... WebSphere Integration Developer
WKD ..... Wertschöpfungskettendiagramm
WPS ..... WebSphere Process Server
WSDL ..... Web Service Description Language
WSDL-S ..... Web Service Semantics
WSFL ..... Web Services Flow Language
WSMF ..... Web Service Modeling Framework
WSMO ..... Web Service Modeling Ontology
WSRR ..... WebSphere Service Registry und Repository
WWW ..... World Wide Web
XML ..... eXtensible Markup Language
XML-RPC ..... Extensible Markup Language Remote Procedure Call
XPDL ..... XML Process Definition Language

XSD ..... XML Schema Definition
XTM ..... Topic Maps

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Publikationen

1. M. Herrmann, O. Dalferth, H-J. Groß, H. Moertl. Real-life SOA experiences: Top-Down approach based on an implemented purchase requisition process. Proceedings of 5th Intern. Workshop on SOA and Web Services in conjunction with Engineering SOA'07 and ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2007), October 21-25, Monteal, Canada, ISBN 978-82-997428-1-8.

2. M. Herrman, M. A. Aslam, O. Dalferth: Applying Semantics (WSDL, WSDL-S, OWL) in Service Oriented Architectures (SOA). 10th Intl. Protégé Conference, July 15-18, 2007, Budapest, Hungary.

3. M. Herrmann, O. Dalferth: SOA activity at MCG. ARIS Process World 2007 in conjunction with ARIS UserDay: Implementation & Controlling, June 13-15, 2007, Berlin, Germany.

4. M. A. Aslam, S. Auer, J. Shen, M. Herrmann: An Integration Life Cycle for Semantic Web Services Composition. Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD 07), April 26-28, 2007, Melbourne, Australia, ISBN 1-4244-0962-4, pp 490-495.

5. M. A. Aslam, S. Auer, J. Shen, M. Herrmann: Web Services Composition to Facilitate Grid and Distributed Computing: Current Approaches and Future Framework: Proceedings of 4th International Workshop on Frontiers of Information Technology (FIT 2006), December 20-21, 2006, Islamabad, Pakistan.

6. M. A. Aslam, M. Herrmann, S. Auer, R. Golden: Real-life SOA experiences and an Approach Towards Semantic SOA. Proceedings of 4th International Workshop on SOA and Web Services in conjunction with ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2006), October 22-26, Portland, Oregon, USA, ISBN 82-997428-0-3, pp. 72-81.

7. M. Herrmann, R. Golden: SOA Choreography and Orchestration - An Enterprise Opera. JBoss World 2006: SOA for the Real World, June 12-15, 2006, Las Vegas (Nevada), USA.

8. M. Herrmann, M. A. Aslam: Mercedes Car Group (MCG) Enterprise Architektur - Ein Ansatz zur semantischen Modellierung der Services in einer SOA. In: Fähnrich, K.-P., Kühne, S. Speck, A. Wagner, J. (Hrsg.): Integration betrieblicher Informationssysteme: Problemanalysen und Lösungsansätze des Model-Driven Integration Engineering, Leipziger Beiträge zur Informatik: Band IV. Leipzig: 2006, S. 145-151, ISBN-10: 3-934178-66-9.

9. M. A. Aslam, S. Auer, J. Shen, M. Herrmann: Expressing Business Process Model as OWL-S Ontologies. Proceedings of the 2nd International Workshop on Grid and Peer-to-Peer based Workflows (GPWW 2006) in conjunction with the 4th International Conference on Business Process Management (BPM 2006), Vienna, Austria, LNCS 4103 , Sept. 4, 2006, pp.400-415.

Die folgenden aktiven Teilnahmen basieren auf eingereichten Kurzberichten (max. 8 Seiten) bzw. Zusammenfassungen, Mitarbeit in Interessensgruppen, internen Veranstaltungen oder persönlichen Einladungen.

- Die vorliegende Arbeit wurde bei dem Wettbewerb „Bestes SOA-Industrieprojekt 2007“ auf dem Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (BITKOM) in der Kategorie „Cross-Vendor“ mit dem ersten Preis ausgezeichnet. Der Preis wurde am 15.11.2007 auf dem Jahresabschluss-Event der BITKOM-Hauptvorstands Taskforce SOA in Berlin entgegengenommen (siehe Anhang I.13 auf Seite 179).

- Reviewer/Rezensent (offiziell seit Oktober 2006) des Institute of Electrical and Electronics Engineers (IEEE) in der Sparte Internet Computing.

- Forschung: „MCG Enterprise Architecture – An approach to enrich services in an SOA with semantics“ präsentiert am 22.09.2006 beim Fraunhofer Institut für Sichere Informationstechnologie (SIT) in Darmstadt.

- Forschung: „MCG Enterprise Architecture – definitions, intermediary results and lab based research“ präsentiert am 12. Oktober 2006 auf dem Competence Center Business Networking 3 (CC BN3) Workshop, veranstaltet von der Universität St. Gallen Institut für Wirtschaftsinformatik.

- Praxisvortrag: „MCG Enterprise Architecture – SOA Ansätze bei der Mercedes Car Group“ präsentiert am 08.11.2006 auf dem Stuttgarter Softwaretechnik Forum 2006, veranstaltet vom Fraunhofer Institut für Arbeit und Organisation (IAO) [53].

- Teilnahme (aktiv) an der *SAP Enterprise Services Community Advisory Group Methodology* (unter Non-Disclosure Agreement NDA) am 19. Oktober 2006 auf der SAP TechEd in Amsterdam sowie am 16. April 2007 über WebEx.

- Teilnahme (aktiv) SOA-Arbeitskreis SÜD (Daimler AG, BMW, Audi, Deutsche Sparkasse) am 22.03.2007 in Böblingen, veranstaltet von der IBM Deutschland GmbH.

- Teilnahme (aktiv) SOA-Talk (Deutsche Telekom Laboratories) am 25. Januar 2008 in Berlin, veranstaltet von der Technischen Universität Berlin.

- Teilnahme (aktiv) Business Process Excellence Summit (BPE) Summit am 24. Juli 2008 (offizielle Einladung als Sprecher) in Amsterdam, veranstaltet vom International Quality and Productivity Center (IQPC) Exchange.

Anhang B

Inhaltsübersicht der beiliegenden CD

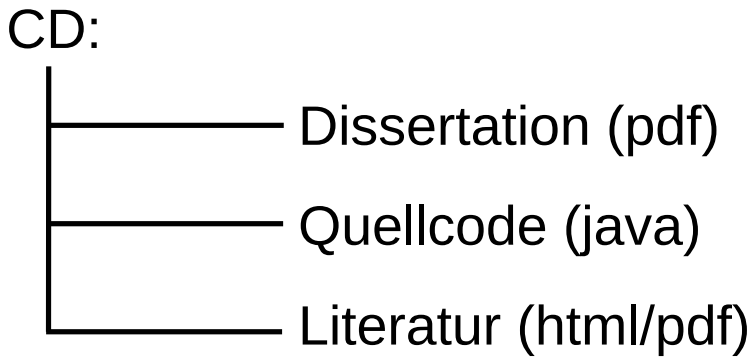

Abbildung B.1: Inhaltsübersicht der beiliegenden CD

Dissertation. Die hier vorliegende Dissertation wurde in LaTeX erstellt und setzt *Adobe Reader 7.0* (oder höher) zur fehlerfreien Darstellung der Arbeit voraus.

Quellcode. Die Java-Klassen des Semantik-Service-Finders (SSF) wurden als lesbaren Quellcode im Ordner Quellcode archiviert. Zur Generierung ablauffähigen Codes wird Java 5.0 vorausgesetzt. Der Mediator im Enterprise Service Bus setzt WebSphere Integration Developer (Entwicklungsumgebung), WebSphere Enterprise Service Bus (Ablaufumgebung) und WebSphere Process Server (Ablaufumgebung) voraus.

Literatur. Die in dieser Arbeit referenzierten Quellen aus dem Internet und die referenzierten Quellen in Form von PDF-Dateien wurden im Ordner Literatur archiviert.

Anhang C

Funktionalen Anforderungen an ein ESR

Klasse Informationsregistrierung:

- Registrierung, De-Registrierung und Aktualisierung von Metadaten über einen Web Service.
- Bereitstellung einer Schnittstelle zur maschinellen Suche nach Services.
- Lokale Speicherung von Metadaten.

Klasse Sicherheits-/Monitoringaspekte:

- Definition und Verwaltung von Nutzern und Nutzergruppen.
- Authentifizierungsmechanismen für Nutzer am Serviceverzeichnis.
- Bestimmung von nutzerspezifischen Sichten und Zugriffsrechten auf die Inhalte des Serviceverzeichnisses (Autorisierung).
- Protokollierungs/Auditing-Funktionen (Überwachung der Zugriffe auf das Serviceverzeichnis).
- Digitale Signatur der registrierten Services.
- Überwachung von QoS-Parametern.

Klasse Informationsmanagement:

- Qualitätssicherung (Validierung registrierter Informationen).
- Approbation registrierter Objekte.
- Automatische Kategorisierung registrierter Objekte.
- Versionierungs-Management von registrierten Objekten.
- Klare Zuordnung von Verantwortlichkeiten für registrierte Objekte.
- Benachrichtigung bei bestimmten Ereignissen im Serviceverzeichnis.

Klasse Architekturbildung:

- Anbindung externer Partner, Bildung von Föderationen von Verzeichnissen.
- Eindeutige Identifikation von Objekten in einer Föderation.
- Föderationsweites Nutzermanagement.

Anhang D

ARIS-Haus

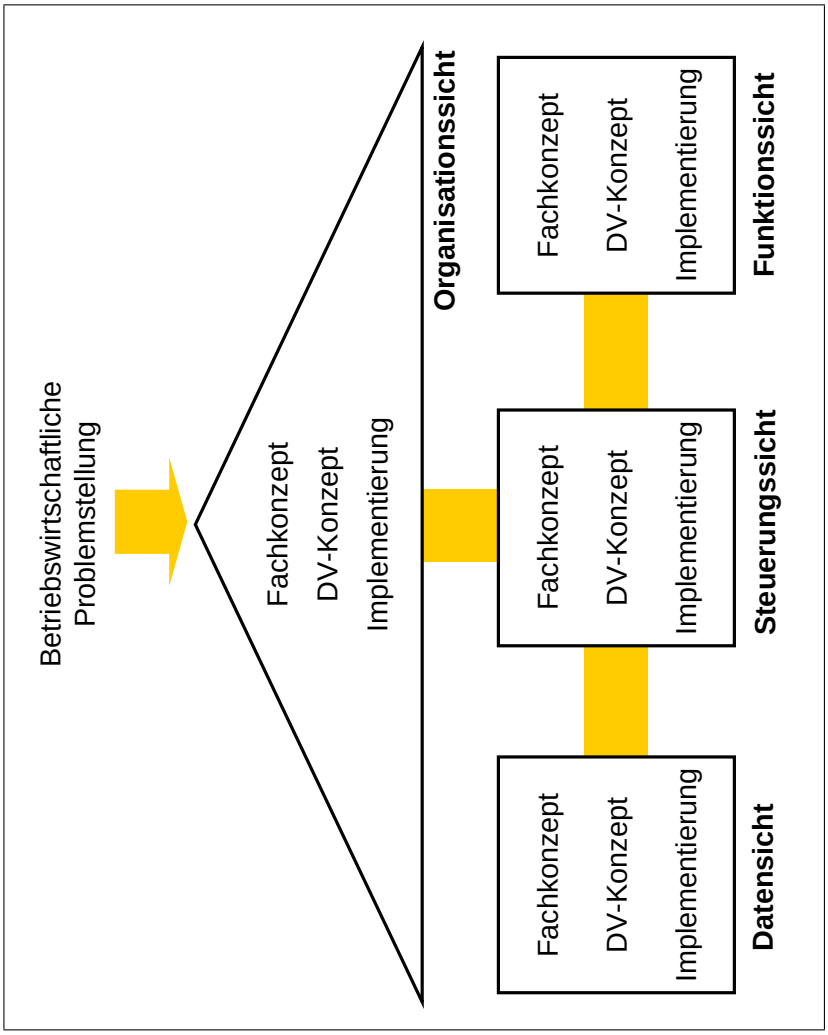

Abbildung D.1: ARIS-Haus

Anhang E

Service-Portfolio-Management

E.1 Rollen

Rolle	Beschreibung
Service-Owner	(Neue Rolle) Fachlich für den Service verantwortlich. Auftraggeber für den Betrieb des Services. Ansprechpartner für alle Detailfragen zu diesem Service.
Service-Consumer	(keine Rolle; nur zur Begriffsabgrenzung) Technische Implementierung der Client-Seite eines Services (nach PAI Terminologie).
Service-Customer	(Neue Rolle) Die Person, die den Service benutzt.
Service-Provider	(keine Rolle; nur zur Begriffsabgrenzung) Technische Implementierung der Server-Seite eines Services (nach PAI Terminologie).
Betreiber	Person, die für den Betrieb von Anwendungen und zugehörigen Services (auf Consumer- und Provider-Seite) verantwortlich ist.
Service-Portfolio-Management	(Neue Rolle) Verwaltet das Service-Repository. Eskalationsinstanz. Das Service-Portfolio-Management benötigt Entscheidungskompetenz über Service-Owner hinweg.
Business Analyst	Person mit fachlichem Know-how, die Geschäftsprozesse modelliert.
Business Architekt (Facharchitekt)	(Neue Rolle) Person mit IT-Know-how, die neue Servicekandidaten aus Geschäftsprozessmodellen ableitet.
Projektoffice	Das Projektoffice ist zuständig für die Priorisierung von ITP-Entwicklungsvorhaben.

Tabelle E.1: Rollen in einer SOA

[1]Service-Owner weisen Ihren Services einem der vier unterschiedlichen Klassifikationsstufen öffentlich *(engl. public)*, intern *(engl. internal)*, vertrauchlich *(engl. confidential)* und geheim *(engl. secret)* zu. Standards wie WS-Security, WS-Federation, WS-Trust etc. reichen hierbei nicht aus, da Unternehmen unterschiedliche Vorstellungen von Klassifizierungen der Daten haben sowie in den verschiedenen Ländern unterschiedliche gesetzliche Vorschriften bestehen.

E.2 Applikationsbeschreibung

Element	Beschreibung
Applikationsname	Eindeutiger Name der Applikation
Version	Eingesetzte Version
Beschreibung	Beschreibung der Applikation
Eigentümer *(engl. Owner)*	Fachlich Verantwortlicher
Betreiber *(engl. Provider)*	Technisch Verantwortlicher
Operationen	Verweis auf die Operationsbeschreibung der von der Applikation genutzten Operationen

Tabelle E.2: Applikationsbeschreibung in einer SOA

E.3 Servicebeschreibung

Element	Beschreibung
Servicename	Eindeutige Kennung des Services
WSDL-Dateiname	URI (siehe Kapitel 2.1.6 auf Seite 27)
Version	Jede Änderung am Service oder einer Operation führt zu einer Erhöhung der Version. Aufbau x.y.z; X . . . Major Release, große fachliche Änderungen Y . . . Minor Release, Einfügen einer neuen Operation Z . . . Bug Fixes
Beschreibung	Aufgabe des Service
Owner	Fachlich Verantwortlicher
Betreiber	Technisch Verantwortlicher
Applikation	Ggf. Applikation, die den Service bereitstellt
Informations-klassifikation	[public, internal, confidential, secret][1]
Sicherheit	Beschreibung der Sicherheitsmechanismen
Security Domäne	Im welchem Umfeld wird der Service eingesetzt [intern, extern]
Interne Dokum.	Verweis auf interne (Entwicklungs-)Dokumentation
Operationen	Verweis auf die Operationsbeschreibung der zugehörigen Operationen[2]

Tabelle E.3: Servicebeschreibung in einer SOA

E.4 Operationsbeschreibung

Element	Beschreibung
Operationsname	Vom Service Consumer verwendeter Name der Service-Operation
Beschreibung	Kurze textuelle Beschreibung der Operation
Status	[Kandidat, geplant, produktiv, deprecated, defunct (inaktiv)] mit Dauer [DD.MM.YYYY]

Tabelle E.4: Operationsbeschreibung in einer SOA (1)

Element	Beschreibung
Human Task	Interaktion mit Benutzer [Ja, Nein]
Input- und Output-Datenstruktur	Von der Operation benötigte bzw. gelieferte Datenstruktur
Fehlermeldungen / Ausnahmezustände	Von der Operation gelieferte Fehlermeldungen bzw. Fehlercodes
Vorbedingungen	Vorbedingungen, die zur korrekten Ausführung einer Operation erfüllt sein muss, insbesondere bezogen auf vorhandene Informationsobjekte. Keine technischen Bedingungen (z.B. Datenbank muss verfügbar sein)
Nachbedingungen	Nachbedingungen, die nach nach Ausführung der Operation wahr sind, insbesondere bezogen auf angelegte bzw. geänderte Informationsobjekte
Technische Anbindung	Implementierung / Schnittstellentechnologie (z.B. Web Service, EJB), Netzwerk- / Transportprotokolle (z.B. SOAP/http, JMS Websphere MQ), Netzwerkadresse
Kompensations-Operation	Operationsname
Bestandsänderung	[JA, NEIN]; JA = create, update, delete; NEIN = readonly, select
Idempotent	[JA, NEIN]; JA= Wiederholter Aufruf der Operation bleibt ohne Auswirkung
Kommunikationsart	[synchron, asynchron]
Antwortzeiten	Ausführungsdauer
Latenzzeit	Dauer bis die Ausführung im asynchronen Fall gestartet wird
Kommunikationsstil	[one-way, request-response]
Test-Consumer	Verweis auf verfügbaren Test-Client. (z.B. Batch-Datei, JUnit-Testklasse)
Referenzen auf Fachlichkeit	Verknüpfung zu fachlichen Aufgaben, Geschäftsprozessen und Fachklassen
Berechtigung	Berechtigung / Rollen
Nachrichtenvolumen	Datenvolumen der ausgetauschten Nachrichten
Erwarteter Durchsatz	Anzahl der Aufrufe pro Zeiteinheit unter spezifischen Randbedingungen
Verfügbarkeit	Ausfallsicherheit
Betriebszeiten	Regulärer Zeitraum, in dem der Service zur Verfügung steht
Verweis auf SLA	Verweis auf SLA-Dokument; ggf. inkl. Versionsnummer
KPI	Festgelegte Key Performance Indikatoren zur Beurteilung der Erreichung von Geschäftszielen
Abhängige Operationen	Verweise auf Operationsbeschreibungen der von dieser Operation genutzten Operationen.

Tabelle E.5: Operationsbeschreibung in einer SOA (2)

[2]Die Integrität zwischen dem WSDL-Dokument und Service- und der Operationsbeschreibung muss toolunterstützt sichergestellt werden.

