Service-Interoperabilität für naturwissenschaftliche Anwendungen:

Identifikation und Anpassung von komponentenbasierten Service-Mediatoren

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Uwe Radetzki

aus

Bensberg

Bonn

2005

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn. Diese Dissertation ist auf dem Hochschulschriftenserver der ULB Bonn `http://hss.ulb.uni-bonn.de/diss_online` elektronisch publiziert.

1. Referent: Univ.-Prof. Dr. Armin B. Cremers

2. Referent: Univ.-Prof. Dr. Ulf Leser

Tag der Promotion: 20.12.2005

Erscheinungsjahr: 2005

Für Svenja

Give her two red roses, each with a note.
The first note says "For the woman I love" and the second, "For my best friend".
— Anonymous

Abstract

Service-orientation is a new software paradigm for building distributed, component-based software. It allows the aggregation of loosely coupled *services* into value-added workflows. In this context the gap between heterogeneous services is an accepted problem with particular commercial interest. Thus, there is the need to create *service interoperability* semi-automatically.

In this thesis the concept of *service mediators* is developed. Through a *software-aided procedure* service mediators are identified, adapted and integrated into workflows in order to bridge the heterogeneity of different services. Service mediators are software components realizing for instance transformation facilities. The open architecture of the developed procedure allows the integration of benefits from current approaches. The discovery of relevant service mediators is a difficult problem, especially if several service mediators have to be combined adequately to reach the desired service interoperability. One major challenge is that such compositions have to be identified during discovery.

The discovery and adaptation phases of the software-aided procedure require a suitable description of the capabilities of service mediators. Such a description should contain both syntactical and semantical information. The OWL-based *Mediator Profile Language* (MPL) addresses these issues. MPL permits among other things the description of *compositions* of service mediators as well as their *customization* by *stateful properties*. Semantical information is assigned by concepts of a domain ontology.

In this dissertation different *matchmaking algorithms* were developed supporting the user in identifying relevant service mediators as well as new compositions of service mediators. Requirements for service mediators are derived from service descriptions and represented by query profiles in MPL. Even though the service descriptions are fuzzy the *query generation algorithm* automatically creates semantical annotations by mapping syntactical information to concepts of the domain ontology. These annotations are also stored within the query profile. Due to the application of the domain ontology the discovery process enables not only syntactical matchmaking but also semantical matchmaking. Furthermore, the matchmaking algorithms were transferred to the problem of *discovering service operations*. By measuring precision and recall it could be shown that ontology-based matchmaking is advantageously over standard information retrieval techniques.

Kurzfassung

In der Softwareentwicklung wird die *Serviceorientierung* als neues Realisierungsparadigma propagiert. Sie erlaubt lose gekoppelte *Services* bedarfsbezogen in *Workflows* zu aggregieren. Hierbei ist die Überbrückung der Heterogenität dieser Services ein anerkanntes Problem von hohem wirtschaftlichem Interesse. Es besteht der Bedarf die *Service-Interoperabilität* weitestgehend automatisch herzustellen.

In dieser Arbeit wurde ein Konzept für *Service-Mediatoren* entwickelt, die über eine offene und erweiterbare, *software-unterstützte Prozedur* (semi-)automatisch identifiziert und problembezogen in einen Workflow eingebettet werden können. Service-Mediatoren überbrücken die Heterogenität der einzelnen Services und erzielen so die geforderte Service-Interoperabilität. Die offene Architektur und Entwicklung dieser Prozedur erlaubt erstmals die Vorteile gängiger Ansätze zu integrieren. Um einmal entwickelte Service-Mediatoren in verschiedenen Workflows einsetzen und wiederverwenden zu können, bedarf es ihrer gezielten Identifikation und Anpassung. Leider stellt gerade die Suche nach benötigten Service-Mediatoren ein besonders schwieriges Problem da. Dies gilt insbesondere, wenn erst mehrere geeignet verknüpfte Service-Mediatoren zusammen die Service-Interoperabilität erreichen und bereits bei der Suche diese Kombination identifiziert werden muss.

Die Aspekte der Suche und der Anpassung erfordern eine Beschreibungssprache, die die Fähigkeiten eines Service-Mediators sowohl syntaktisch als auch semantisch beschreiben kann. Mit der *Mediator Profile Language* (MPL) wurde eine derartige, auf OWL basierende Beschreibungssprache entwickelt, die die Grundlage des entworfenen Komponentenmodells der Service-Mediatoren bildet. Sie erlaubt u. a. die Beschreibung der *Komposition* mehrerer Service-Mediatoren, sowie deren *Konfiguration* über *zustandsbehaftete Eigenschaftsfelder*. Die semantische Annotation eines Service-Mediators geschieht hierbei über Konzepte einer Domänenontologie.