Anhang F

Global-Datatypes (GDT)

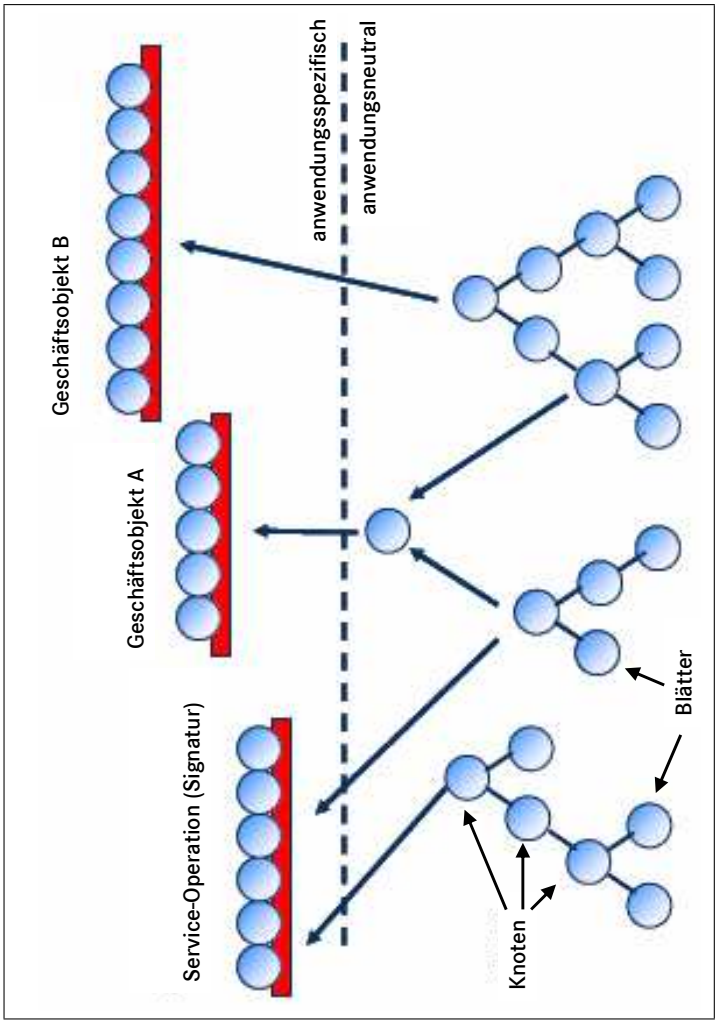

Abbildung F.1: Global-Datatypes (GDT)

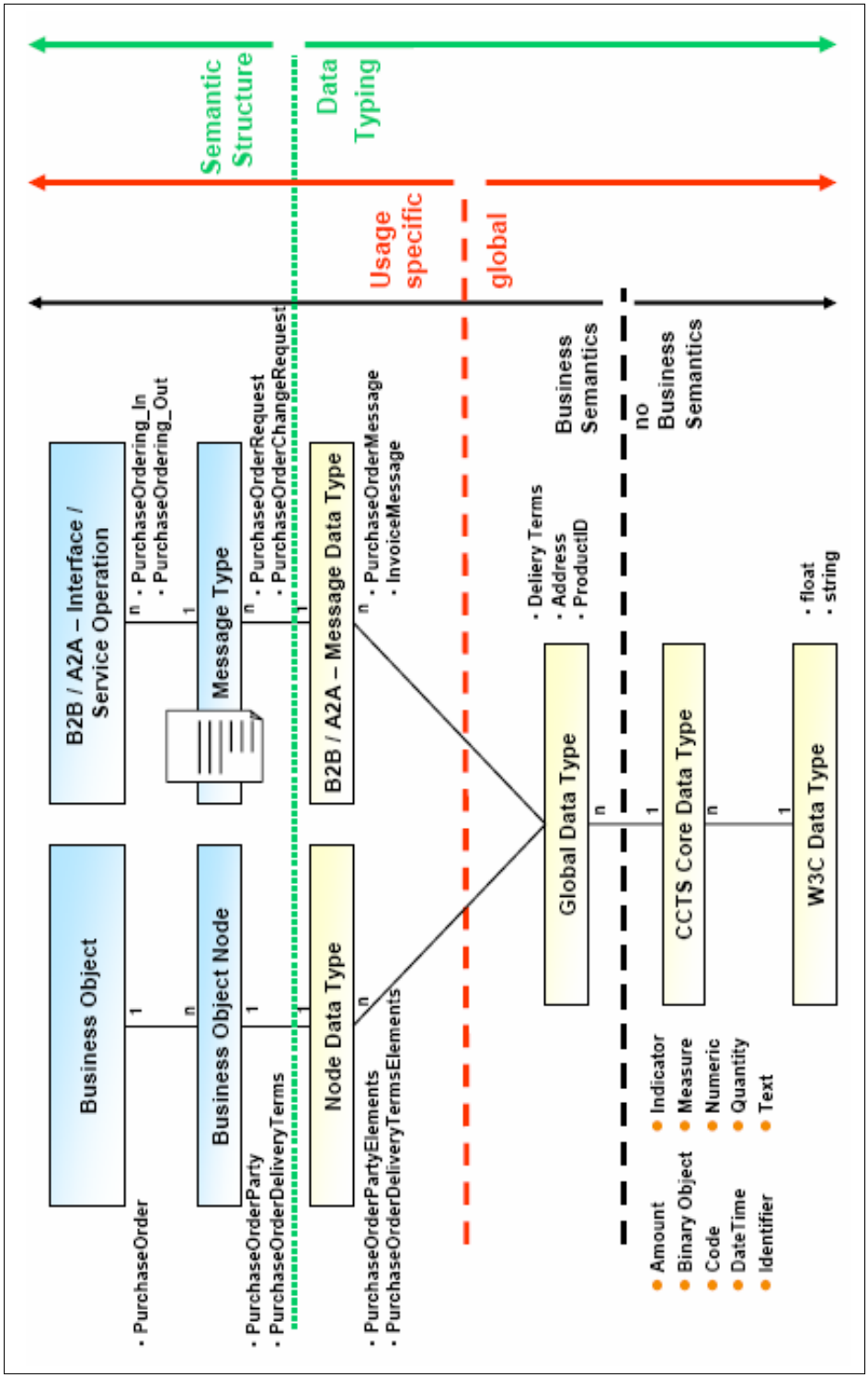

Abbildung F.2: Meta-Modell Global-Datatypes

Anhang G

On-To-Knowledge

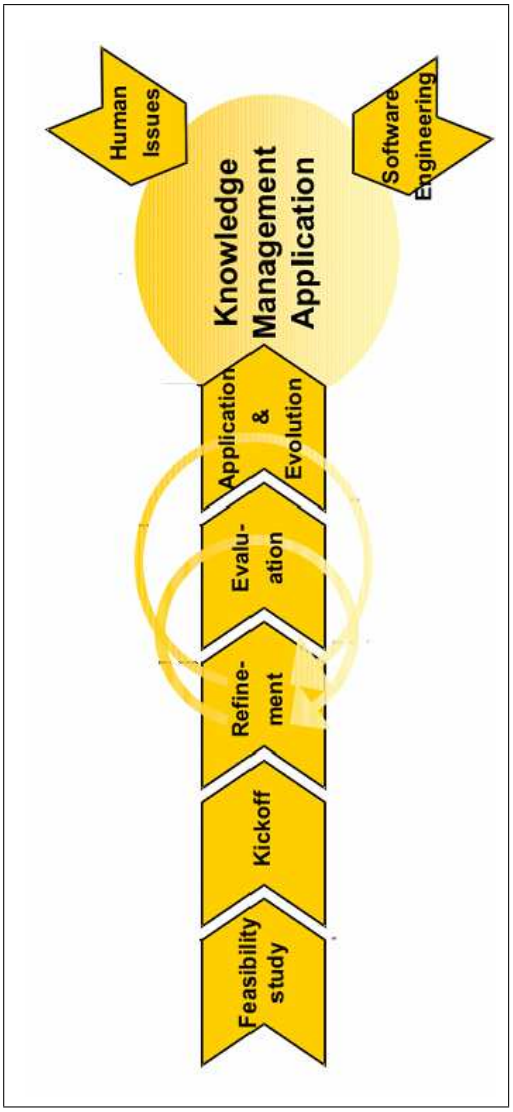

Abbildung G.1: On-To-Knowledge Methodology (OTKM)

Anhang H

SLK in OWL

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY Ontology1181215061 "http://www.owl-ontologies.com/Ontology1181215061.owl#">
  <!ENTITY DAI.owl "http://www.daimler.com/ontologies/DAI.owl">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&Ontology1181215061;"
         xmlns:DAI="&DAI;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;">
  <owl:Class rdf:about="#SLK">
    <rdfs:comment rdf:datatype="&xsd;string">hasSeat == 2</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#DAICar"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasSeat"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
              <owl:Class rdf:about="#leftSeat"/>
              <owl:Class rdf:about="#rightSeat"/>
            </owl:unionOf>
          </owl:Class>
        </owl:allValuesFrom>
        <owl:onProperty rdf:resource="#hasSeat"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
```

```
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasDoorway"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue>
          <Ontology1181215061:TopValuePartition rdf:about="#topSoftTop"/>
        </owl:hasValue>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasTop"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
              <owl:Class rdf:about="#leftDoorFront"/>
              <owl:Class rdf:about="#rightDoorFront"/>
            </owl:unionOf>
          </owl:Class>
        </owl:allValuesFrom>
        <owl:onProperty rdf:resource="#hasDoorway"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:about="#SLR_HardTop"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#SLR_SoftTop"/>
    </owl:disjointWith>
  </owl:Class>

  <owl:Class rdf:about="#TopValuePartition"/>
  <owl:AnnotationProperty rdf:about="&rdfs;comment"/>
</rdf:RDF>
```

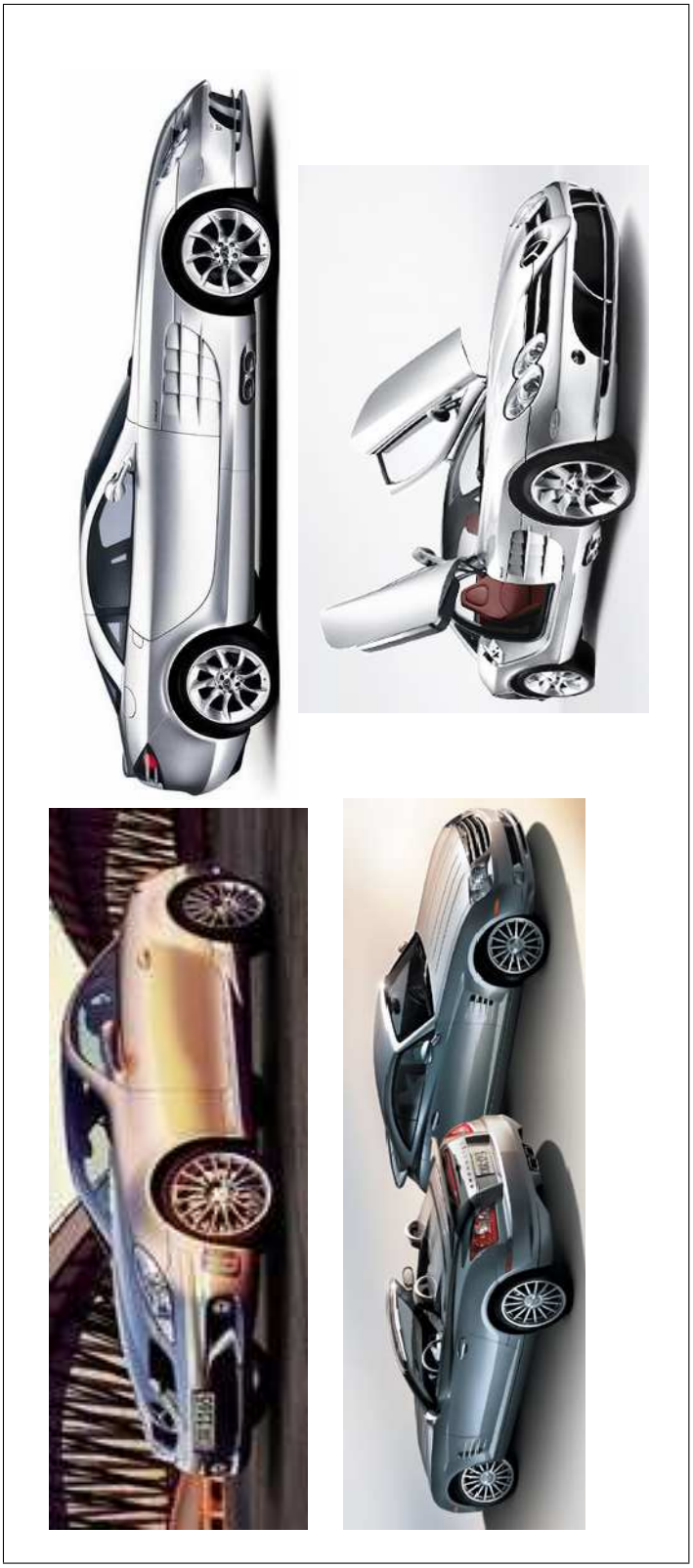

Abbildung H.1: SLK (Roadster), SLR (Coupe) und Crossfire (Roadster und Coupe)

Anhang I

Bestellanforderung (BANF)

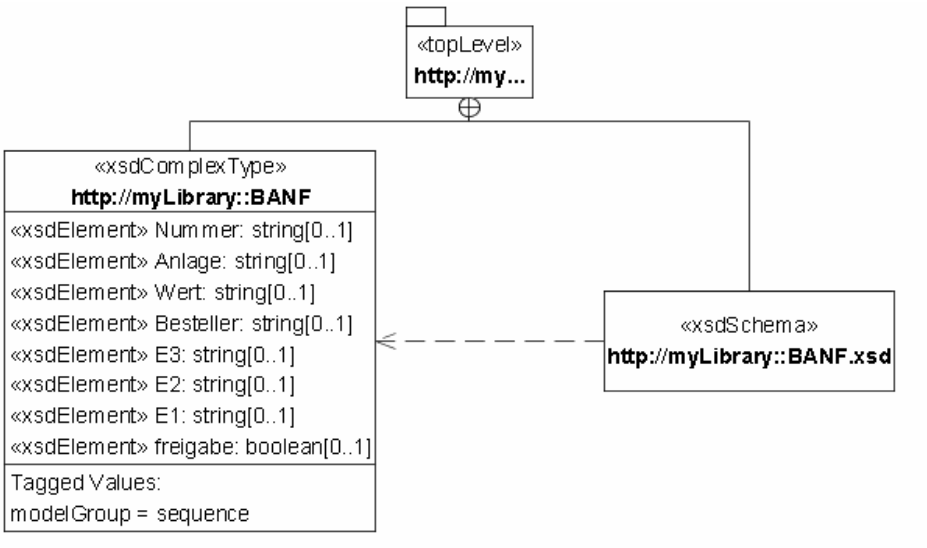

Abbildung I.1: UML Repräsentation des Datenobjektes BANF

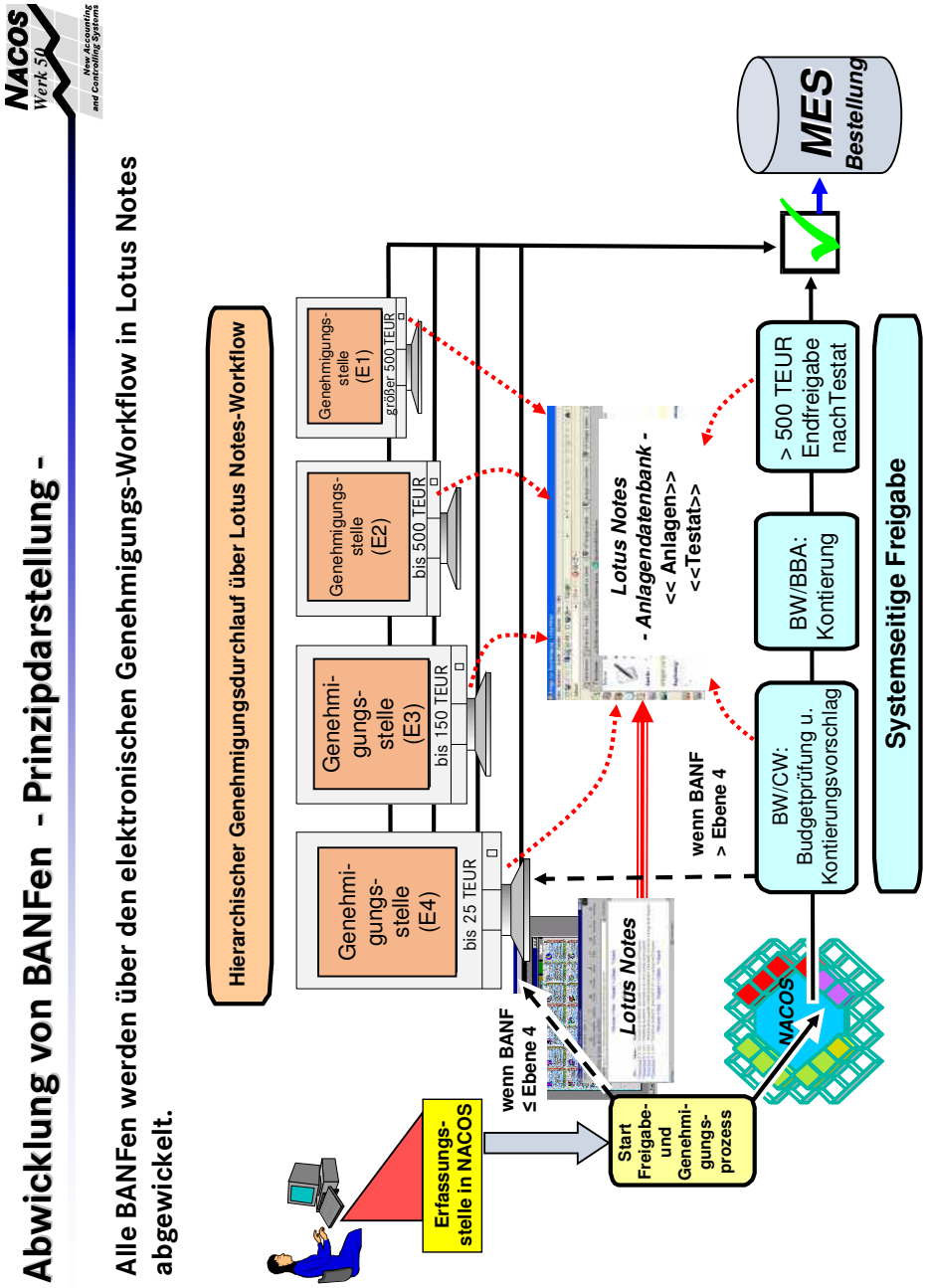

Abbildung I.2: Fachliche Prozessbeschreibung BANF

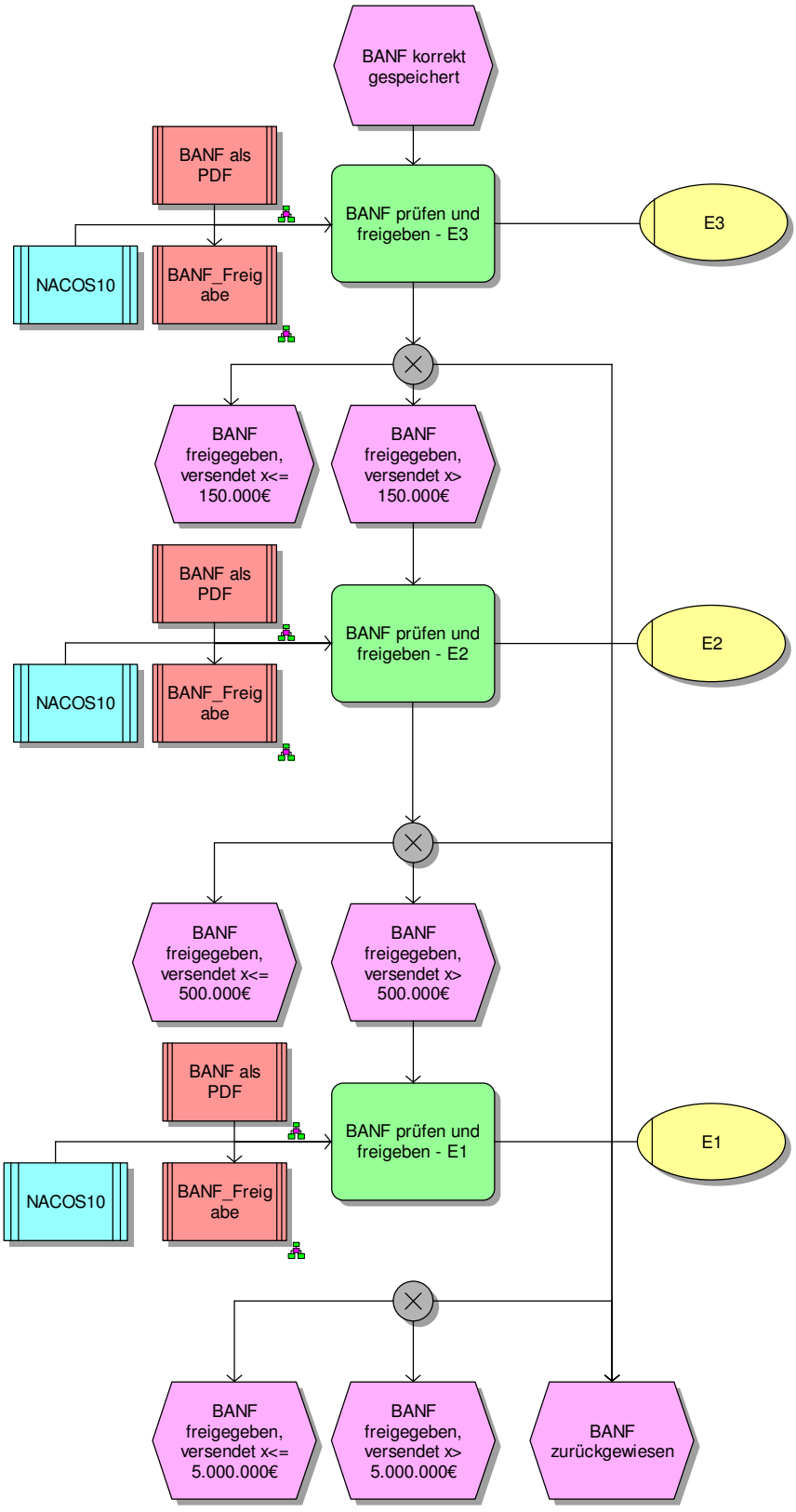

Abbildung I.3: BANF-Prozess in Form einer eEPK

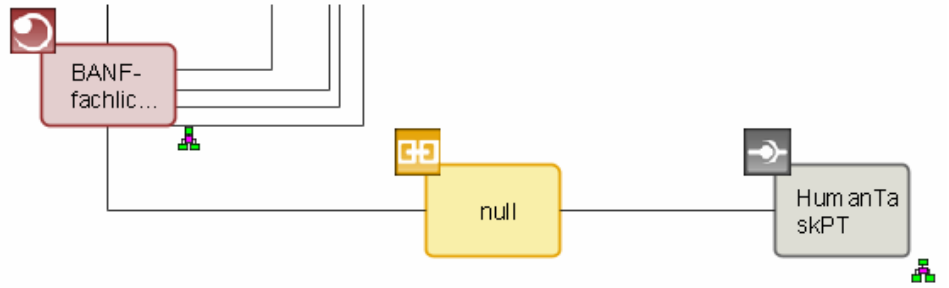

Abbildung I.4: Erweiterung im BPEL Allokationsdiagramm

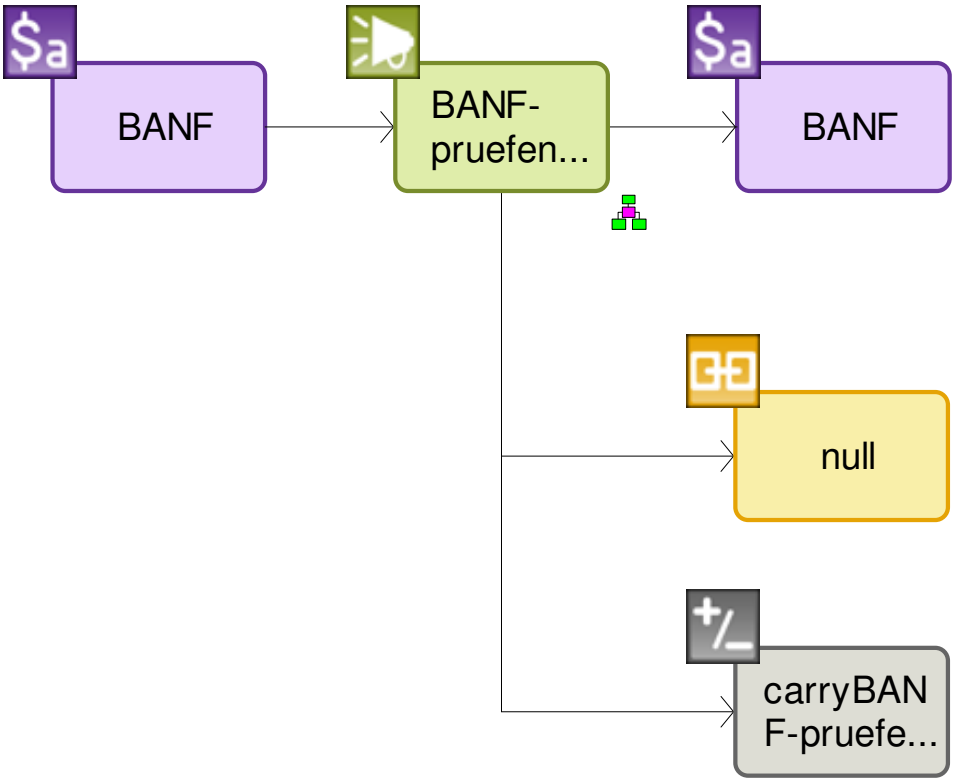

Abbildung I.5: Vollständigtes BPEL Zugriffsdiagramm

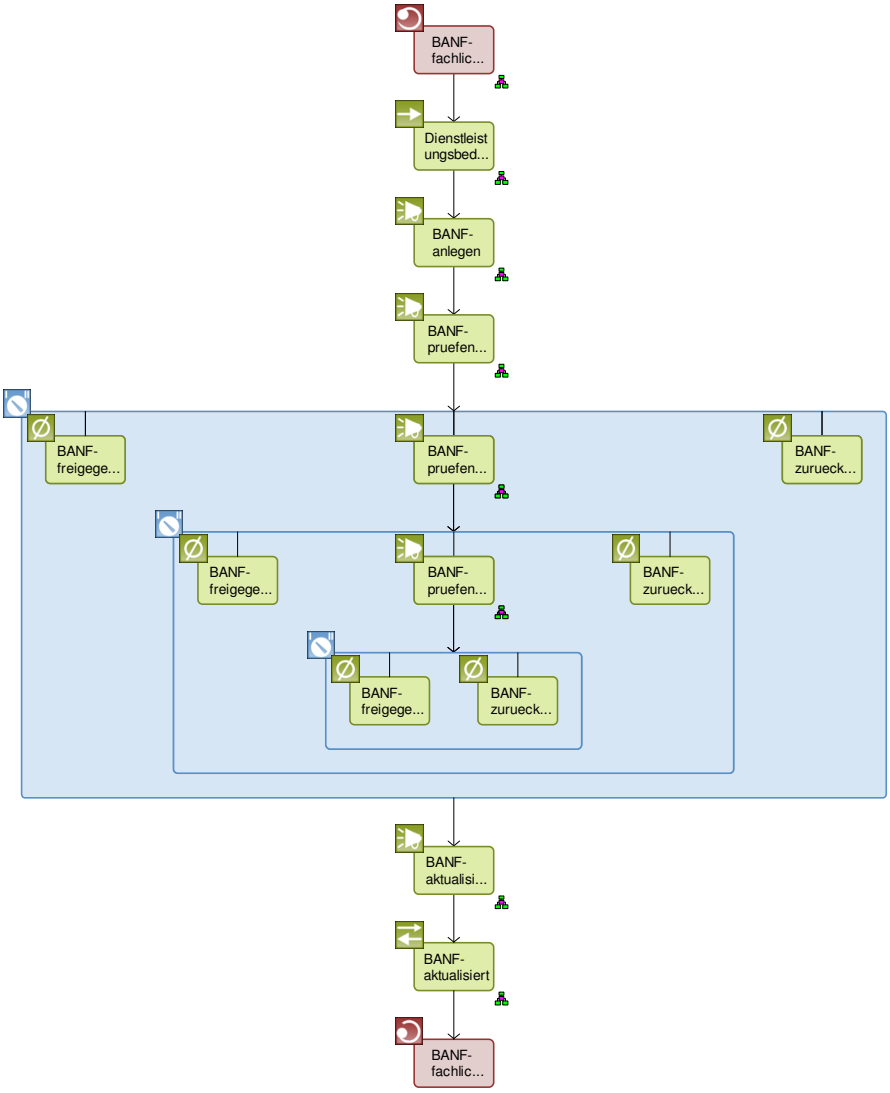

Abbildung I.6: Generierter BPEL-Code im ARIS SOA Architekt

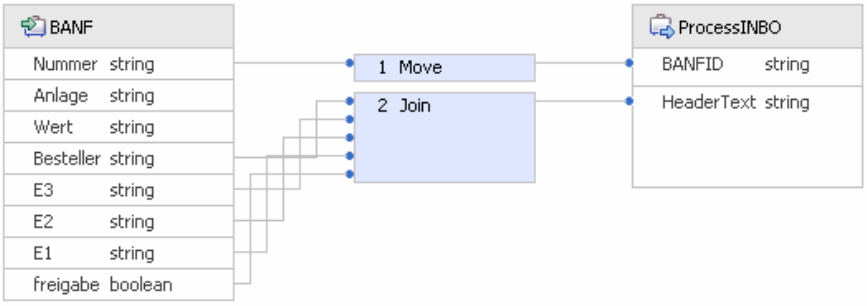

Abbildung I.7: Geschäftsobjektzuweisung

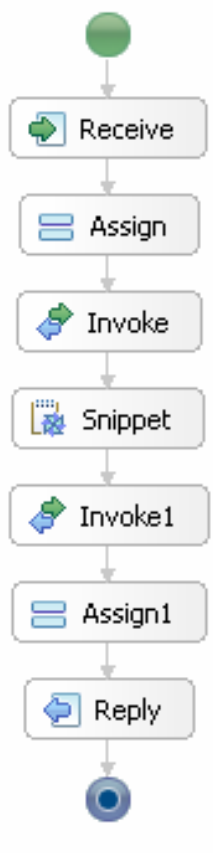

Abbildung I.8: Service in Form einer BPEL-Sequenz

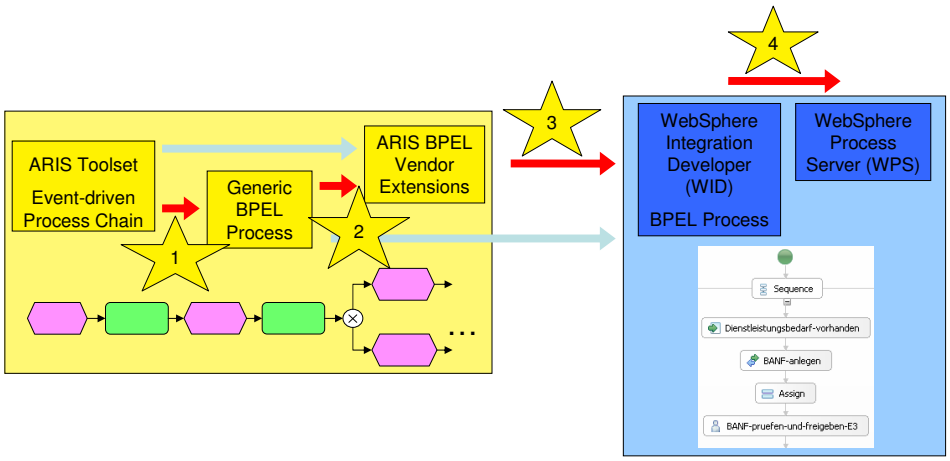

Abbildung I.9: Vorgehensweise im Entwicklungsprozess

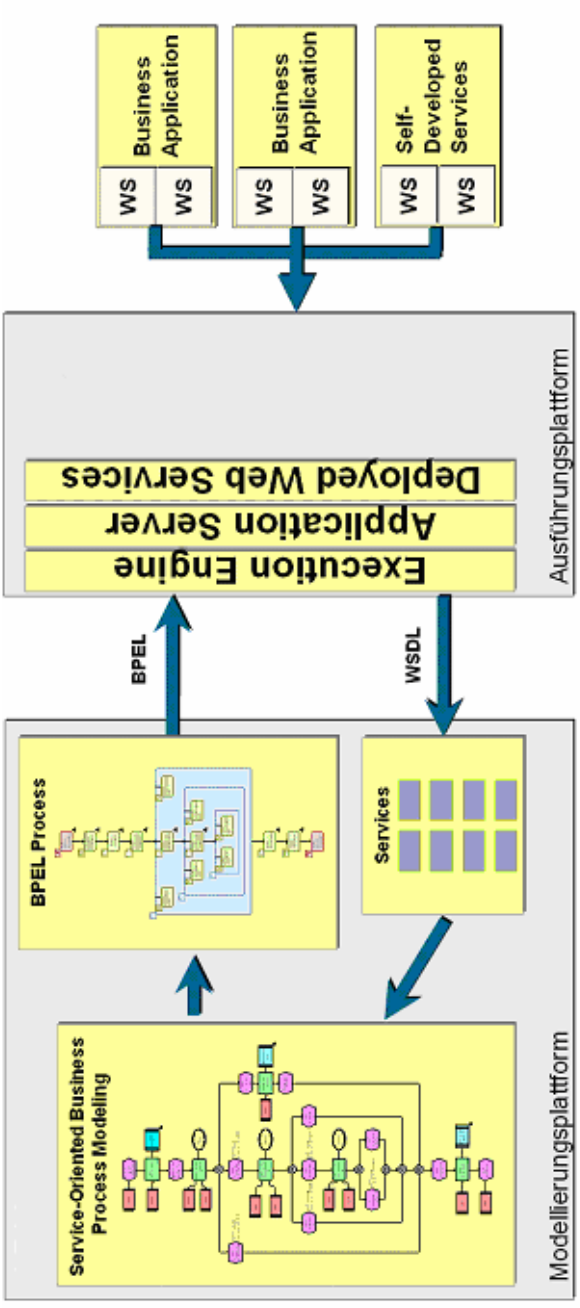

Abbildung I.10: Modellierung und Entwicklung

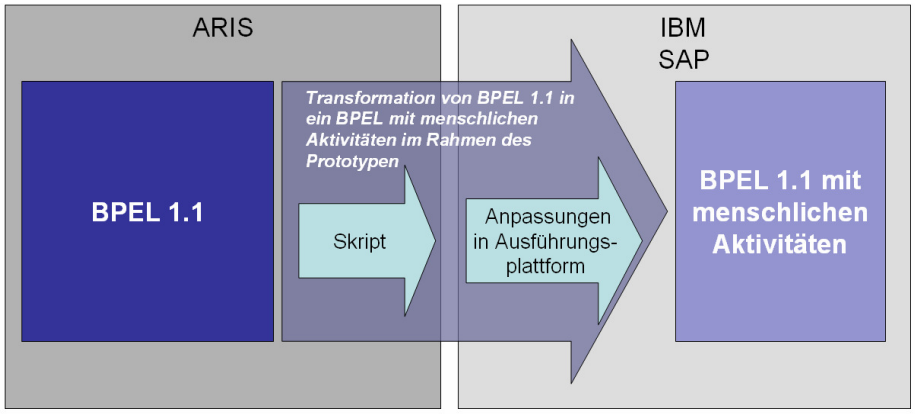

Abbildung I.11: Menschliche Aktivitäten

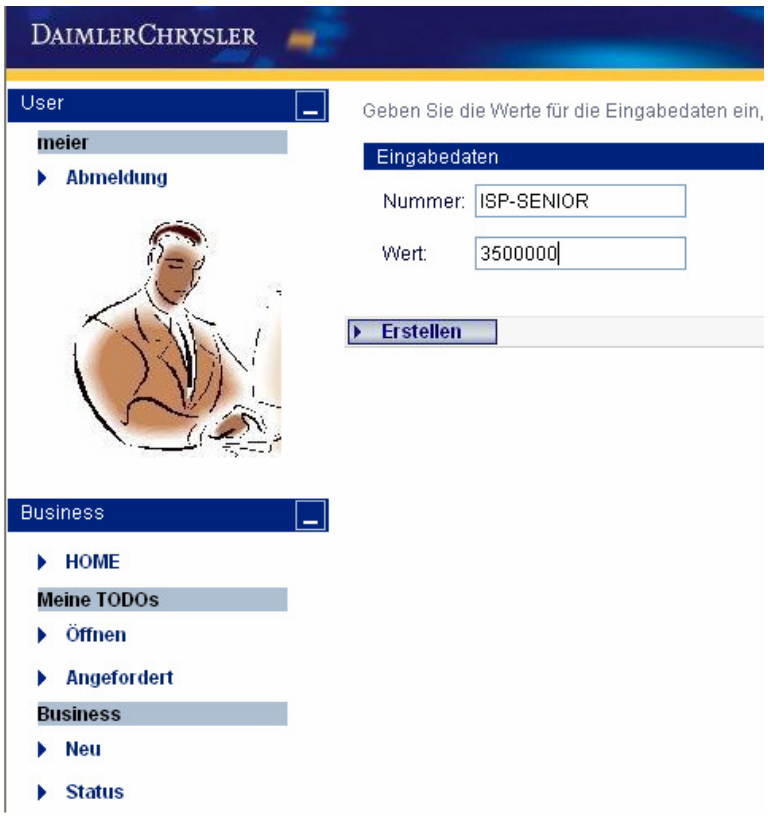

Abbildung I.12: Graphical User Interface

BITKOM

Kundenwettbewerb „Beste Prozessoptimierung mit SOA“
des Bundesverbandes Informationswirtschaft,
Telekommunikation und neue Medien e.V.

Bestes SOA-Industrieprojekt 2007
in der Kategorie „Cross Vendor“

für das Projekt:
„SOA Live Session“
des Teams „ITP/AM Methoden und Technologien“ der

Daimler AG

Karl-Heinz Streibich
Mitglied des Präsidiums BITKOM e.V.
Vorstandsvorsitzender Software AG

Dr. Mario Tobias
Mitglied der Geschäftsleitung
BITKOM e.V.

Berlin, 15. November 2007

accenture Alcatel-Lucent CISCO CONET EMDS EMPRISE empolis GFT hp invent
IDS SCHEER INTEC INTERSYSTEMS Microsoft ORACLE sage SAP SIEMENS software

Abbildung I.13: „Bestes SOA-Industrieprojekt 2007“, Kategorie „Cross-Vendor“

Anhang J

Übersicht Powertrain

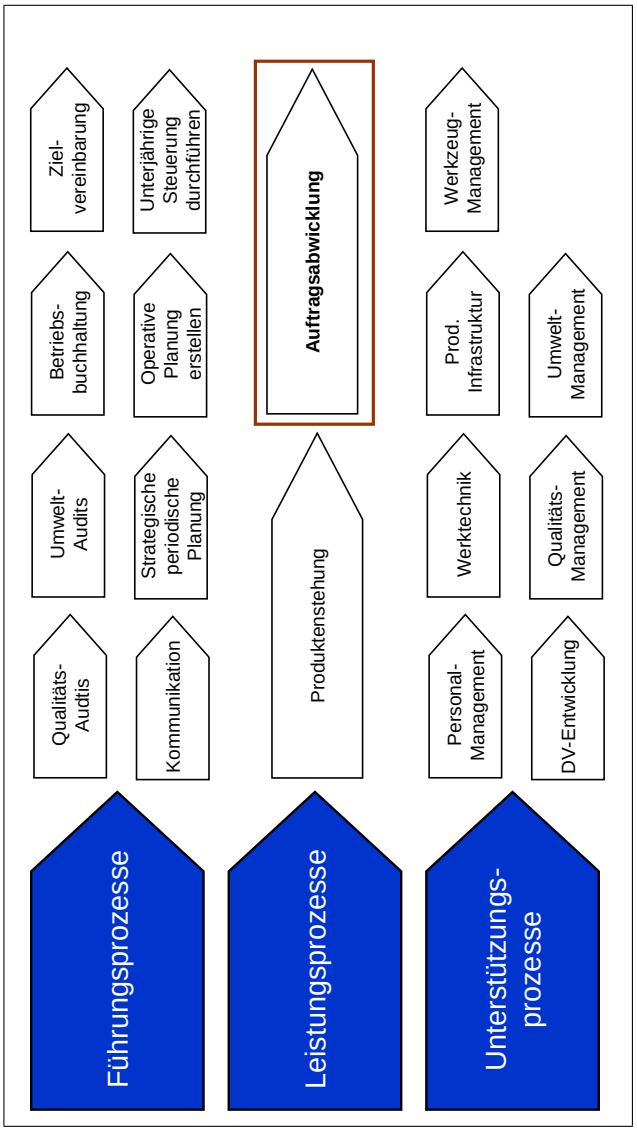

Abbildung J.1: Übersicht Prozesse *Powertrain*

Anhang K

Entwurfsmuster Mediator

K.1 Klasse Widget.java

```
package dcx.mediator;
import java.util.Calendar;
import javax.swing.JTextArea;

public abstract class Widget extends JTextArea{

   public abstract void calc(Calendar cal,
                             Integer imenge, Long isnr);

   public Widget (Mediator m, String s) {
      append(s + "\n");
      m.register(this);
   }
}
```

K.2 Klasse WidgetBZM.java

```
package dcx.mediator;

import java.util.Calendar;

public class WidgetBZM extends Widget {

   private static final long serialVersionUID = 1L;

   public WidgetBZM(Mediator med, String s) {
      super(med, s);
   }

   public void calc(Calendar cal, Integer imenge, Long isnr) {
      cal.add(Calendar.DAY_OF_WEEK,
      (int) ((-1 * (imenge.intValue()) / 40) *
             (isnr.doubleValue() % 7)));
   }
}
```

K.3 Klasse WidgetMASPlus.java

```
package dcx.mediator;

import java.util.Calendar;

public class WidgetMASPlus extends Widget {

   private static final long serialVersionUID = 1L;

   public WidgetMASPlus(Mediator med, String s) {
      super(med, s);
   }

   public void calc(Calendar cal, Integer imenge, Long isnr) {
      cal.add(Calendar.DAY_OF_WEEK,
      (int) (((imenge.intValue()) / 40)* -1 *
              (isnr.doubleValue() % 7)));
   }
}
```

K.4 Klasse Mediator.java

```
package dcx.mediator;

import java.text.SimpleDateFormat;
import java.util.*;

import javax.swing.JCheckBox;

public class Mediator {
   private ArrayList widgets = new ArrayList();
   private static String MASPLUS = "MASPlus";
   private static String BZM = "BZM";
   private JCheckBox masPlus;
   private JCheckBox bzm;
   private String lookUpValue;
   private SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd");
   private Calendar cal = Calendar.getInstance();

   private void setLookUpValue() {
      if (masPlus.isSelected()) {
         lookUpValue = MASPLUS;
      } else if (bzm.isSelected()) {
         lookUpValue = BZM;
      } else
         lookUpValue = "none.";
   }

   public String getLookUpValue() {
      return lookUpValue;
   }
```

```
   public void lookup() {
      setLookUpValue();
      System.out.println("Lookup ESR: " + getLookUpValue());
   }

   public void montage(Date dt, Integer imenge, Long lsnr) {

      Calendar c = Calendar.getInstance();
      c.setTime(dt);

      lookup();