Im Rahmen dieser Arbeit wurden *Matchmaking*-Algorithmen zur Suche entwickelt, die eine Identifikation adäquater Service-Mediatoren und deren Komposition erlauben. Durch den Einsatz von Ontologien zur semantischen Annotation der Service-Mediatoren kann die Suche auch über rein syntaktische Merkmale hinaus durchgeführt werden. Trotz der den Servicebeschreibungen innewohnenden Unschärfe wurde ein Verfahren realisiert, welches die Servicebeschreibungen auf MPL abbildet und dabei eine automatische Annotation durch die Konzepte einer Ontologie vornimmt. Die Matchmaking-Algorithmen wurden auch auf das Problem der semantischen Suche nach *Service-Operationen* übertragen. Die entwickelten ontologiebasierten Matchmaking-Verfahren liefern im Vergleich zu Standard-IR-Techniken signifikant bessere Ergebnisse, wie durch entsprechende Benchmarks mit anschließender Messung von *Precision* und *Recall* gezeigt werden konnte.

Danksagung

An dieser Stelle möchte ich allen ganz herzlich danken, die zum Gelingen der vorliegenden Arbeit beigetragen haben.

Besonderer Dank geht an meinen Doktorvater Prof. Dr. Armin B. Cremers für die Betreuung dieser Arbeit und den großen Freiraum, den er mir bei der Bearbeitung eingeräumt hat. Das von ihm aufgebrachte außerordentliche Interesse an dieser Arbeit war mir immer ein Ansporn.

Prof. Dr. Ulf Leser danke ich für das große Engagement, das er dieser Arbeit entgegengebracht hat, und die bereitwillige Übernahme des Koreferats. Viele fruchtbare Diskussionen haben zum Erfolg dieser Arbeit beigetragen.

Für die große Unterstützung der Kolleginnen und Kollegen der Informatik III an der Universität Bonn möchte ich mich bedanken. Bei der Bewältigung meiner Probleme stand ich nie vor geschlossenen Türen. Insbesondere geht mein Dank an Sascha Alda, Dr. Thomas Bode, Dr. Pascal Costanza, Melanie Gnasa, Julia Kuck, Patrick Lay, Prof. Dr. Jens Lüssem, Jürgen Schumacher, Dr. Serge Shumilov und meinen ehemaligen Zimmerkollegen Jörg Zimmermann.

Für die vielen Stunden anregender Diskussionen mit zahlreichen wertvollen Ideen möchte ich mich bei Dr. Gabrielle Witterstein bedanken. Mein Dank gilt auch den Diplomanden der GIS-Gruppe Thomas Erdenberger, Sebastian Mancke und Michael Schaefers.

Der größte Dank gebührt meiner Lebensgefährtin Svenja Schulze-Rauschenbach, die mich in allen Belangen zu jeder Zeit unterstützt hat. Ohne ihr Verständnis wäre die Verwirklichung dieser Arbeit nicht möglich gewesen. Schließlich möchte ich mich bei meiner Familie, der Familie meiner Lebensgefährtin und meinen Freunden für ihre Unterstützung und ihren Zuspruch bedanken.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Naturwissenschaftliche Anwendungsgebiete sind durch eine stetig steigende Anzahl unterschiedlicher Informationsquellen und Analysemethoden charakterisiert, die gegenwärtig häufig als verteilte Dienste bereitgestellt werden. Diese Dienste sind hochgradig heterogen, so dass eine Kopplung häufig nur unter hohem Aufwand und durch erfahrene Entwickler erzielt werden kann. Anwender benötigen für die Beantwortung ihrer Fragestellungen jedoch ein problembezogenes Zusammenspiel verschiedenster Informations- und Analysequellen. Daher besteht von Benutzerseite die Forderung, autonome Dienste flexibel und ohne großen Programmieraufwand in anwendungsbezogene Workflows integrieren zu können. Für die Entwicklung verteilter Dienste wird heute die vom standardisierte Web Services Technologie eingesetzt. Die vorliegende Arbeit adressiert die Service-Interoperabilität innerhalb der Workflows durch eine (semi-)automatische Prozedur, in der benötigte komponentenbasierte Service-Mediatoren identifiziert, angepasst und komponiert werden.