      Widget widget = getWidget(getLookUpValue());

      if (widget != null) {
         call(c, imenge, lsnr, widget);
      }
   }

   private Widget getWidget(String lookUpValue) {
      Iterator iter = widgets.iterator();
      while (iter.hasNext()) {
         Widget widget = (Widget) iter.next();
         if (lookUpValue.equals(MASPLUS) &&
             widget instanceof WidgetMASPlus) {
            return widget;
         } else if (lookUpValue.equals(BZM) &&
                    widget instanceof WidgetBZM) {
            return widget;
         }
      }
      return null;
   }

   private void call(Calendar c, Integer imenge,
                     Long lsnr, Widget widget) {
      widget.calc(c, imenge, lsnr);
      widget.append("\n Soll-Termin: " +
                     df.format(c.getTime()));
   }

   public void register(Widget w) {
      widgets.add(w);
   }

   public void registerJBMASPlus(JCheckBox jb) {
      masPlus = jb;
   }

   public void registerJBBZM(JCheckBox jb) {
      bzm = jb;
   }
}
```

K.5 Klasse Main.java

```
package dcx.mediator;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

public class MedDemo extends JxFrame implements ActionListener
{
   JButton buttonLookup;
   JButton buttonService;
   Mediator med = new Mediator();
   Calendar cal = Calendar.getInstance();
   JTextField lft = new JTextField(10);
   JTextField mng = new JTextField("20", 10);
   JTextField snr = new JTextField("3500446212", 10);
   SimpleDateFormat df = new SimpleDateFormat( "yyyy-MM-dd" );

   private static String MASPLUS = "MASPlus";
   private static String BZM = "BZM";

   static public void main(String argv[])  {
      new MedDemo();
   }

   public MedDemo()
   {
      super("Prototyp: Mediator");

      JPanel jp = new JPanel();
      getContentPane().add(jp);
      jp.setLayout(new BorderLayout());
      JPanel center = new JPanel();

      JPanel left1 = new JPanel();
      JPanel right1 = new JPanel();
      JPanel left2 = new JPanel();
      JPanel right2 = new JPanel();

      left2.setBorder(new EmptyBorder(5,5,5,5));
      right2.setBorder(new EmptyBorder(5,5,5,5));
      left2.setLayout(new BorderLayout());
      right2.setLayout(new BorderLayout());

      center.setLayout(new GridLayout(2,2));
      center.add(left1);
      center.add(right1);
```

```
      center.add(left2);
      center.add(right2);

      left1.setLayout(new GridLayout(4,1));
      right1.setLayout(new GridLayout(4,1));

      left1.add(new Label("Sachnummer (SNR): ",Label.RIGHT));
      left1.add(new Label("Menge: ",Label.RIGHT));

      cal.add(Calendar.DAY_OF_WEEK, 10);
      lft.setText(df.format(cal.getTime()));
      left1.add(new Label("Lieferdatum (LFT): ",Label.RIGHT));

      right1.add(snr);
      right1.add(mng);
      right1.add(lft);

      buttonLookup = new JButton("Lookup");
      buttonLookup.addActionListener(this);
      buttonService = new JButton("getSollTermin");
      buttonService.addActionListener(this);

      JCheckBox masplus = new JCheckBox(MASPLUS, true);
      JCheckBox bzm = new JCheckBox(BZM, true);

      Box box = new Box(BoxLayout.X_AXIS);
      box.add(new JLabel("Enterprise Service Repository (ESR): "));
      box.add(masplus);
      box.add(bzm);

      WidgetMASPlus widgetMASPlus = new WidgetMASPlus(med, MASPLUS);
      WidgetBZM widgetBZM = new WidgetBZM(med, BZM);

      med.registerJBMASPlus(masplus);
      med.registerJBBZM(bzm);

      JPanel rtop = new JPanel();
      rtop.add(box);

      right1.add(buttonService);
      left1.add(buttonLookup);

      left2.add("Center", widgetMASPlus);
      right2.add("Center", widgetBZM);

      jp.add("Center", center);
      jp.add ("North", rtop);

      setSize(new Dimension(600,300));
      setVisible(true);
   }

   public void actionPerformed(ActionEvent e) {
      Date dt = null;
```

```
      Integer imenge = new Integer(mng.getText());
      Long lsnr      = new Long(snr.getText());
      try {
         dt = df.parse(lft.getText());
       } catch (ParseException e1) {
         e1.printStackTrace();
       }

      if (e.getSource()==buttonLookup) {
         med.Lookup();
      }
      if (e.getSource()==buttonService) {
         med.Montage( dt, imenge, lsnr);
      }
   }
}
```

K.6 Klasse JxFrame.java

```
package dcx.mediator;

import java.awt.event.*;
import javax.swing.*;

public class JxFrame extends JFrame
{
   public JxFrame(String title)   {
      super(title);
      setCloseClick();
      setLF();
   }
   private void setCloseClick()
   {
      addWindowListener(new WindowAdapter()
       {
    public void windowClosing(WindowEvent e) {System.exit(0);}
    });
   }
   private void setLF()    {
      String laf = UIManager.getSystemLookAndFeelClassName();
      try {
         UIManager.setLookAndFeel(laf);
      }
       catch (UnsupportedLookAndFeelException exc)
         {System.err.println("Warning: UnsupportedLookAndFeel: "
                             + laf);}
       catch (Exception exc) {System.err.println("Error loading "
                             + laf + ": " + exc);
      }
   }
}
```

Anhang L

Semantik-Service-Finder

L.1 Klasse Main.java

```
package dcx.hash;

import java.util.*;

public class Main {

   public static void Main(String[] args) throws Exception, Exception {
      HashMap mapMASPlusgetSollTermin = new HashMap();
    HashMap mapServicesInESR        = new HashMap();
    ESRServices serviceESR          = new ESRServices();
      RankService rankService         = new RankService();

      mapMASPlusgetSollTermin.put(serviceESR.SERVICE,"MASPlus");
      mapMASPlusgetSollTermin.put(serviceESR.OP,     "getSollTermin");
      mapMASPlusgetSollTermin.put(serviceESR.I,      "DAISachnummer;Menge;DAILiefertermin");
      mapMASPlusgetSollTermin.put(serviceESR.O,      "DAISollTermin");
      mapMASPlusgetSollTermin.put(serviceESR.P,      "MengePlausibel;Gear;SLK");
      mapMASPlusgetSollTermin.put(serviceESR.E,      "DAISollTerminVerfuegbar");

    if (serviceESR.readESR()) {
      mapServicesInESR = serviceESR.getNextServiceOperation();
    }

      rankService.connectToReasoner();
      rankService.doInference();

    do {
       rankService.printHeader(mapMASPlusgetSollTermin, mapServicesInESR);
         rankService.detectEquivalence(serviceESR.I, mapMASPlusgetSollTermin,
                                                     mapServicesInESR);
         rankService.detectEquivalence(serviceESR.O, mapMASPlusgetSollTermin,
                                                     mapServicesInESR);
         rankService.detectEquivalence(serviceESR.P, mapMASPlusgetSollTermin,
                                                     mapServicesInESR);
         rankService.detectEquivalence(serviceESR.E, mapMASPlusgetSollTermin,
                                                     mapServicesInESR);

         rankService.printFooter(serviceESR.I, serviceESR.O,
                                 serviceESR.P, serviceESR.E);
```

```
       mapServicesInESR = serviceESR.getNextServiceOperation();

  }  while (serviceESR.hasNextServiceOperation());

  rankService.printRanking();
   }
}
```

L.2 Klasse ESRServices.java

```
package dcx.hash;

import java.util.HashMap;

public class ESRServices {

   final String SERVICE = "Service";
   final String OP = "Operation";
   final String I = "INPUT";
   final String O = "OUTPUT";
   final String P = "PRECONDITION";
   final String E = "EFFECT";

   private final int MAXOperation = 4;

   private int hashCount = 0;

   private HashMap hashMap = new HashMap();

   private int getHashCount() {
      return hashCount;
   }
   private void setHashCount(int hashCount) {
      this.hashCount = hashCount;
   }

   public boolean readESR() {
      return true;
   }

   public HashMap getNextServiceOperation() {
      if (hasNextServiceOperation()) {
         setHashCount( getHashCount() + 1);
      }
      addServiceToHashMap();
      return hashMap;
   }

   public boolean hasNextServiceOperation() {
      if (getHashCount() <= MAXOperation) {
         return true;
      } else return false;
   }

   private void addServiceToHashMap() {
      if (getHashCount() == 1) {
        hashMap.clear();
```

```
    hashMap.put(SERVICE, "BZM");
    hashMap.put(OP,      "uebergebeDaten");
    hashMap.put(I,       "DCXLachnummer;Amount;DCXLieferTermine");
    hashMap.put(O,       "DCXIstTermin;Xmount");
    hashMap.put(P,       "AmountPlausibles");
    hashMap.put(E,       "DCXSollTerminVerfuegbarer");
     }
      if (getHashCount() == 2) {
    hashMap.put(SERVICE, "BZM");
    hashMap.put(OP,      "ermittleSollTermin");
    hashMap.put(I,       "DAISachnummer;Amount;DAILiefertermin");
    hashMap.put(O,       "DAISollTermin");
    hashMap.put(P,       "AmountPlausible;Aggregate;Roadster");
    hashMap.put(E,       "DAISollTerminVerfuegbar");
       }
      if (getHashCount() == 3) {
        hashMap.clear();
    hashMap.put(SERVICE, "CrtService");
    hashMap.put(OP,      "getPartOwner");
    hashMap.put(I,       "getPartOwner");
    hashMap.put(O,       "getPartOwnerResponse");
    hashMap.put(P,       "DCXPartOwner");
    hashMap.put(E,       "DCXPartOwnerUpdated");
     }
      if (getHashCount() == 4) {
        hashMap.clear();
      hashMap.put(SERVICE, "CrtPlusService");
      hashMap.put(OP,      "getCustomer");
      hashMap.put(I,       "getCustomer");
      hashMap.put(O,       "getCustomerResponse");
      hashMap.put(P,       "AmountPlausible");
      hashMap.put(E,       "DCXSollTerminVerfuegbar");
      }

   }
}
```

L.3 Klasse RankServices.java

```
package dcx.hash;

import edu.stanford.smi.protegex.owl.*;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.text.DecimalFormat;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class RankService {

   final String RANKING      = "Ranking";
   final String ONTOLOGY_URL = "D:/Projekte/_Promotion/owl/dcx.owl";
   final String REASONER_URL = "http://localhost:8080";
```

```
final String DELIMITER    = ";";
final String SERVICE      = "Service";
final String OP           = "Operation";

private double             serviceFragment;
private double             serviceFragmentCount;
private boolean            fragmentRanked;
private OWLModel           model;
private ProtegeOWLReasoner reasoner;
private String             sIOPE;

private HashMap rankedServices       = new HashMap();
private HashMap rankServiceOperation = new HashMap();
private HashMap mapMap = new HashMap();
private HashMap mapServiceESR;
private DecimalFormat myFormatter = new DecimalFormat("0000");

private boolean isFragmentRanked() {
   return fragmentRanked;
}

private void setFragmentRanked(boolean fragmentRanked) {
   this.fragmentRanked = fragmentRanked;
}

private void calcPercent(String sIOPE) {
   String sFrag      = String.valueOf(getRankServiceFragment());
   String sFragCount = String.valueOf(getServiceFragmentCount());
   String sKey       = (String) mapServiceESR.get(SERVICE) + "."
                     + (String) mapServiceESR.get(OP);

   Integer iErgebnis = new Integer ( (int) Math.round(
      ( Double.parseDouble(sFrag) / Double.parseDouble(sFragCount) )
      * 100) );

   rankServiceOperation.put(sIOPE, sFrag + DELIMITER +
                            sFragCount + DELIMITER + iErgebnis);
   this.mapServiceESR.put(RANKING, rankServiceOperation);

   if (rankedServices.get(sKey) == null) {
      rankedServices.put( sKey, "0");
   }

   double dRanking =  Double.parseDouble( (String) rankedServices.get(sKey) ) +
                      Double.parseDouble( (String) iErgebnis.toString() );
   String sRanking = myFormatter.format(dRanking);
   rankedServices.put( sKey, sRanking);
}

private double getServiceFragmentCount() {
   return serviceFragmentCount;
}

private void setServiceFragmentCount(double serviceFragmentCount) {
   this.serviceFragmentCount = serviceFragmentCount;
}

private void addRank(double i) {
```

```
      setServiceFragment( getRankServiceFragment() + i);
   }

   private void setServiceFragment(double i) {
      serviceFragment = i;
   }

   public void printRanking() {
      String[] values             = new String[rankedServices.size()];
      Collection collectionValues = rankedServices.values();
      Collection collectionKeys   = rankedServices.keySet();
      Iterator itKeys             = collectionKeys.iterator();
      Iterator itValues           = collectionValues.iterator();

      int j=0;
      for( ; itKeys.hasNext() ; ) {
         values[j] = (String) itValues.next() + "\t" + (String) itKeys.next();
         j++;
      }
      Arrays.sort(values);

      System.out.println("Äquivalente Serviceoperationen (Kandidaten):");
      System.out.println("=========================================\n");
      System.out.println("Bewertung\tServiceoperation");
      System.out.println("-----------------------------------------");
      do {
         System.out.println(values[j-1]);
         j--;
      } while (j>0);

      //System.out.println(mapMap);
   }

   public boolean connectToReasoner() throws FileNotFoundException, Exception {
      model = ProtegeOWL.createJenaOWLModelFromReader(
         new BufferedReader(new FileReader(ONTOLOGY_URL)));
      ReasonerManager reasonerManager = ReasonerManager.getInstance();
      reasoner = reasonerManager.getReasoner(model);
      reasoner.setURL(REASONER_URL);

      return true;
   }

   public boolean doInference() throws DIGReasonerException {
      if(reasoner.isConnected()) {
          DIGReasonerIdentity reasonerIdentity = reasoner.getIdentity();
          System.out.print("Connected to " + reasonerIdentity.getName() +
                           ". Classifying ...");
          reasoner.classifyTaxonomy(null);
          System.out.println("  done.\n");
          return true;
       }
          else return false;
   }

   public double getRankServiceFragment() {
      return serviceFragment;
   }
```

```
  public void printFooter(String I, String O, String P, String E) {
     HashMap hashMap = new HashMap();
     hashMap.putAll((Map) this.mapServiceESR.get(RANKING));

     System.out.println(I + "=" + hashMap.get(I) + "%  " +
                   O + "=" + hashMap.get(O) + "%  " +
                   P + "=" + hashMap.get(P) + "%  " +
                   E + "=" + hashMap.get(E) + "%  \n"
                  );
  }

  public void printHeader(HashMap mapService, HashMap mapServiceESR) {
 System.out.println(mapService.get("Service")  + "." + mapService.get("Operation") +
" ==> " + mapServiceESR.get("Service") + "." + mapServiceESR.get("Operation") + "\n"
          );
  }

  private void printCollection(Collection collection,
                               String frag, String type, double ranking) {
     System.out.print(type + " of " + frag + "(s):  " + collection.size());

     for(Iterator it = collection.iterator(); it.hasNext();) {
        OWLNamedClass curClass = (OWLNamedClass) it.next();
        rankFrag(ranking, curClass.getName(), frag);
     }
     System.out.println("");
  }

  private void rankFrag(double ranking, String curClass, String frag) {
     String sInputESR   = (String) this.mapServiceESR.get(sIOPE);
   String sSplitESR[] = sInputESR.split(DELIMITER);

     if ( ! isFragmentRanked() ) {
        for (int j=0 ; j < sSplitESR.length; j++ ){
   if (curClass.equals(sSplitESR[j]) || frag.equals(sSplitESR[j])) {
     addRank(ranking);
     System.out.print("(ADD " + ranking + ")");

           if (mapServiceESR.get(SERVICE).equals("BZM") &&
               mapServiceESR.get(OP).equals("ermittleSollTermin")) {
              String sKey = mapServiceESR.get(SERVICE) + "." +
                            mapServiceESR.get(OP) + "." + sIOPE;
              mapMap.put(sKey, curClass + "-->" +sSplitESR[j]);
           }

       setFragmentRanked(true);
      j = sSplitESR.length;   }
       }
     }
  }

  public void detectEquivalence(final String sIOPE, HashMap mapService,
     HashMap mapServiceESR) throws DIGReasonerException {
    this.mapServiceESR = mapServiceESR;
    this.sIOPE = sIOPE;

     System.out.println(sIOPE);
```

```
      String sInput = (String) mapService.get(sIOPE);
      String sSplit[] = sInput.split(DELIMITER);

      setServiceFragmentCount(sSplit.length);
      setServiceFragment(0);

      for (int i=0 ; i < sSplit.length; i++ ){
         OWLNamedClass namedClass             = model.getOWLNamedClass(sSplit[i]);

         Collection inferredSubclasses        = namedClass.getInferredSubclasses();
         Collection inferredEquivalentClasses = namedClass.getInferredEquivalentClasses();
         Collection inferredSuperClasses      = namedClass.getInferredSuperclasses();

     setFragmentRanked(false);
         printCollection(inferredEquivalentClasses, sSplit[i],
                         "inferredEquivalentClasses", 1);
         printCollection(inferredSuperClasses,      sSplit[i],
                         "inferredSuperClasses     ", 0.5);
         printCollection(inferredSubclasses,        sSplit[i],
                         "inferredSubclasses       ", 0.5);

         System.out.print("\n");
      }
      calcPercent(sIOPE);
   }
}
```

L.4 Inferenz in Protégé

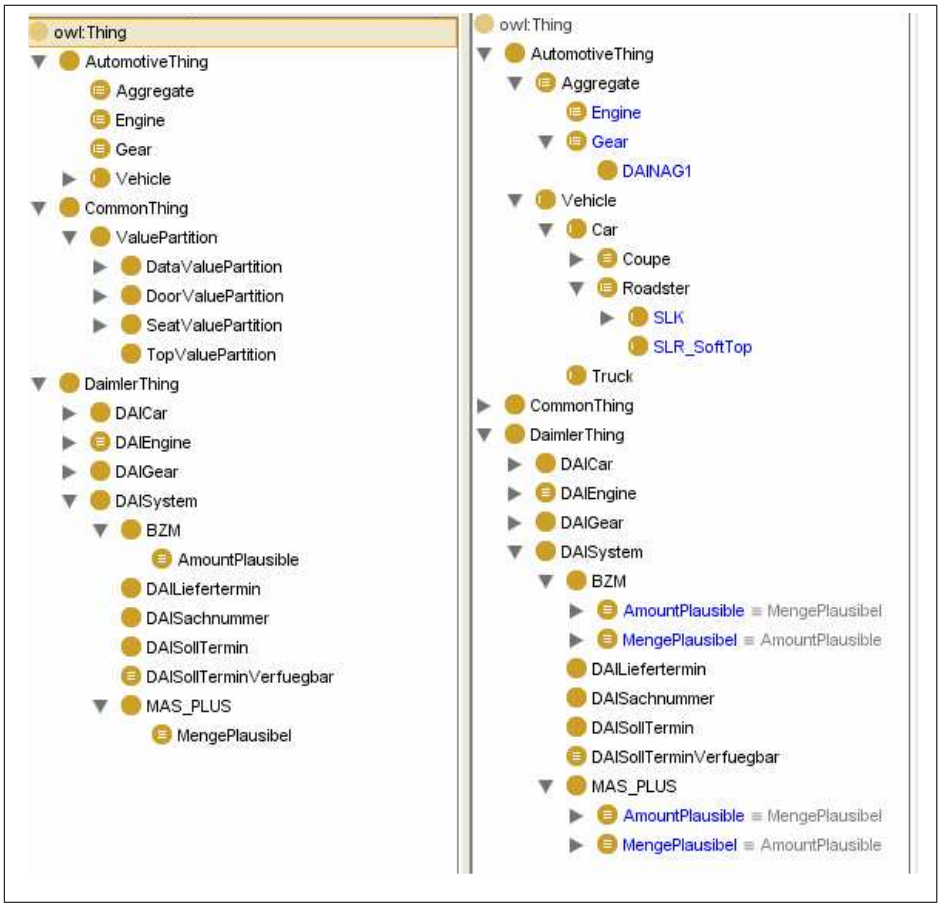

Abbildung L.1: Inferenz via Inferenzmaschine RACER [85] in Protégé [114] (1/2)

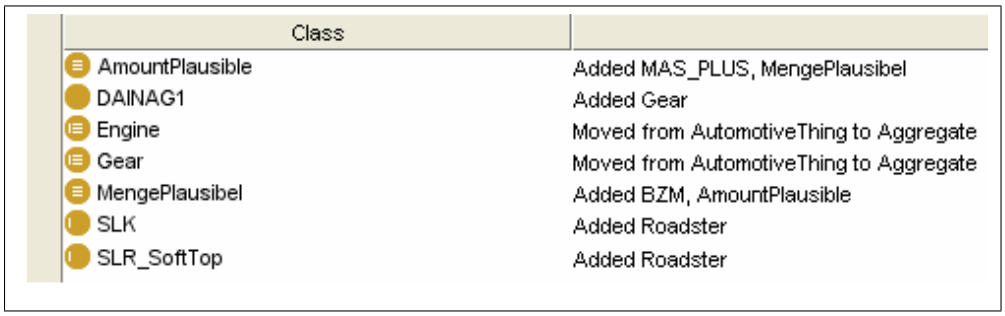

Abbildung L.2: Inferenz via Inferenzmaschine RACER [85] in Protégé [114] (2/2)

Anhang M

SSF Konsolenausgabe

```
INFO: Loading triples
INFO: [ProtegeOWLParser] Completed triple loading after 703 ms
INFO: [TripleChangePostProcessor] Completed lists after 16 ms
INFO: [TripleChangePostProcessor] Completed anonymous classes after 0 ms
INFO: [TripleChangePostProcessor] Completed deprecated classes after 0 ms
INFO: [TripleChangePostProcessor] Completed properties after 15 ms
INFO: [TripleChangePostProcessor] Completed named classes after 16 ms
INFO: ... Loading completed after 812 ms
Connected to Racer. Classifying ...  done.

MASPlus.getSollTermin ==> BZM.ermittleSollTermin

INPUT
equalClasses (candidates) of DAISachnummer(s):  3 (ADD 1.0)
inferredEquivalentClasses of DAISachnummer(s):  0
inferredSuperClasses      of DAISachnummer(s):  1
inferredSubclasses        of DAISachnummer(s):  0

equalClasses (candidates) of Menge(s):  3
inferredEquivalentClasses of Menge(s):  1 (ADD 1.0)
inferredSuperClasses      of Menge(s):  2
inferredSubclasses        of Menge(s):  1

equalClasses (candidates) of DAILiefertermin(s):  3 (ADD 1.0)
inferredEquivalentClasses of DAILiefertermin(s):  0
inferredSuperClasses      of DAILiefertermin(s):  1
inferredSubclasses        of DAILiefertermin(s):  0

OUTPUT
equalClasses (candidates) of DAISollTermin(s):  1 (ADD 1.0)
inferredEquivalentClasses of DAISollTermin(s):  0
inferredSuperClasses      of DAISollTermin(s):  1
inferredSubclasses        of DAISollTermin(s):  0

PRECONDITION
equalClasses (candidates) of MengePlausibel(s):  3
inferredEquivalentClasses of MengePlausibel(s):  1 (ADD 1.0)
inferredSuperClasses      of MengePlausibel(s):  3
inferredSubclasses        of MengePlausibel(s):  1

equalClasses (candidates) of Gear(s):  3
inferredEquivalentClasses of Gear(s):  0
```

```
inferredSuperClasses      of Gear(s):  1 (ADD 0.5)
inferredSubclasses        of Gear(s):  1

equalClasses (candidates) of SLK(s):  3
inferredEquivalentClasses of SLK(s):  0
inferredSuperClasses      of SLK(s):  2 (ADD 0.5)
inferredSubclasses        of SLK(s):  2

EFFECT
equalClasses (candidates) of DAISollTerminVerfuegbar(s):  1 (ADD 1.0)
inferredEquivalentClasses of DAISollTerminVerfuegbar(s):  0
inferredSuperClasses      of DAISollTerminVerfuegbar(s):  1
inferredSubclasses        of DAISollTerminVerfuegbar(s):  0

INPUT=3.0;3.0;100%  OUTPUT=1.0;1.0;100%  PRECONDITION=2.0;3.0;67%  EFFECT=1.0;1.0;100%

----- Vorschlag zur Abbildung der Schnittstellen ---------------
I: | DAISachnummer ===> DAISachnummer |
   | Menge ===> Amount |
   | DAILiefertermin ===> DAILiefertermin |
O: | DAISollTermin ===> DAISollTermin |
----- Abbildung der Konzepte ----------------------------------
P: | MengePlausibel ===> AmountPlausible |
   | Gear ---> Aggregate |
   | SLK ---> Roadster |
E: | DAISollTerminVerfuegbar ===> DAISollTerminVerfuegbar |
-----------------------------------------------------------------


MASPlus.getSollTermin ==> BZM.uebergebeDaten

INPUT
equalClasses (candidates) of DAISachnummer(s):  3
inferredEquivalentClasses of DAISachnummer(s):  0
inferredSuperClasses      of DAISachnummer(s):  1
inferredSubclasses        of DAISachnummer(s):  0

equalClasses (candidates) of Menge(s):  3
inferredEquivalentClasses of Menge(s):  1 (ADD 1.0)
inferredSuperClasses      of Menge(s):  2
inferredSubclasses        of Menge(s):  1

equalClasses (candidates) of DAILiefertermin(s):  3
inferredEquivalentClasses of DAILiefertermin(s):  0
inferredSuperClasses      of DAILiefertermin(s):  1
inferredSubclasses        of DAILiefertermin(s):  0

OUTPUT
equalClasses (candidates) of DAISollTermin(s):  2
inferredEquivalentClasses of DAISollTermin(s):  0
inferredSuperClasses      of DAISollTermin(s):  1
inferredSubclasses        of DAISollTermin(s):  0

PRECONDITION
equalClasses (candidates) of MengePlausibel(s):  1
inferredEquivalentClasses of MengePlausibel(s):  1
inferredSuperClasses      of MengePlausibel(s):  3
inferredSubclasses        of MengePlausibel(s):  1
```

```
equalClasses (candidates) of Gear(s):  1
inferredEquivalentClasses of Gear(s):  0
inferredSuperClasses      of Gear(s):  1
inferredSubclasses        of Gear(s):  1

equalClasses (candidates) of SLK(s):  1
inferredEquivalentClasses of SLK(s):  0
inferredSuperClasses      of SLK(s):  2
inferredSubclasses        of SLK(s):  2

EFFECT
equalClasses (candidates) of DAISollTerminVerfuegbar(s):  1
inferredEquivalentClasses of DAISollTerminVerfuegbar(s):  0
inferredSuperClasses      of DAISollTerminVerfuegbar(s):  1
inferredSubclasses        of DAISollTerminVerfuegbar(s):  0

INPUT=1.0;3.0;33%  OUTPUT=0.0;1.0;0%  PRECONDITION=0.0;3.0;0%  EFFECT=0.0;1.0;0%

----- Vorschlag zur Abbildung der Schnittstellen ---------------
I: | Menge ===> Amount |
O: |
----- Abbildung der Konzepte ----------------------------------
P: |
E: |
-----------------------------------------------------------------


MASPlus.getSollTermin ==> CrtService.getPartOwner

INPUT
equalClasses (candidates) of DAISachnummer(s):  1
inferredEquivalentClasses of DAISachnummer(s):  0
inferredSuperClasses      of DAISachnummer(s):  1
inferredSubclasses        of DAISachnummer(s):  0

equalClasses (candidates) of Menge(s):  1
inferredEquivalentClasses of Menge(s):  1
inferredSuperClasses      of Menge(s):  2
inferredSubclasses        of Menge(s):  1

equalClasses (candidates) of DAILiefertermin(s):  1
inferredEquivalentClasses of DAILiefertermin(s):  0
inferredSuperClasses      of DAILiefertermin(s):  1
inferredSubclasses        of DAILiefertermin(s):  0

OUTPUT
equalClasses (candidates) of DAISollTermin(s):  1
inferredEquivalentClasses of DAISollTermin(s):  0
inferredSuperClasses      of DAISollTermin(s):  1
inferredSubclasses        of DAISollTermin(s):  0

PRECONDITION
equalClasses (candidates) of MengePlausibel(s):  1
inferredEquivalentClasses of MengePlausibel(s):  1
inferredSuperClasses      of MengePlausibel(s):  3
inferredSubclasses        of MengePlausibel(s):  1
```

```
equalClasses (candidates) of Gear(s):  1
inferredEquivalentClasses of Gear(s):  0
inferredSuperClasses      of Gear(s):  1
inferredSubclasses        of Gear(s):  1

equalClasses (candidates) of SLK(s):  1
inferredEquivalentClasses of SLK(s):  0
inferredSuperClasses      of SLK(s):  2
inferredSubclasses        of SLK(s):  2

EFFECT
equalClasses (candidates) of DAISollTerminVerfuegbar(s):  1
inferredEquivalentClasses of DAISollTerminVerfuegbar(s):  0
inferredSuperClasses      of DAISollTerminVerfuegbar(s):  1
inferredSubclasses        of DAISollTerminVerfuegbar(s):  0

INPUT=0.0;3.0;0%  OUTPUT=0.0;1.0;0%  PRECONDITION=0.0;3.0;0%  EFFECT=0.0;1.0;0%

----- Vorschlag zur Abbildung der Schnittstellen ---------------
I: |
O: |
----- Abbildung der Konzepte -----------------------------------
P: |
E: |
----------------------------------------------------------------


MASPlus.getSollTermin ==> CrtPlusService.getCustomer

INPUT
equalClasses (candidates) of DAISachnummer(s):  1
inferredEquivalentClasses of DAISachnummer(s):  0
inferredSuperClasses      of DAISachnummer(s):  1
inferredSubclasses        of DAISachnummer(s):  0

equalClasses (candidates) of Menge(s):  1
inferredEquivalentClasses of Menge(s):  1
inferredSuperClasses      of Menge(s):  2
inferredSubclasses        of Menge(s):  1

equalClasses (candidates) of DAILiefertermin(s):  1
inferredEquivalentClasses of DAILiefertermin(s):  0
inferredSuperClasses      of DAILiefertermin(s):  1
inferredSubclasses        of DAILiefertermin(s):  0

OUTPUT
equalClasses (candidates) of DAISollTermin(s):  1
inferredEquivalentClasses of DAISollTermin(s):  0
inferredSuperClasses      of DAISollTermin(s):  1
inferredSubclasses        of DAISollTermin(s):  0

PRECONDITION
equalClasses (candidates) of MengePlausibel(s):  1
inferredEquivalentClasses of MengePlausibel(s):  1 (ADD 1.0)
inferredSuperClasses      of MengePlausibel(s):  3
inferredSubclasses        of MengePlausibel(s):  1

equalClasses (candidates) of Gear(s):  1
```

```
inferredEquivalentClasses of Gear(s):  0
inferredSuperClasses      of Gear(s):  1
inferredSubclasses        of Gear(s):  1

equalClasses (candidates) of SLK(s):  1
inferredEquivalentClasses of SLK(s):  0
inferredSuperClasses      of SLK(s):  2
inferredSubclasses        of SLK(s):  2

EFFECT
equalClasses (candidates) of DAISollTerminVerfuegbar(s):  1
inferredEquivalentClasses of DAISollTerminVerfuegbar(s):  0
inferredSuperClasses      of DAISollTerminVerfuegbar(s):  1
inferredSubclasses        of DAISollTerminVerfuegbar(s):  0

INPUT=0.0;3.0;0%  OUTPUT=0.0;1.0;0%  PRECONDITION=1.0;3.0;33%  EFFECT=0.0;1.0;0%

----- Vorschlag zur Abbildung der Schnittstellen ---------------
I: |
O: |
----- Abbildung der Konzepte -----------------------------------
P: | MengePlausibel ===> AmountPlausible |
E: |
------------------------------------------------------------------


Äquivalente Serviceoperationen (Kandidaten):
============================================

Bewertung Serviceoperation
------------------------------------------
0367 BZM.ermittleSollTermin
0033 CrtPlusService.getCustomer
0033 BZM.uebergebeDaten
0000 CrtService.getPartOwner
```

Anhang N

Ontologie in OWL

```
?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY protege "http://protege.stanford.edu/plugins/owl/protege#" >
]>

<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1181215061.owl#"
     xml:base="http://www.owl-ontologies.com/Ontology1181215061.owl"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#">
    <owl:Ontology rdf:about=""/>
    <owl:AllDifferent>
        <owl:distinctMembers rdf:parseType="Collection"/>
    </owl:AllDifferent>
    <owl:Class rdf:ID="Aggregate">
        <owl:equivalentClass>
            <owl:Class>
                <owl:unionOf rdf:parseType="Collection">
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#isAssembledIn"/>
                        <owl:someValuesFrom rdf:resource="#Vehicle"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#machtesTo"/>
                        <owl:someValuesFrom rdf:resource="#Gear"/>
                    </owl:Restriction>
                </owl:unionOf>
            </owl:Class>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#AutomotiveThing"/>
        <owl:disjointWith rdf:resource="#Vehicle"/>
        <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
    </owl:Class>
    <owl:Class rdf:ID="Amount">
        <owl:equivalentClass>
```

```
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isSameAs"/>
                <owl:someValuesFrom rdf:resource="#Menge"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#DataValuePartition"/>
    </owl:Class>
    <owl:Class rdf:ID="AmountPlausible">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isValid"/>
                <owl:someValuesFrom rdf:resource="#Amount"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#BZM"/>
    </owl:Class>
    <owl:Class rdf:ID="AutomotiveThing"/>
    <owl:Class rdf:ID="backSeat">
        <rdfs:subClassOf rdf:resource="#SeatValuePartition"/>
        <owl:disjointWith rdf:resource="#leftSeat"/>
        <owl:disjointWith rdf:resource="#rightSeat"/>
    </owl:Class>
    <owl:Class rdf:ID="BZM">
        <rdfs:subClassOf rdf:resource="#DAISystem"/>
    </owl:Class>
    <owl:Class rdf:ID="Car">
        <rdfs:subClassOf rdf:resource="#Vehicle"/>
        <owl:disjointWith rdf:resource="#Truck"/>
        <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
    </owl:Class>
    <owl:Class rdf:ID="CDI200">
        <rdfs:subClassOf rdf:resource="#DAIEngine"/>
        <owl:disjointWith rdf:resource="#CDI220"/>
        <owl:disjointWith rdf:resource="#Compressor200"/>
        <owl:disjointWith rdf:resource="#Compressor230"/>
        <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
    </owl:Class>
    <owl:Class rdf:ID="CDI220">
        <rdfs:subClassOf rdf:resource="#DAIEngine"/>
        <owl:disjointWith rdf:resource="#CDI200"/>
        <owl:disjointWith rdf:resource="#Compressor200"/>
        <owl:disjointWith rdf:resource="#Compressor230"/>
        <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
    </owl:Class>
    <owl:Class rdf:ID="CommonThing"/>
    <owl:Class rdf:ID="Compressor200">
        <rdfs:subClassOf rdf:resource="#DAIEngine"/>
        <owl:disjointWith rdf:resource="#CDI200"/>
        <owl:disjointWith rdf:resource="#CDI220"/>
        <owl:disjointWith rdf:resource="#Compressor230"/>
        <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
    </owl:Class>
    <owl:Class rdf:ID="Compressor230">
        <rdfs:subClassOf rdf:resource="#DAIEngine"/>
        <owl:disjointWith rdf:resource="#CDI200"/>
        <owl:disjointWith rdf:resource="#CDI220"/>
        <owl:disjointWith rdf:resource="#Compressor200"/>
        <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
```

```
</owl:Class>
<owl:Class rdf:ID="Coupe">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasDoorway"/>
                    <owl:cardinality rdf:datatype="&xsd;int">2</owl:cardinality>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasTop"/>
                    <owl:hasValue rdf:resource="#topHardTop"/>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Car"/>
    <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">false</protege:subclassesDisjoint>
    <rdfs:comment rdf:datatype="&xsd;string"
        >hasSeat = 2 fehlt</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="DAICar">
    <rdfs:subClassOf rdf:resource="#DaimlerThing"/>
    <owl:disjointWith rdf:resource="#DAIGear"/>
    <owl:disjointWith rdf:resource="#DAIEngine"/>
</owl:Class>
<owl:Class rdf:ID="DAIEngine">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#machtesTo"/>
            <owl:someValuesFrom rdf:resource="#DAICar"/>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#DaimlerThing"/>
    <owl:disjointWith rdf:resource="#DAICar"/>
    <owl:disjointWith rdf:resource="#DAIGear"/>
</owl:Class>
<owl:Class rdf:ID="DAIGear">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isAssembledIn"/>
            <owl:allValuesFrom rdf:resource="#DAICar"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#DaimlerThing"/>
    <owl:disjointWith rdf:resource="#DAICar"/>
    <owl:disjointWith rdf:resource="#DAIEngine"/>
</owl:Class>
<owl:Class rdf:ID="DAILiefertermin">
    <rdfs:subClassOf rdf:resource="#DAISystem"/>
    <owl:disjointWith rdf:resource="#DAISachnummer"/>
    <owl:disjointWith rdf:resource="#DAISollTermin"/>
</owl:Class>
<owl:Class rdf:ID="DaimlerThing"/>
<owl:Class rdf:ID="DAINAG1">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isAssembledIn"/>
```

```
            <owl:someValuesFrom rdf:resource="#SLK230"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isAssembledIn"/>
            <owl:someValuesFrom rdf:resource="#SLK200"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#DAIGear"/>
</owl:Class>
<owl:Class rdf:ID="DAISachnummer">
    <rdfs:subClassOf rdf:resource="#DAISystem"/>
    <owl:disjointWith rdf:resource="#DAILiefertermin"/>
    <owl:disjointWith rdf:resource="#DAISollTermin"/>
</owl:Class>
<owl:Class rdf:ID="DAISollTermin">
    <rdfs:subClassOf rdf:resource="#DAISystem"/>
    <owl:disjointWith rdf:resource="#DAILiefertermin"/>
    <owl:disjointWith rdf:resource="#DAISachnummer"/>
</owl:Class>
<owl:Class rdf:ID="DAISollTerminVerfuegbar">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isValid"/>
            <owl:someValuesFrom rdf:resource="#DAISollTermin"/>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#DAISystem"/>
</owl:Class>
<owl:Class rdf:ID="DAISystem">
    <rdfs:subClassOf rdf:resource="#DaimlerThing"/>
</owl:Class>
<owl:Class rdf:ID="DataValuePartition">
    <rdfs:subClassOf rdf:resource="#ValuePartition"/>
</owl:Class>
<owl:Class rdf:ID="DoorValuePartition">
    <rdfs:subClassOf rdf:resource="#ValuePartition"/>
</owl:Class>
<owl:Class rdf:ID="Engine">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#machtesTo"/>
            <owl:someValuesFrom rdf:resource="#Gear"/>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#AutomotiveThing"/>
    <owl:disjointWith rdf:resource="#Gear"/>
    <owl:disjointWith rdf:resource="#Vehicle"/>
    <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
</owl:Class>
<owl:Class rdf:ID="Gear">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isAssembledIn"/>
            <owl:someValuesFrom rdf:resource="#Vehicle"/>
        </owl:Restriction>
    </owl:equivalentClass>
```

```
    <rdfs:subClassOf rdf:resource="#AutomotiveThing"/>
    <owl:disjointWith rdf:resource="#Engine"/>
    <owl:disjointWith rdf:resource="#Vehicle"/>
    <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasDoorway">
    <rdfs:range rdf:resource="#DoorValuePartition"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasEngine">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:range rdf:resource="#Engine"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasSeat">
    <rdfs:range rdf:resource="#SeatValuePartition"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasTop">
    <rdfs:range rdf:resource="#TopValuePartition"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isAssembledIn">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isSameAs">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isValid">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="leftDoorBack">
    <rdfs:subClassOf rdf:resource="#DoorValuePartition"/>
    <owl:disjointWith rdf:resource="#leftDoorFront"/>
    <owl:disjointWith rdf:resource="#rightDoorBack"/>
    <owl:disjointWith rdf:resource="#rightDoorFront"/>
</owl:Class>
<owl:Class rdf:ID="leftDoorFront">
    <rdfs:subClassOf rdf:resource="#DoorValuePartition"/>
    <owl:disjointWith rdf:resource="#leftDoorBack"/>
    <owl:disjointWith rdf:resource="#rightDoorBack"/>
    <owl:disjointWith rdf:resource="#rightDoorFront"/>
</owl:Class>
<owl:Class rdf:ID="leftSeat">
    <rdfs:subClassOf rdf:resource="#SeatValuePartition"/>
    <owl:disjointWith rdf:resource="#backSeat"/>
    <owl:disjointWith rdf:resource="#rightSeat"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="machtesTo">
    <rdf:type rdf:resource="&owl;SymmetricProperty"/>
    <owl:inverseOf rdf:resource="#machtesTo"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="MAS_PLUS">
    <rdfs:subClassOf rdf:resource="#DAISystem"/>
</owl:Class>
<owl:Class rdf:ID="Menge">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isSameAs"/>
            <owl:someValuesFrom rdf:resource="#Amount"/>
        </owl:Restriction>
    </owl:equivalentClass>
```