"Where shall I begin, please your Majesty?" he asked.
"Begin at the beginning," the King said, gravely,
"and go on till you come to the end: then stop."
— Lewis Carroll (1832-90), *Alice's Adventures in Wonderland, Ch. 12 (1865)*

Moderne Unternehmen befinden sich im ständigen Fluss der sich ändernden Anforderungen und sich neu ergebenden Geschäftsziele. In gleicher Weise ist die Softwareentwicklung einem stetigen Wandel der Realisierungsparadigmen und Softwarearchitekturen unterworfen, die aus neuen Errungenschaften der Informatik hervorgehen. Das Einbinden dieser neuen Möglichkeiten in die IT-Landschaft eines Unternehmens ist von zentraler Bedeutung, da hierdurch die geschäftsbedingten Änderungen zeiteffizienter und kostengünstiger vollzogen werden können. Seit einigen Jahren herrscht ein breiter Konsens, dass vor allem modul- bzw. komponentenbasierte Ansätze den sich wandelnden Anforderungen der Unternehmen

Rechnung tragen (Stojanovic und Dahanayake, 2005). Durch die Weiterentwicklung der Internettechnologie und die Standardisierung relevanter Übertragungsprotokolle (HTTP(S), SMTP) und Beschreibungssprachen (XML, RDF(S), OWL) zeichnet sich gerade im Bereich der Komponententechnologie eine Evolution hin zu *serviceorientierten Architekturen* (SOA) ab.

In serviceorientierten Anwendungen werden Komponenten durch lose gekoppelte, meist verteilt vorliegende Dienste realisiert. Diese werden bei Bedarf in einen Workflow[1] aggregiert, um die vollständige Anwendung zu beschreiben. Aufgrund ihrer Autonomie und nur bedingt vorhandener Schnittstellen- und Datenformatstandards sind diese Dienste heterogen. Dieses erschwert ihre Kopplung, die häufig nur manuell und unter hohem Zeit- und Kostenaufwand erzielt werden kann. Eine verbesserte Unterstützung des Workflowdesigners bei der Erzielung der *Service-Interoperabilität* ist daher ein zentrales Anliegen an die Informatik und liefert die Motivation der vorliegenden Arbeit. Kürzlich entwickelte technische Interoperabilitätsstandards, die so genannten *Web Services*[2], die auf der *eXtensible Markup Language* (XML) basieren, lösen die Probleme nur sehr bedingt, da sie weder Semantik beschreiben, noch die Schnittstellen der Ressourcen standardisieren. Ein Anwendungsfeld aus den Naturwissenschaften untermauert diesen generellen Sachverhalt.

1.1 Anwendungsfeld Biowissenschaften

Die derzeitig nach dem Stand der Technik entwickelten Applikationen in naturwissenschaftlichen Anwendungsgebieten wie den Biowissenschaften, benötigen eine Vielzahl verschiedener, häufig zueinander inkompatibler Werkzeuge, um die komplexen und hochgradig heterogenen Daten analysieren und auswerten zu können. Die Erfolge der molekularbiologischen Forschung über die letzten Jahre haben allerdings nicht nur zu exponentiell anwachsenden Datenvolumina verschiedener Arten (Gen- und Proteinsequenzen, Microarrays, Proteininteraktionen und Proteinfamilien, etc.), sondern auch zu einer unüberschaubaren Anzahl verfügbarer (öffentlicher) Daten- und Methodenressourcen geführt. So existieren neben der *International Nucleotide Sequence Database Collaboration* bestehend aus der *DNA Data Bank of Japan* (DDBJ)[3], der *EMBL Nucleotide Sequence Database* des *European Bioinformatics Institute* (EBI)[4] und der *GenBank* des *National Center for Biotechnology Information* (NCBI)[5] eine Vielzahl spezialisierter biologischer Datenbanken, die häufig aus ganz konkreten Fragestellungen heraus entstanden sind (Baxevanis, 2003).

[1] In dieser Arbeit gilt: Servicekompositionen werden durch Workflows gebildet. Ein Workflow beschreibt die Art und Weise, wie Services abgearbeitet (Kontrollstrukturen) und wie Daten zwischen den Services ausgetauscht werden (Datenflüsse).

[2] Web Services beschreiben eine Sammlung von (W3C-)Standards zur Realisierung von verteilten Diensten. Sie werden ausführlich in Kapitel 2.3 behandelt.