```
    <rdfs:subClassOf rdf:resource="#DataValuePartition"/>
</owl:Class>
<owl:Class rdf:ID="MengePlausibel">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isValid"/>
            <owl:someValuesFrom rdf:resource="#Menge"/>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#MAS_PLUS"/>
</owl:Class>
<owl:Class rdf:ID="rightDoorBack">
    <rdfs:subClassOf rdf:resource="#DoorValuePartition"/>
    <owl:disjointWith rdf:resource="#leftDoorBack"/>
    <owl:disjointWith rdf:resource="#leftDoorFront"/>
    <owl:disjointWith rdf:resource="#rightDoorFront"/>
</owl:Class>
<owl:Class rdf:ID="rightDoorFront">
    <rdfs:subClassOf rdf:resource="#DoorValuePartition"/>
    <owl:disjointWith rdf:resource="#leftDoorBack"/>
    <owl:disjointWith rdf:resource="#leftDoorFront"/>
    <owl:disjointWith rdf:resource="#rightDoorBack"/>
</owl:Class>
<owl:Class rdf:ID="rightSeat">
    <rdfs:subClassOf rdf:resource="#SeatValuePartition"/>
    <owl:disjointWith rdf:resource="#backSeat"/>
    <owl:disjointWith rdf:resource="#leftSeat"/>
</owl:Class>
<owl:Class rdf:ID="Roadster">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasDoorway"/>
                    <owl:cardinality rdf:datatype="&xsd;int">2</owl:cardinality>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasSeat"/>
                    <owl:cardinality rdf:datatype="&xsd;int">2</owl:cardinality>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasTop"/>
                    <owl:hasValue rdf:resource="#topSoftTop"/>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Car"/>
    <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
</owl:Class>
<owl:Class rdf:ID="SeatValuePartition">
    <rdfs:subClassOf rdf:resource="#ValuePartition"/>
</owl:Class>
<owl:Class rdf:ID="SLK">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDoorway"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
```

```
        <owl:valuesFrom rdf:resource="#leftDoorFront"/>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#DAICar"/>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasDoorway"/>
        <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
        <owl:valuesFrom rdf:resource="#rightDoorFront"/>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasTop"/>
        <owl:hasValue rdf:resource="#topSoftTop"/>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasDoorway"/>
        <owl:allValuesFrom>
            <owl:Class>
                <owl:unionOf rdf:parseType="Collection">
                    <owl:Class rdf:about="#leftDoorFront"/>
                    <owl:Class rdf:about="#rightDoorFront"/>
                </owl:unionOf>
            </owl:Class>
        </owl:allValuesFrom>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasSeat"/>
        <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
        <owl:valuesFrom rdf:resource="#rightSeat"/>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasSeat"/>
        <owl:allValuesFrom>
            <owl:Class>
                <owl:unionOf rdf:parseType="Collection">
                    <owl:Class rdf:about="#leftSeat"/>
                    <owl:Class rdf:about="#rightSeat"/>
                </owl:unionOf>
            </owl:Class>
        </owl:allValuesFrom>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasSeat"/>
        <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
        <owl:valuesFrom rdf:resource="#leftSeat"/>
    </owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#SLR_SoftTop"/>
```

```
    <owl:disjointWith rdf:resource="#SLR_HardTop"/>
    <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
    <rdfs:comment rdf:datatype="&xsd;string">hasSeat == 2</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="SLK200">
    <rdfs:subClassOf rdf:resource="#SLK"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasEngine"/>
            <owl:allValuesFrom rdf:resource="#Compressor200"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#SLK230"/>
    <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
</owl:Class>
<owl:Class rdf:ID="SLK230">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasEngine"/>
            <owl:allValuesFrom rdf:resource="#Compressor230"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#SLK"/>
    <owl:disjointWith rdf:resource="#SLK200"/>
    <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
</owl:Class>
<owl:Class rdf:ID="SLR_HardTop">
    <rdfs:subClassOf rdf:resource="#DAICar"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDoorway"/>
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#leftDoorFront"/>
                        <owl:Class rdf:about="#rightDoorFront"/>
                    </owl:unionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDoorway"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
            <owl:valuesFrom rdf:resource="#rightDoorFront"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDoorway"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
            <owl:valuesFrom rdf:resource="#leftDoorFront"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasTop"/>
```

```
            <owl:hasValue rdf:resource="#topHardTop"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#SLR_SoftTop"/>
    <owl:disjointWith rdf:resource="#SLK"/>
    <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
</owl:Class>
<owl:Class rdf:ID="SLR_SoftTop">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasSeat"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
            <owl:valuesFrom rdf:resource="#rightSeat"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDoorway"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
            <owl:valuesFrom rdf:resource="#leftDoorFront"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasSeat"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
            <owl:valuesFrom rdf:resource="#leftSeat"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasTop"/>
            <owl:hasValue rdf:resource="#topSoftTop"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#DAICar"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDoorway"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
            <owl:valuesFrom rdf:resource="#rightDoorFront"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasSeat"/>
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#leftSeat"/>
                        <owl:Class rdf:about="#rightSeat"/>
                    </owl:unionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
```

```
                <owl:onProperty rdf:resource="#hasDoorway"/>
                <owl:allValuesFrom>
                    <owl:Class>
                        <owl:unionOf rdf:parseType="Collection">
                            <owl:Class rdf:about="#leftDoorFront"/>
                            <owl:Class rdf:about="#rightDoorFront"/>
                        </owl:unionOf>
                    </owl:Class>
                </owl:allValuesFrom>
            </owl:Restriction>
        </rdfs:subClassOf>
        <owl:disjointWith rdf:resource="#SLR_HardTop"/>
        <owl:disjointWith rdf:resource="#SLK"/>
        <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
    </owl:Class>
    <TopValuePartition rdf:ID="topHardTop"/>
    <TopValuePartition rdf:ID="topSoftTop"/>
    <owl:Class rdf:ID="TopValuePartition">
        <rdfs:subClassOf rdf:resource="#ValuePartition"/>
    </owl:Class>
    <owl:Class rdf:ID="Truck">
        <rdfs:subClassOf rdf:resource="#Vehicle"/>
        <owl:disjointWith rdf:resource="#Car"/>
        <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
    </owl:Class>
    <owl:Class rdf:ID="ValuePartition">
        <rdfs:subClassOf rdf:resource="#CommonThing"/>
    </owl:Class>
    <owl:Class rdf:ID="Vehicle">
        <rdfs:subClassOf rdf:resource="#AutomotiveThing"/>
        <owl:disjointWith rdf:resource="#Aggregate"/>
        <owl:disjointWith rdf:resource="#Gear"/>
        <owl:disjointWith rdf:resource="#Engine"/>
        <protege:subclassesDisjoint rdf:datatype="&xsd;boolean">true</protege:subclassesDisjoint>
    </owl:Class>
</rdf:RDF>
```

Anhang O

Exemplarische Implementierung

O.1 WebSphere Service Registry und Repository

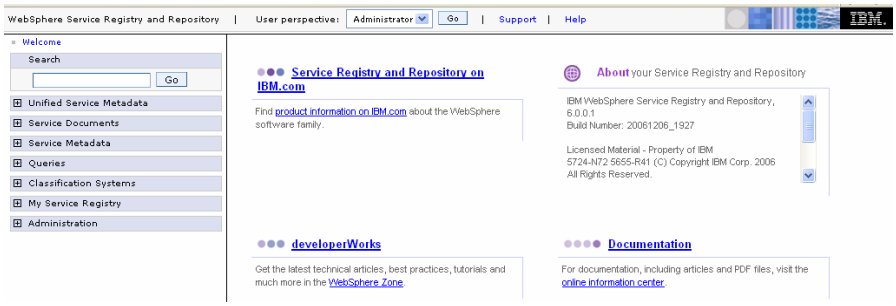

Abbildung O.1: WebSphere Service Registry und Repository (WSRR)

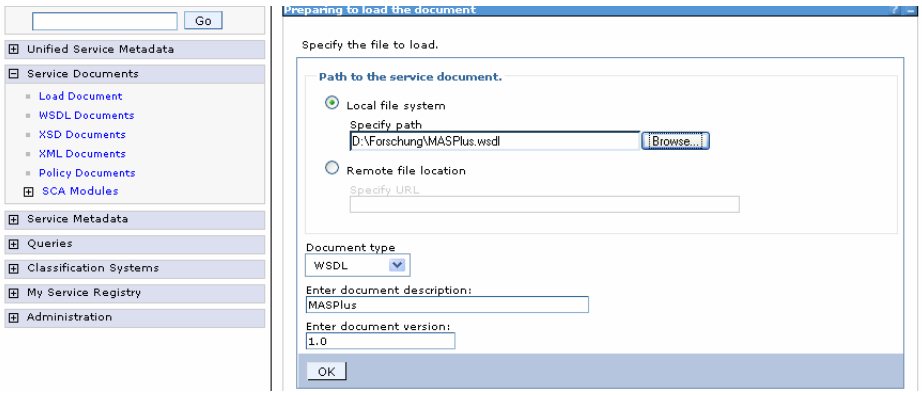

Abbildung O.2: WSDL-Datei im WSRR registrieren

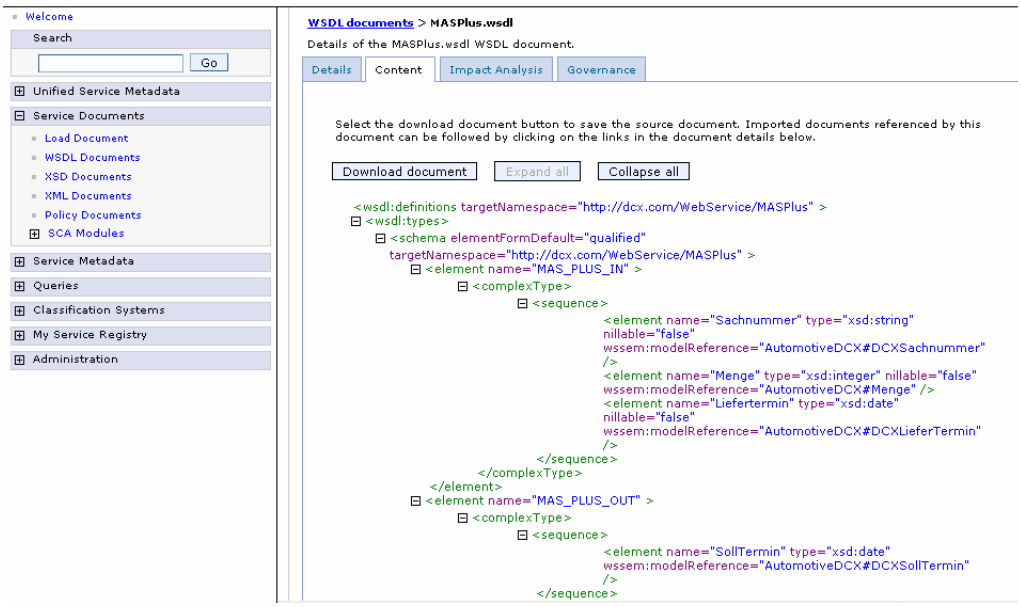

Abbildung O.3: Ablagestruktur WSRR

O.2 WebSphere Integration Developer

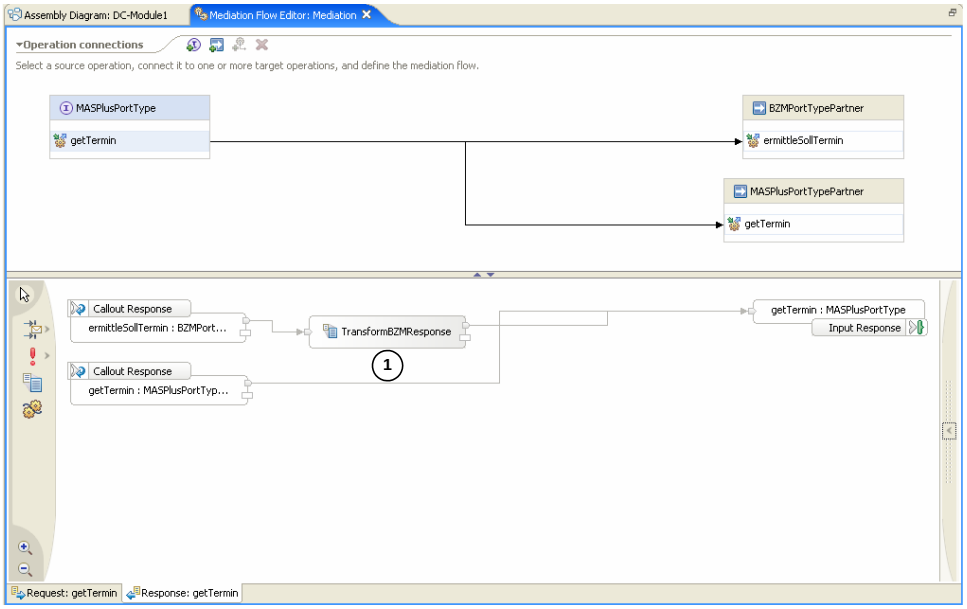

Abbildung O.4: Prozessfluss (Mediation) Anfrage

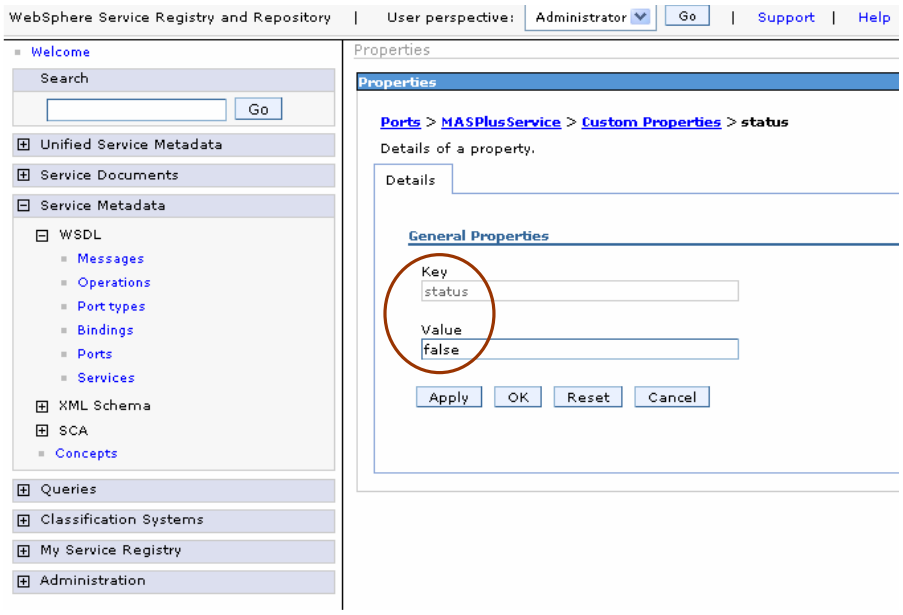

Abbildung O.6: Benutzerdefinierte Eigenschaft *status=false*

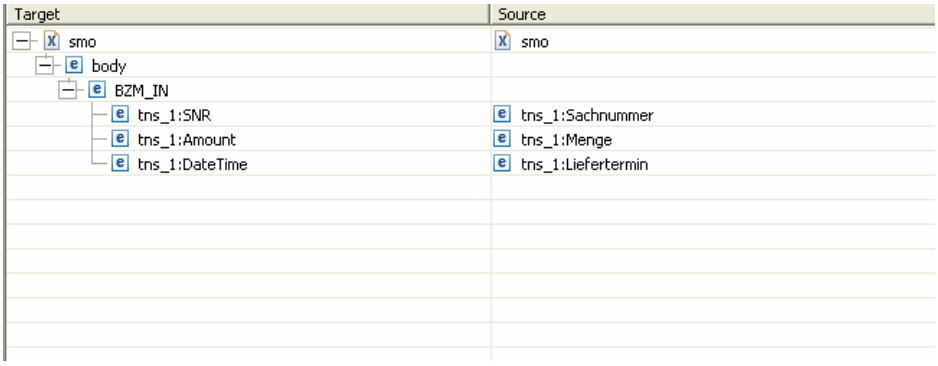

Abbildung O.5: Abbildung auf Service Message Objekts

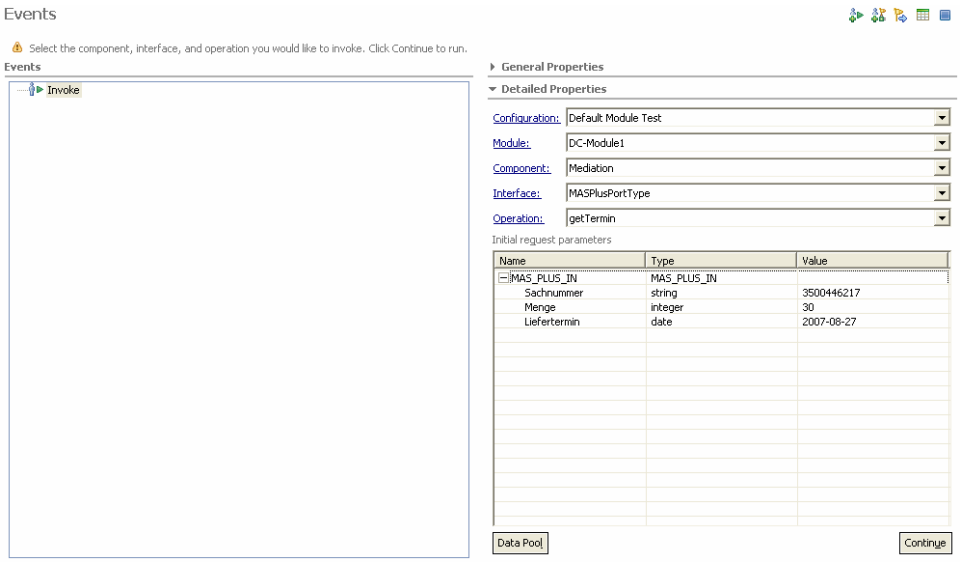

Abbildung O.7: Ablaufumgebung Mediation *Start*

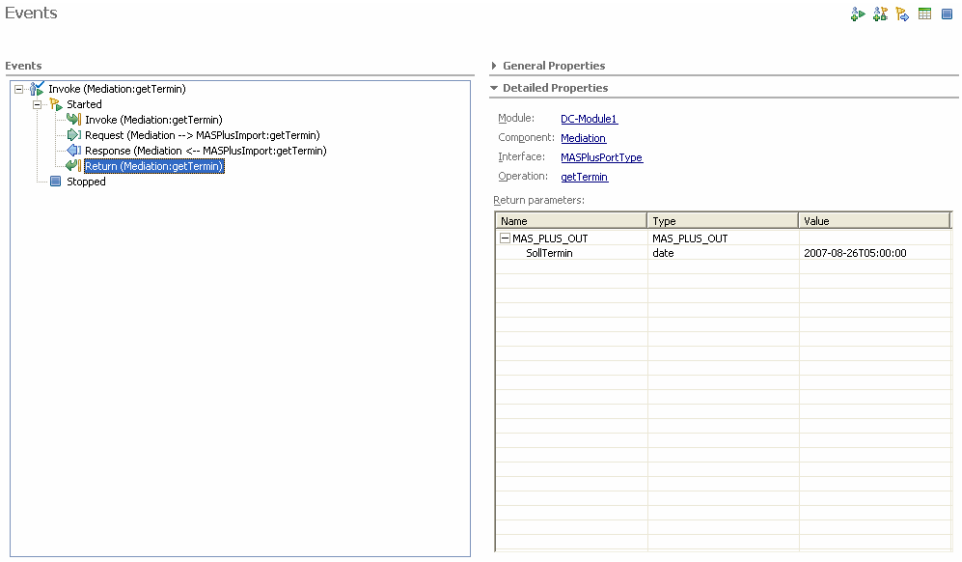

Abbildung O.8: Ablaufumgebung Mediation *status=true*

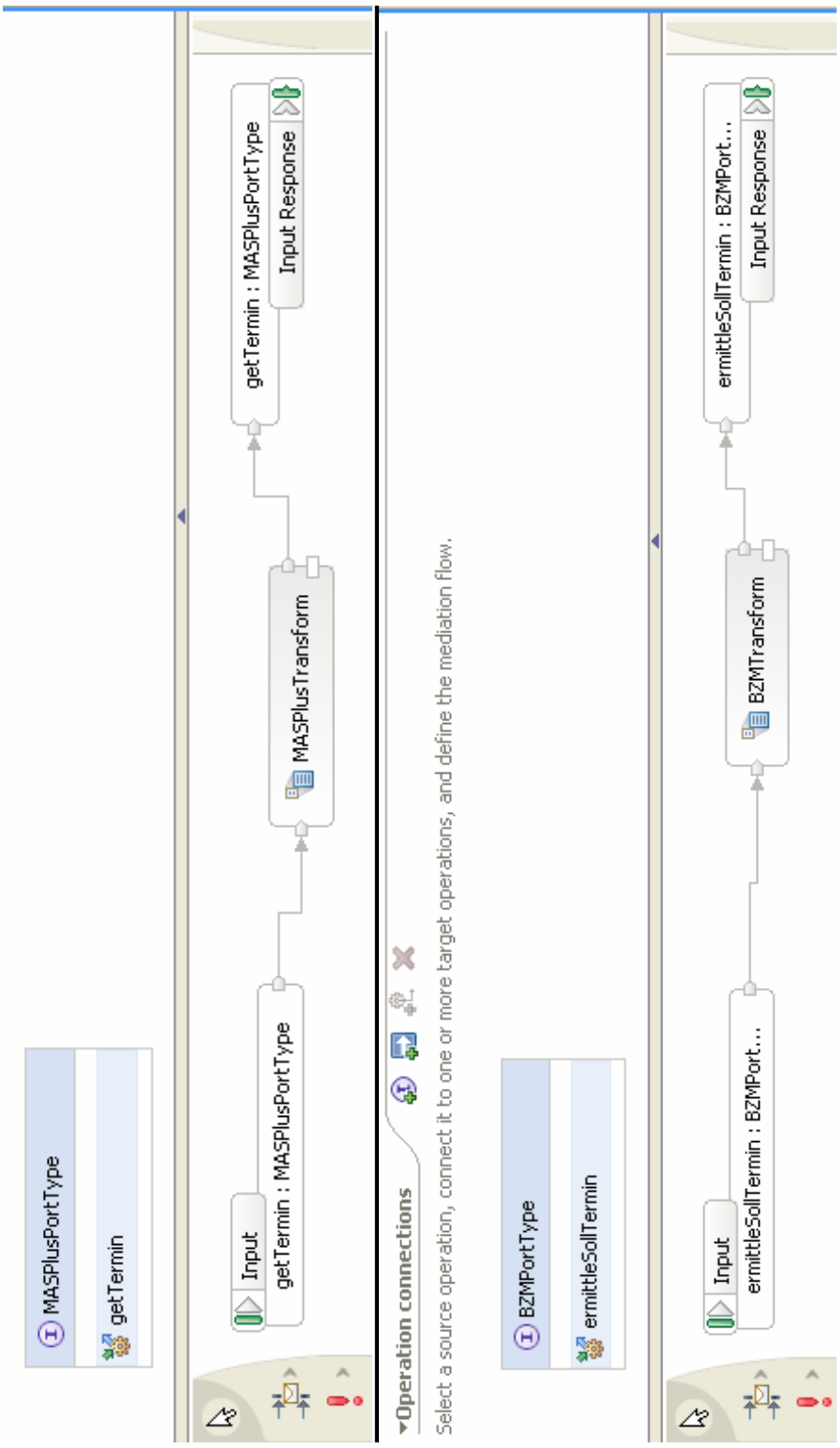

Abbildung O.9: Simulation der Service-Provider `MASPlus` und `BZM`

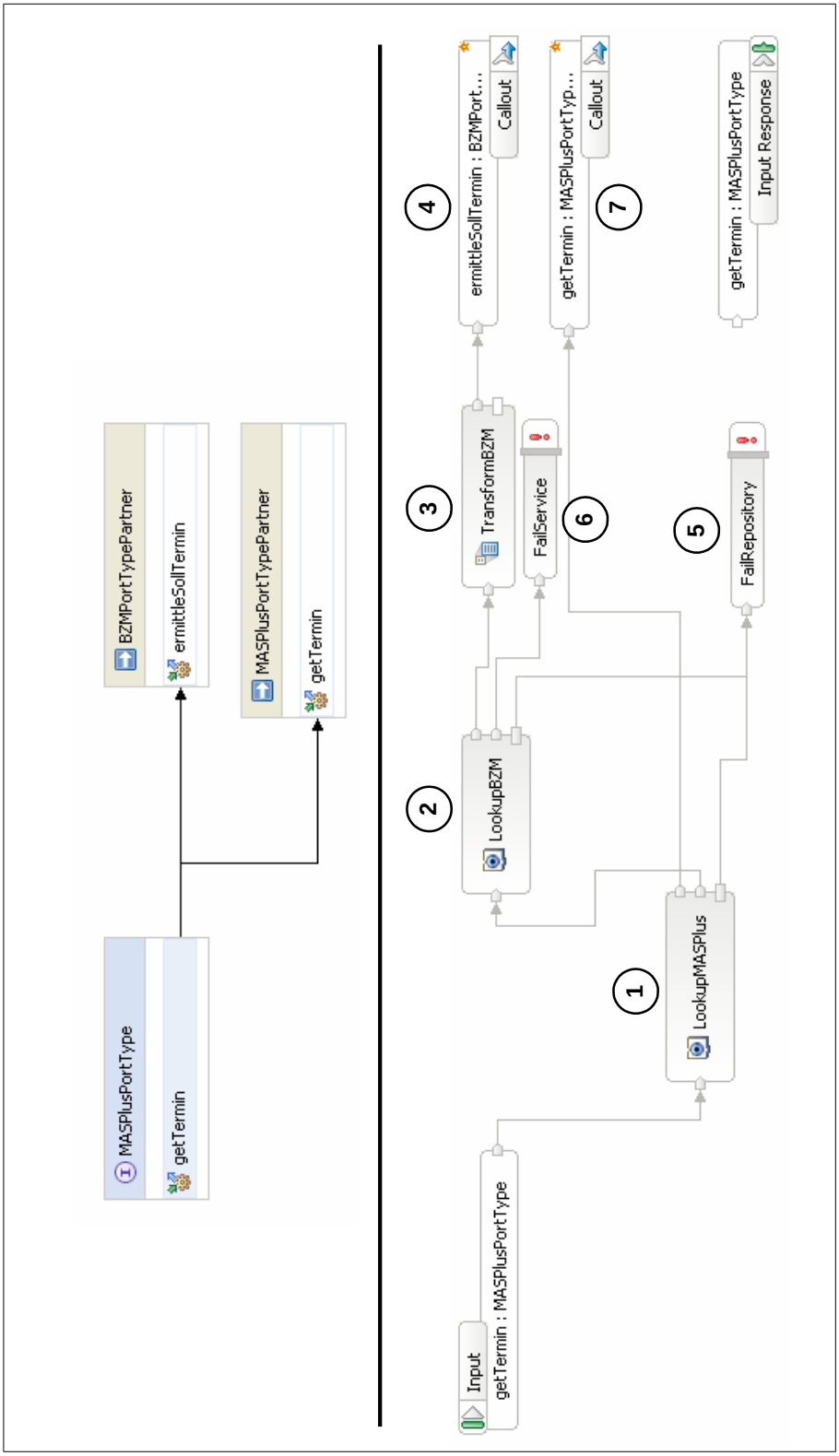

Abbildung O.10: Prozessfluss (Mediation) Anfrage

Anhang P

Service-Provider (WSDL-S)

P.1 MASPlus.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://dcx.com/WebService/MASPlus"
   xmlns:intf="http://dcx.com/WebService/MASPlus"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
   xmlns:AutomotiveDCX="http://www.dcx.com/ontologies/dcx.owl#"
   targetNamespace="http://dcx.com/WebService/MASPlus">

   <wsdl:types>
 <schema elementFormDefault="qualified"
         targetNamespace="http://dcx.com/WebService/MASPlus"
         xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="MAS_PLUS_IN">
         <complexType>
            <sequence>
<element name="Sachnummer"
         type="xsd:string" nillable="false"
         wssem:modelReference="AutomotiveDCX#DCXSachnummer"/>
<element name="Menge"
         type="xsd:integer"
         nillable="false"
         wssem:modelReference="AutomotiveDCX#Menge"/>
<element name="Liefertermin"
         type="xsd:date"
         nillable="false"
         wssem:modelReference="AutomotiveDCX#DCXLieferTermin"/>
</sequence>
         </complexType>
      </element>
      <element name="MAS_PLUS_OUT">
         <complexType>
            <sequence>
            <element name="SollTermin"
                     type="xsd:date"
                     wssem:modelReference="AutomotiveDCX#DCXSollTermin"/>
            </sequence>
```

```
      </complexType>
   </element>
</schema>
</wsdl:types>

<wsdl:message name="MAS_PLUS_OUT">
   <wsdl:part name="parameters" element="intf:MAS_PLUS_OUT"/>
</wsdl:message>

<wsdl:message name="MAS_PLUS_IN">
   <wsdl:part name="parameters" element="intf:MAS_PLUS_IN"/>
</wsdl:message>

<wsdl:portType name="MASPlusPortType">
   <wsdl:operation name="getTermin">
      <wsdl:input name="MAS_PLUS_IN" message="intf:MAS_PLUS_IN"/>
      <wsdl:output name="MAS_PLUS_OUT" message="intf:MAS_PLUS_OUT"/>
      <wssem:precondition name="MengePlausibel"
       wssem:modelReference="AutomotiveDCX#MengePlausibel"/>
      <wssem:effect name="SollTerminErmittelt"
       wssem:modelReference="AutomotiveDCX#DCXSollTerminVerfuegbar"/>
   </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="MASPlusBinding" type="impl:MASPlusPortType">
   <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="getTermin">
      <wsdlsoap:operation/>
         <wsdl:input>
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
      <wsdl:output>
   <wsdlsoap:body use="literal"/>
   </wsdl:output>
   </wsdl:operation>
      </wsdl:binding>
      <wsdl:service name="MASPlus">
      <wsdl:port name="MASPlusService" binding="impl:MASPlusBinding">
      <wsdlsoap:address location="http://localhost:29080/DCX/services/MASPlus"/>
   </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

P.2 BZM.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://dcx.com/WebService/BZM"
   xmlns:intf="http://dcx.com/WebService/BZM"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
   xmlns:AutomotiveDCX="http://www.dcx.com/ontologies/dcx.owl#"
   targetNamespace="http://dcx.com/WebService/BZM">

   <wsdl:types>
 <schema elementFormDefault="qualified"
         targetNamespace="http://dcx.com/WebService/BZM"
         xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="BZM_IN">
         <complexType>
            <sequence>
<element name="Sachnummer"
         type="xsd:string" nillable="false"
         wssem:modelReference="AutomotiveDCX#DCXSachnummer"/>
<element name="Menge"
         type="xsd:integer"
         nillable="false"
         wssem:modelReference="AutomotiveDCX#Menge"/>
<element name="Liefertermin"
         type="xsd:date"
         nillable="false"
         wssem:modelReference="AutomotiveDCX#DCXLieferTermin"/>
</sequence>
         </complexType>
      </element>
      <element name="BZM_OUT">
         <complexType>
            <sequence>
            <element name="SollTermin"
                     type="xsd:date"
                     wssem:modelReference="AutomotiveDCX#DCXSollTermin"/>
            </sequence>
         </complexType>
      </element>
   </schema>
   </wsdl:types>

   <wsdl:message name="BZM_OUT">
      <wsdl:part name="parameters" element="intf:BZM_OUT"/>
   </wsdl:message>

   <wsdl:message name="BZM_IN">
      <wsdl:part name="parameters" element="intf:BZM_IN"/>
   </wsdl:message>

   <wsdl:portType name="BZMPortType">
      <wsdl:operation name="getTermin">
         <wsdl:input name="BZM_IN" message="intf:BZM_IN"/>
```

```
        <wsdl:output name="BZM_OUT" message="intf:BZM_OUT"/>
        <wssem:precondition name="MengePlausibel"
         wssem:modelReference="AutomotiveDCX#MengePlausibel"/>
        <wssem:effect name="SollTerminErmittelt"
         wssem:modelReference="AutomotiveDCX#DCXSollTerminVerfuegbar"/>
     </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="BZMBinding" type="impl:BZMPortType">
     <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
     <wsdl:operation name="ermittleSollTermin">
        <wsdlsoap:operation/>
           <wsdl:input>
              <wsdlsoap:body use="literal"/>
           </wsdl:input>
        <wsdl:output>
     <wsdlsoap:body use="literal"/>
     </wsdl:output>
     </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="BZM">
        <wsdl:port name="BZMService" binding="impl:BZMBinding">
        <wsdlsoap:address location="http://localhost:29080/DCX/services/BZM"/>
     </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Literatur

[1] IBM United States Software Announcement 206-283. IBM WebSphere Business Services Fabric delivers a comprehensive, end-to-end platform for composite business services. Website, 2006. Available online at http://www.ibm.com/common/ssi/rep_ca/3/897/ENUS206-283/ENUS206-283.PDF; visited on September 22th 2007.

[2] R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Dynamic Web service Composition in METEOR-S. *Technical report, LSDIS Lab*, Computer Science Department, University of Georgia, USA, 2004.

[3] A. M. Ahtisham. *Towards Integration of Business Processes and Semantic Web Services*. PhD thesis, Leipzig University, Faculty of Mathematics and Computing Science, 2007.

[4] R. Akkiraju, J. Farell, J.A. Miller, M. Nagarajan, A. Sheth, and K. Verma. Web Service Semantics – WSDL-S. Website, 2005. Available online at http://www.w3.org/Submission/WSDL-S/; visited on August 22th 2007.

[5] R. Akkiraju, B. Srivastava, A.-A. Ivan, R. Goodwin, and T. Syeda-Mahmood. SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition. *IEEE International Conference on Web Services (ICWS'06)*, pages 37–44, 2006.

[6] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Website, 2003. Available online at ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf; visited on August 22th 2007.

[7] J. Angele, E. Mönch, A. Nierlich, H. Rudat, and H-P. Schnurr. *Anwendungen und Good Practices Semantischer Technologien, Semantic Web, X.media.press*, pages 337–356. Springer Berlin Heidelberg, 2006.

[8] C. Au. Dienstbeschreibungen und -verzeichnisse in serivce-orientierten architekturen. Master's thesis, Karlsruhe (TH) University, Faculty of Telematics, 2006.

[9] S. Auer. *Towards Agile Knowledge Engineering: Methodology, Concepts and Applications*. PhD thesis, Leipzig University, Faculty of Mathematics and Computing Science, 2006.

[10] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook. Theory, Implementation and Applications*. Cambridge University Press, New York, 2003.

[11] D. Beimborn, S. Mintert, and T. Weitzel. Web Services und ebXML. *Wirtschaftsinformatik*, 44(2):277–280, 2002.

[12] T. Berners-Lee and W3C Consortium. Extensible Markup Language (XML). Website, 1997. Available online at http://www.w3.org/XML/; visited on August 22th 2007.

[13] T. Berners-Lee, R. Fielding, and L. Masinter. URI Interest Group Charter. Website, 2005. Available online at http://www.ietf.org/rfc/rfc3986.txt; visited on August 22th 2007.

[14] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. *Scientific American*, 284(5):34–43, 2001.

[15] T. Berners-Lee, L. Masinter, and M. McCahill. Network Working Group. Website, 1994. Available online at http://www.ietf.org/rfc/rfc1738.txt; visited on August 22th 2007.

[16] N. Bieberstein, S. Bose, M. Fiammante, K. Jones, and R. Shah. *Service-Oriented Architecture Compass – Business Value, Planning and Enterprise Roadmap*. Pearson Education, Inc., One Lake Street. Upper Saddle River, NJ 07458, 2006.

[17] BITKOM. Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. Website, 2007. Available online at http://bitkom.org/; visited on November 3th 2007.

[18] M. Blow, Y. Goland, M. Kloppmann, F. Leymann, G. Phau, D. Roller, and M. Rowley. BPELj: BPEL for Java. Website, 2004. Available online at http://www.ibm.com/developerworks/library/specification/ws-bpelj/; visited on August 22th 2007.

[19] P. Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, University Twente, 1997.

[20] R. Chinnici, J. J. Moreau, A. Ryman, and S. Weerawarana. Web services Description Language (WSDL) version 2.0. Website, 2007. Available online at http://www.w3.org/TR/wsdl20/; visited on August 22th 2007.

[21] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services Description Language (WSDL) 1.1. Website, 2001. Available online at http://www.w3.org/TR/wsdl; visited on August 22th 2007.

[22] J. Clark and W3C Consortium. XSL Transformations (xslt). Website, 1999. Available online at http://www.w3.org/TR/xslt; visited on August 22th 2007.

[23] OASIS Committee. OASIS UDDI 3.0 – Advancing Web Services Discovery Standard. Website, 2002. Available online at http://www.uddi.org/; visited on August 22th 2007.

[24] Technical Committee. OASIS – Advancement of Structured Information Standards. Website, 1993. Available online at http://www.oasis-open.org/; visited on August 22th 2007.

[25] Software AG The SOA Company. Credit Suisse: Wichtigster Neukunde der Software AG im 2. Quartal 2007. Website, 2007. Available online at http://www.softwareag.com/de/Images/Q2%202007_KHS_dt__tcm17-31387.pdf; visited on August 22th 2007.

[26] D. Connolly and W3C Consortium. Standard Generalized Markup Language (SGML). Website, 1995. Available online at http://www.w3.org/MarkUp/SGML/; visited on August 22th 2007.

[27] D. Connolly and N. Walsh. URI Interest Group Charter. Website, 2005. Available online at http://www.w3.org/2004/07/uri-ig-charter.html; visited on August 22th 2007.

[28] W3C Consortium. DAML+OIL: DAML Ontology Interchange Language. Website, 2001. Available online at http://www.daml.org/2001/03/daml+oil-index.html; visited on October 16th 2007.

[29] Word Wide Web Consortium. Web Service Activity. Website, 2002. Available online at http://www.w3.org/2002/ws/; visited on August 22th 2007.

[30] R. Credle, J. Adams, K. Clark, P.G. Yun, H. Jeter, J. Lopes, S. Nasser, and K. Peri. Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB. sg24-7369-00. Website, 2007. Available online at http://www.redbooks.ibm.com/redbooks/pdfs/sg247369.pdf; visited on August 22th 2007.

[31] O. Dalferth. Exemplary implementation of a purchase requisition process according to the principles of a service-oriented architecture. Master's thesis, Tübingen University, Faculty of Computing Science, 2007.

[32] T. H. Davenport. Reengineering Work through Information Technology. *Process Innovation*, page 5, 1993.

[33] J. de Bruijn and ESSI WSMO Working Group. WSML – Web Service Modeling Language. Website, 2002. Available online at http://www.wsmo.org/wsml/; visited on August 22th 2007.

[34] J. Dehnert. *A Methodology for Workflow Modeling – From business process modeling towards sound workflow specification.* PhD thesis, Technische Universität Berlin, Fakultät IV Elektrotechnik und Informatik, 2003.

[35] SAP Deutschland. SAP NetWeaver – Die Plattform für innovative Prozesse. Website, 2005. Available online at http://www.sap.com/germany/plattform/index.epx; visited on November 12th 2007.

[36] SAP Deutschland. SAP Business ByDesign. Website, 2006. Available online at http://www.sap.com/germany/solutions/sme/businessbydesign/index.epx; visited on November 12th 2007.

[37] W. Dostal, M. Jeckle, I. Melzer, and B. Zengler. *Service-orientierte Architekturen mit Web Services. Konzepte – Standards – Praxis.* Elsevier GmbH, München, 2005.

[38] D. F. D'Souza and A. C. Wills. *Objects, Components and Frameworks with UML: The Catalysis Approach.* Addison-Wesley, 1999.

[39] M. Duerst and M. Suignard. Network Working Group. Website, 2005. Available online at http://www.ietf.org/rfc/rfc3987.txt; visited on August 22th 2007.

[40] Wolfgang Emmerich. *Konstruktion von verteilten Objekten.* dpunkt.verlag, Heidelberg, 2003.

[41] Active Endpoints, Adobe, BEA, IBM, Oracle, and SAP. Web Services Human Task (WS-HumanTask), Version 1.0. Website, 2007. Available online at http://www.adobe.com/devnet/livecycle/pdfs/ws_humantask_spec.pdf; visited on August 22th 2007.

[42] T. Erl. *Service-Oriented Architecture – Concepts, Technology, and Design.* Prentice Hall, Crawfordsville, Indiana, 2006.

[43] ATHENA (EU-Forschungsprojekt). Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Applications (ATHENA). Website, 2004. Available online at http://www.athena-ip.org/; visited on September 28th 2007.

[44] R4eGOV (EU-Forschungsprojekt). Making eGovernment work. Website, 2007. Available online at http://www.r4egov.eu/; visited on November 12th 2007.

[45] THESEUS (EU-Forschungsprojekt). Forschungsprogramm für eine neue internetbasierte Wissensinfrastruktur. Website, 2007. Available online at http://theseus-programm.de/was-ist-theseus; visited on November 12th 2007.

[46] J. Farrel and H. Lausen. Semantic Annotations for Web services Description Language Working Group. Website, 2007. Available online at http://www.w3.org/2002/ws/sawsdl/; visited on August 22th 2007.

[47] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[48] D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Dominque. *Enabling Semantic Web Services – The Web Service Modeling Ontology.* Springer Verlag, Heidelberg, 2002.

[49] H. Fischer, A. Fleischmann, and S. Obermeier. *Geschäftsprozesse realisieren – Ein praxisorientierter Leitfaden von der Strategie bis zur Implementierung.* vieweg Verlag, Wiesbaden, 2006.

[50] United Nations Economic Commission for Europe. United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT). Website, 2006. Available online at http://www.unece.org/cefact/; visited on September 28th 2007.

[51] Semantic Annotations for Web Services Description Language Working Group. Semantic Annotations for WSDL WG teleconference. Website, 2006. Available online at http://www.w3.org/2002/ws/sawsdl/minutes/20060606#action04; visited on November 8th 2007.

[52] Apache Software Foundation. Web Services – AXIS. Website, 2003. Available online at http://ws.apache.org/axis/; visited on August 22th 2007.

[53] Fraunhofer Institut für Arbeit und Organisation (IAO). Stuttgarter Softwaretechnik Forum 2006. Website, 2007. Available online at http://projekte.swm.iao.fhg.de/ssf2006/tag2.html; visited on November 3th 2007.

[54] Institut für Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB). WSDL-S Semantic Web Services. Website, 2006. Available online at http://www.aifb.uni-karlsruhe.de/Lehre/Sommer2006/ws/PresentationWSDL-S; visited on November 8th 2007.

[55] E. Gamma. *Objektorienterte Software-Entwicklung am Beispiel von ET++: Design-Muster, Klassenbibliothek, Werkzeuge.* PhD thesis, Zürich University, Faculty of Computing Science, 1991.

[56] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Entwurftsmuster – Elemente wiederverwendbarer objektorierntierter Software.* Addison-Wesley, Bonn, 1995.

[57] Credit Suisse Group. Credit Suisse – Home. Website, 2007. Available online at http://www.credit-suisse.com/ch/de/; visited on August 22th 2007.

[58] Open Service Oriented Architecture Group. Service Data Objects Specifications. Website, 2006. Available online at http://www.osoa.org/display/Main/Service+Data+Objects+Specifications; visited on August 22th 2007.

[59] T. R. Gruber. *Towards principles for the design of ontologies used for knowledge sharing.* Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, 1993.

[60] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. Soap version 1.2. Website, 2007. Available online at http://www.w3.org/TR/soap12-part1/; visited on August 22th 2007.

[61] C. Hackländer. *ebXML – Das umfassende Rahmenwerk für Electronic Business. Eine kritische Darstellung.* Tectum Verlag, Marburg, 2004.

[62] M. Hammer and J. Champy. Business Reengineering. page 52, 1995.

[63] M. Harvey. *Essential Business Process Modeling.* O'REILLY, Gravenstein, 2005.

[64] H. Helbig. *Die semantische Struktur natürlicher Sprache. Wissenspräsentation mit MultiNet.* Spinger Verlag Heidelberg, 2001.

[65] H. Herre. General Formal Ontology (GFO) – A Foundational Ontology for Conceptual Modelling to appear in 'Theory and Application of Ontology'. pages 1–50, Springer Verlag, 2008.

[66] M. Herrmann. Ein agiles vorgehensmodell für das software-engineering mit fokus auf extreme programming, der unifed modelling language und dem unified process. Master's thesis, Reutlingen University, Faculty Computing Science, 2004.

[67] M. Herrmann, M. A. Aslam, S. Auer, and R. Golden. *Real-life SOA Experiences and an Approach Towards Semantic SOA, Best Practices and Methodologies in SOA, OOPSLA 2006*, pages 72–81. L. A. Skar and A. A. Bjerkestrand, 2006.

[68] M. Herrmann, H-J. Groß, P. Rothländer, M. Flehmig, and I. Melzer. Daimler AG. SOA Special Interest Group, 2005.

[69] P. Herrmann, U. Kebschull, and W. Spruth. *Einführung in die z/OS und OS/390. Web-Services und Internet-Anwendungen für Mainframes.* Oldenburg Verlag, München, 2004.

[70] D. Hollingsworth. WfMC – The Workflow Reference Model. Website, 1995. Available online at http://www.wfmc.org/standards/docs/tc003v11.pdf; visited on October 28th 2007.

[71] Ian Horrocks. FaCT++ – Fast Classification of Terminologies. Website, 2006. Available online at http://owl.man.ac.uk/factplusplus/; visited on November 30th 2007.

[72] S. Mintert (Hrsg.). *XML & CO. Die W3C-Spezifikation für Dokumenten- und Datenarchitektur*. Addison-Wesley, 2002.

[73] M. Kloppmann (IBM), D. Koenig (IBM), F. Leymann (IBM), G. Pfau (IBM), A. Rickayzen (SAP), C. von Riegen (SAP), P. Schmidt (SAP), and I. Trickovic (SAP). WS-BPEL Extension for People – BPEL4People. A Joint White Paper by IBM and SAP. Website, 2005. Available online at http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/; visited on August 22th 2007.

[74] The Internet Engineering Task Force (IETF). RPC: Remote Procedure Call Protocol Specification Version 2. Website, 1995. Available online at http://www.ietf.org/rfc/rfc1831.txt; visited on October 22th 2007.

[75] The Internet Engineering Task Force (IETF). URI-List Document Format. Website, 2007. Available online at http://www.ietf.org/internet-drafts/draft-ietf-sip-uri-list-subscribe-01.txt; visited on November 8th 2007.

[76] Innsbruck Institut für Informatik. Digital Enterprise Research Institute (DERI). Website, 2001. Available online at http://informatik.uibk.ac.at/index.html.de; visited on August 22th 2007.

[77] Software Engineering Institute. Capability Maturity Modell Integration V1.2 – (CMMI). Website, 2006. Available online at http://www.sei.cmu.edu/cmmi/; visited on September 22th 2007.

[78] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-Oriented Software Engineering – Use Case Driven Approach*. Addison-Wesley, Wokingham, England, 1992.

[79] N. M. Josuttis. *SOA in Practice – The Art of Distributed System Design*. O'REILLY Köln, 2007.

[80] SUN JSR 208. Java Business Integration (JBI) 1.0, 2005.

[81] H. Kagermann and J. Oesterle. *Geschäftsmodelle 2010 – Wie CEOs Unternehmen transformieren*. Frankfurter Allgemeine Buch, Frankfurt am Main, 2006.

[82] Kalbarczyk, R. K. Iyer, S. Bagchi, and K. Whisnant. A software infrastructure for adaptive fault tolerance. *IEEE Transactions on Parallel and Distributed Systems*, 10:6–7, 1999.

[83] FZI Karlsruhe. KAON2 – OWL-DL, SWRL, and F-Logic. Website, 2004. Available online at http://kaon2.semanticweb.org/; visited on November 30th 2007.

[84] M. Keen, H-H. Chin, C. Ganapathi, D. Ghazaleh, P. Krogdahl, W. Neave, M. Sahni, and J. Thorwart. *Patterns: Extended Enterprise SOA and Web Services*. IBM redbook, SG24-7135-00, 2006.

[85] Racer Systems GmbH & Co. KG. RacerPro – der OWL Reasoner und Inference Server für das Semantic Web. Website, 2004. Available online at http://www.racer-systems.com/; visited on October 03th 2007.

[86] H. Klaeren. Skriptum Softwaretechnik, 2006.

[87] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and abstract syntax. W3C Recommendation. Website, 2004. Available online at http://www.w3.org/TR/rdf-concepts; visited on August 22th 2007.

[88] W. Kozaczynski. Architecture Frame-work for Business Components. *5th International Conference on Software Reuse ICSR98*, page 300, 1998.

[89] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA, Service-Oriented Architecture Best Practices*. Prentice Hall, New Jersey, 2005.

[90] V. Kramberg. Pattern-based Evaluation of IBM WebSphere BPEL. Master's thesis, Institut für Architektur von Anwendungssystemen (IAAS), 2006.

[91] Volker Kramberg. Pattern-based Evaluation of IBM WebSphere BPEL, 2006.

[92] F. Leymann. Web Services Flow Language (WSFL 1.0). Website, 2001. Available online at http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf; visited on August 22th 2007.

[93] D. J. Mandell and S. A. McIlraith. Adapting BPEL4WS for the Semantic Web: The bottom-up approach to Web service interoperation. *International Semantic Web Conference*, volume 2870 of Lecture Notes in Computer Science:227–241, 2003.

[94] E. Marcus and H. Stern. *Blueprints for High Availability, 2nd Edition*. WILEY, 2003.

[95] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. Website, 2002. Available online at http://www.daml.org/services/owl-s/1.1/overview/; visited on August 22th 2007.

[96] T. Maurer. Interactive Objects – Process-to-Application (P2A). Website, 2005. Available online at http://www.arcstyler.com/; visited on November 30th 2007.

[97] D. McCoy and Y. Natis. Service-Oriented Architecture: Mainstream Straight Ahead. *Gartner Research*, pages LE–19–7652, 2003.

[98] D. L. McGuinness and F. V. Harmelen. OWL Web Ontology Language – overview. Website, 2004. Available online at http://www.w3.org/TR/owl-features; visited on August 22th 2007.

[99] H. Meyer, H. Overdick, and M. Weske. Plængine: A system for automated Service Composition and Process en-actment. *In Proceedings of WWW Service Composition with Se-mantic Web Services*, University of Technology of Compiegne:3–12.

[100] Microsoft. COM: Component Object Model Technologies. Website, 1993. Available online at http://www.microsoft.com/com/default.mspx; visited on October 22th 2007.

[101] Microsoft. DCOM Technical Overview. Website, 1996. Available online at http://msdn2.microsoft.com/en-us/library/ms809340.aspx; visited on October 22th 2007.

[102] mindswap. pellet – OWL DL reasoner in Java. Website, 2003. Available online at http://pellet.owldl.com/; visited on November 30th 2007.

[103] mindswap. SWOOP – A Hypermedia-based Featherweight OWL Ontology Editor. Website, 2004. Available online at http://code.google.com/p/swoop/; visited on November 30th 2007.

[104] M. Minsky. A Framework for Representing Knowledge. MIT-AI Laboratory Memo 306. Website, 1974. Available online at http://web.media.mit.edu/~minsky/papers/Frames/frames.html; visited on August 22th 2007.

[105] E. Mönch, M. Ullrich, H-P. Schnurr, and J. Angele. SemanticMiner – Ontology-Based Knowledge Retrieval. *Journal of Universal Computer Science*, 9(7):682–696, 2003.

[106] H. Moertl and H-J. Groß. Daimler AG. Prozessorientiertes Managementsystem, Optimieren und Managen (ProOM), 2006.

[107] Marc Nübling. Entwurf einer security-infrastruktur für soa auf basis von web services und ibm websphere developer. Master's thesis, Tübingen University, Faculty Computing Science, 2007.

[108] SAP Community Network. Standards and Enterprise SOA. Website, 2006. Available online at https://www.sdn.sap.com/irj/sdn/standards; visited on September 28th 2007.

[109] SAP Community Network. UN/CEFACT Core Components Technical Specification (UN/CEFACT CCTS). Website, 2006. Available online at https://www.sdn.sap.com/irj/sdn?rid=/webcontent/uuid/1baa57f9-0a01-0010-1684-c42a08982294; visited on September 28th 2007.

[110] N. F. Noy and D. L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Website, 2006. Available online at http://protege.stanford.edu/publications/ontology_development/ontology101.pdf; visited on August 22th 2007.

[111] OASIS. ebXML – Electronic Business XML. Website, 2000. Available online at http://www.ebxml.org/; visited on August 22th 2007.

[112] B. Oestereich. *Analyse und Design mit UML 2.1. Objektorientierte Softwareentwicklung.* Oldenburg Verlag, München, 2006.

[113] H. Oesterle and C. Legner. SOA for Automotive. Competence Center Business Networking 3. Website, 2006. Available online at http://soa.iwi.unisg.ch/; visited on August 22th 2007.

[114] Stanford Center of Biomedical Informatics Research. Protégé – Open Source Ontology Editor. Website, 2000. Available online at http://protege.stanford.edu/; visited on November 30th 2007.

[115] Department of the Navy XML. Registry Requirenments. Website, 2003. Available online at http://xml.gov/presentations/lmi3/DONXMLRegistryRequirements.pdf; visited on August 22th 2007.

[116] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour. Semantic WS-agreement partner selection. In Proceedings of the 15th International Conference on World Wide Web. pages 697–706, 2006.

[117] Object Management Group (OMG). CORBA – Common Object Request Broker Architecture. Website, 2001. Available online at http://www.corba.org/; visited on October 13th 2007.

[118] Object Management Group (OMG). Business Process Modeling Initiative (BPMI). Website, 2005. Available online at http://www.bpmi.org/; visited on October 28th 2007.

[119] Computerworld online. Credit Suisse setzt auf Applikation der Software AG. Website, 2007. Available online at http://www.computerworld.ch/aktuell/news/40983/index.html; visited on August 22th 2007.

[120] OpenQuasar. Quasar – Quality Software Architecture. Website, 2005. Available online at http://www.openquasar.de/de/quasar/index.html; visited on October 30th 2007.

[121] S. R. Ponnekanti and A. Fox. SWORD: A developer toolkit for web service composition. *International Semantic Web Conference*, 2002.

[122] L. Prise and H. Wimmel. *Theoretische Informatik. Petri-Netze.* Springer Verlag, Heidelberg, 2003.

[123] M. R. Quillian. Word concepts. A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12:410–430, 1967.

[124] L. Quin and XSL Working Group. eXtensible Stylesheet Language family (xsl). Website, 1998. Available online at http://www.w3.org/TR/xslt; visited on August 22th 2007.

[125] R. Reussner and W. Hasselbring. *Handbuch der Software-Architektur.* dpunkt.verlag, 2006.

[126] D. Roman and ESSI WSMO Working Group. WSMO – Web Services Modeling Ontology. Website, 2002. Available online at http://www.wsmo.org; visited on August 22th 2007.

[127] RosettaNet. RosettaNet Community. Website, 1998. Available online at http://portal.rosettanet.org/cms/sites/RosettaNet/; visited on November 13th 2007.

[128] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, NJ, 1991.

[129] S. Russel and P. Norvig. *Künstliche Intelligenz – Ein moderner Ansatz.* Prentice Hall, 2004.

[130] M. Wagner S. Balzer, T. Liebig. Pitfalls of OWL-S - A practical Semantic Web Use Case. *2nd International Conference on Service Oriented Computing*, ICSOC, 2004.

[131] D. Sadoski and S. Comella-Dorda. Three Tier Software Architectures. Software Technology Roadmap. http://www.sei.cmu.edu/str/descriptions/threetier_body.html, 2000.

[132] A. Saimi, T. Syomura, H. Suganuma, and I. Ishida. Presentation Layer Framework of Web Application Systems with Server-Side Java Technology. *International Computer Software and Applications Conference*, page 473, 2000.

[133] A.-W. Scheer. *ARIS-Modellierungs-Methoden, Metamodelle, Anwendungen.* Springer Verlag Heidelberg, 1991.

[134] A. W. Scheer. ARIS – Vom Geschäftsprozess zum Anwendungssystem. page 3, 2002.

[135] J. H. Schmeler and W. Sesselmann. Geschäftsprozessmanagement in der Praxis. pages 39–40, 2003.

[136] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[137] U. Schneider and D. Werner. *Taschenbuch der Informatik*. Fachbuchverlag Leipzig im Carl Hanser Verlag München Wien, 2001.

[138] R. W. Schulte and Y. V. Natis. „Service Oriented“ Architectures, Part 1 and Part 2. Website, 1996. Available online at http://www.gartner.com/DisplayDocument?id=302868 and http://www.gartner.com/DisplayDocument?id=302869; visited on October 04th 2007.

[139] R. W. Schulte and Y. V. Natis. Introduction to Service-Oriented Architecture. Website, 2003. Available online at http://www.gartner.com/DisplayDocument?doc_cd=114295; visited on October 05th 2007.

[140] Sun Sun Developer Network (SDN). Java SE at a Glance. Website, 1994. Available online at http://java.sun.com/javase/; visited on October 22th 2007.

[141] Sun Sun Developer Network (SDN). Remote Method Invocation (RMI). Website, 1999. Available online at http://72.5.124.55/javase/technologies/core/basic/rmi/index.jsp; visited on October 22th 2007.

[142] Sun Sun Developer Network (SDN). What are Web Services? Website, 2003. Available online at http://java.sun.com/developer/technicalArticles/J2EE/new1_4/; visited on October 22th 2007.

[143] H. Seidelmeier. *Prozessmodellierung mit ARIS. Eine beispielorientierte Einführung für Studium und Praxis*. Vieweg Verlag, 2002.

[144] D. Sell, F. Hakimpour, J. Domingue, E. Motta, and R. C. S. Pacheco. Interactive Composition of WSMO-based Semantic Web services in IRS-III. *In In proceedings of the First AKT Workshop on Semantic Web Services (AKT-SWS04) KMi*, The Open University, Milton Keynes, UK, 2004.

[145] J. Siedersleben. *Moderne Softwarearchitektur*. dpunkt.verlag, Heidelberg, 2005.

[146] E. Sirin, J. A. Hendler, and B. Parsia. Semi-automatic Composition of Web services using Semantic descriptions. *WSMAI*, ICEIS Press:17–24, 2003.

[147] E. Sirin, B. Parsia, and J. Hendler. Template-based Composition of Semantic Web services. *In AAAI Fall Symposium on Agents and the Semantic Web*, Virginia, USA, 2005.

[148] E. Sirin, B. Parsia, D. Wu, J. A. Hendler, and D. S. Nau. HTN Planning for Web service Composition using SHOP2. *International Semantic Web Conference*, Semantic Web:377–396, 2004.

[149] R. Smith, D. F. Ferguson, and S. Weerawarana. Web Services Endpoint Language (WSEL). Website, 2001. Available online at http://www.w3.org/2001/04/wsws-proceedings/rod_smith/img13.htm; visited on August 22th 2007.

[150] Sonic Software. SOA Maturity Modell (SOA MM). Website, 2005. Available online at http://www.sonicsoftware.com/solutions/service_oriented_architecture/soa_maturity_model/index.ssp; visited on September 22th 2007.

[151] software design & management (sd&m). *Tagungsband sd&m-Konferenz 2003*. sd&m, ICM München, 2003.

[152] C. Solleder. Business Process Management: Ein ganzheitlicher Ansatz zur Etablierung und Umsetzung einer Methodik für prozessorientierte Projekte – Mit Bezug auf ein Industrieprojekt. Master's thesis, Ulm University, Faculty of Computing Science, 2004.

[153] TopicMaps.Org Specification. XML Topic Maps (XTM) 1.0. Website, 2001. Available online at http://topicmaps.org/xtm/index.html; visited on January 10th 2008.

[154] S. Staab and R. Studer. *Handbook on Ontologies*. Springer Verlag, 2005.

[155] H.-P. Steiert. Towards a Component-based n-Tier C/S-Architecture. *3rd Int. Software Architecture Workshop (ISA W3), Orlando*, pages 137–140, 1998.

[156] K. Stöhr. Aspekte der Abbildung von Geschäftsprozessen, spezifiziert im Business Process Definition Metamodel, in BPEL4WS. Master's thesis, Leipzig University, Faculty of Mathematics and Computing Science, 2005.

[157] M. Stollberg, E. Cimpian, A. Mocan, and D. Fensel. A Semantic Web Mediation Architecture. *1st Canadian Semantic Web Working Symposium (CSWWS2006)*, Quebec, 2006.

[158] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197, 1998.

[159] C. Szyperski, D. Gruntz, and Stehpan Murer. *Component Software. Beyond Object-Oriented Programming*. ACM Press, New York, 2002.

[160] S. Thatte. XLANG – Web Services for Business Process Design. Website, 2001. Available online at http://www.ebpml.org/xlang.htm; visited on August 22th 2007.

[161] T. Tudorache. *Employing Ontologies for an Improved Development Process in Collaborative Engineering*. PhD thesis, Berlin University, Faculty of Compunting Science, 2006.

[162] Fraunhofer Institut Arbeitswirtschaft und Organisation (IAO). Ubiquitous Semantic Web Services (USWS). Website, 2006. Available online at http://www.usws.iao.fraunhofer.de/main_01.html; visited on September 22th 2007.

[163] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barrios. Workflow Patterns. *Technical report, Distributed and Parallel Databases*, 14(1):5–51, 2003.

[164] W3C. Web Services Glossary. Website, 2004. Available online at http://www.w3.org/TR/ws-gloss/; visited on October 13th 2007.

[165] W3C. Semantic Web Services Framework (SWSF) Overview. Website, 2005. Available online at http://www.w3.org/Submission/SWSF/; visited on October 28th 2007.

[166] D. Wackerow. *MQ Series Primer*. MQ-EAI Center, 1999.

[167] S. Weerawarana, R. Chinnici, J.-J. Moreau, and A. Ryman. Web services Description Language (WSDL) version 2.0: Core language. Website, 2007. Available online at http://www.w3.org/TR/2007/PR-wsdl20-20070523/#Endpoint; visited on August 22th 2007.

[168] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture. SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and more.* Pearson Education, Inc., Upper Saddle River, NJ 07458, 2005.

[169] Workflow Management Coalition (WfMC). Process Thought Leadership. Website, 1993. Available online at http://www.wfmc.org/; visited on October 28th 2007.

[170] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49, 1992.

[171] D. Woods and T. Mattern. *Enterprise SOA – Designing IT for Business Innovation*. O'REILLY Cambridge, 2006.

[172] XML-PRC.COM. XML-RPC Specification – Simple cross-platform distributed computing, based on the standards of the Internet. Website, 1998. Available online at http://www.xmlrpc.com/spec; visited on October 22th 2007.

[173] O. Zimmermann, M. Tomlinson, and S. Peuser. *Perspectives on Web Services – Applying SOAP, WSDL and UDDI to Real-World Projects*. Springer Verlag Berlin, 2003.

[174] Heinz Züllighoven. *Object-Oriented Construction Handbook: Developing Application-oriented Software with the Tools and Materials Approach*. Morgan Kaufmann Publishers Inc, US, 2004.