[3] `www.ddbj.nig.ac.jp/`

[4] `www.ebi.ac.uk/embl/`

[5] `www.ncbi.nlm.nih.gov/Genbank/`

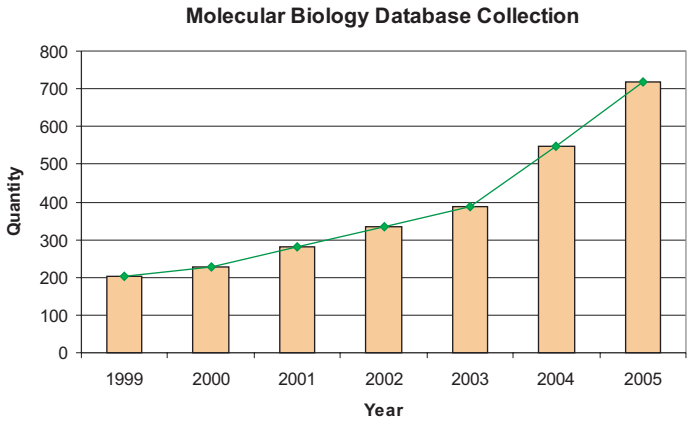

Abbildung 1.1: Trends in der *Molecular Biology Database Collection* (MBDC).

Die jährlich erscheinende *Molecular Biology Database Collection* (MBDC) liefert einen Überblick über die verschiedenen Datenressourcen, die für die Biowissenschaften von großer Bedeutung sind. Abbildung 1.1 verdeutlicht die Entwicklung der MBDC über die letzten Jahre (Burks, 1999; Baxevanis, 2000, 2001, 2002, 2003; Galperin, 2004, 2005). Ein ähnliches Repository liegt mit dem *DBcat Katalog* vor (Discala et al., 2000). Der *BioCatalog* beinhaltet hingegen Referenzen auf Softwarewerkzeuge, die im Kontext der Biowissenschaften Anwendung finden (Rodriguez-Tomé, 1998).

Durch die Autonomie und schnelle Entwicklung innerhalb dieser Anwendungsdomäne ist die Etablierung von anerkannten Standards faktisch nicht möglich, so dass die Ressourcen keinem einheitlichen Datenschema unterliegen. Zudem sind Syntax und Semantik der Schnittstellen der Ressourcen unterschiedlich. Ein Beispiel von ähnlichen Datenressourcen mit unterschiedlichen Strukturen und Formaten sind die verschiedenen Proteinsignaturdatenbanken. Sie wurden entwickelt, um verschiedene Aspekte der funktionalen Klassifikation von Proteinen zu adressieren (Kanapin et al., 2001).

Wie bereits angedeutet, genügt zur Lösung wissenschaftlicher Fragestellungen in den Biowissenschaften die Konsultierung einer einzelnen Informationsquelle nicht, sondern es wird ein Zusammenspiel verschiedener Methoden- und Datenressourcen benötigt (Siepel et al., 2001). Derzeit stellt sich die Verknüpfung biologischer Ressourcen und die damit verbundene Integration der heterogenen und typischerweise semi-strukturierten Daten als weitgehend ungelöstes Problem dar. Die Daten- und Prozessintegration bleibt eine zeitintensive, manuell durchgeführte und häufig fehleranfällige Aufgabe (Backofen et al., 1999; Stevens et al., 2001). Betrachten wir als motivierendes Beispiel *in silico* Experimente aus dem Gebiet der Biowissenschaften:

***In silico* Experimente in den Biowissenschaften**

Die Durchführung von biologischen Experimenten im Labor ist kostenintensiv und zeitaufwändig. Daher wird seit einiger Zeit versucht, diese Experimente teilweise *in silico*, d.h. im Rechner, durchzuführen und damit die Arbeiten im Labor effizienter zu gestalten und Kosten einzusparen. Das Resultat der *in silico* Biologie liefert mögliche Lösungskandidaten der experimentell untersuchten biologischen Fragestellung, die anschließend im Laborversuch verifiziert werden müssen. Die geschickte Kopplung von rechnergestützten Experimenten und Laborexperimenten soll neben der Kostensenkung auch neue Einsichten in die Funktionsweise der komplexen Biologie liefern.

In silico Experimente benötigen das Zusammenspiel von Daten- und Methodenressourcen, die in problembezogener Art und Weise mittels eines Workflows kombiniert und automatisch abgearbeitet werden. Stevens et al. (2004) haben beispielsweise ein *in silico* Experiment auf Basis eines Workflows mit verschiedenen Services entwickelt, mit dessen Hilfe sie neue Kandidatengene im Zusammenhang mit dem *Williams-Beuren Syndrom* identifizieren konnten.

Derartige Workflows können sehr umfangreich werden, doch auch schon bei kleineren Kompositionen zeigt sich der Bedarf nach Techniken, die sich der Heterogenität der Services stellen und Interoperabilität der komponierten Dienste untereinander ermöglichen (*Service-Interoperabilität*) (Radetzki et al., 2004b). Dies ist eine Grundvoraussetzung dafür, dass sich Anwender auf ihre wesentliche Arbeit konzentrieren können: die Entwicklung neuer *in silico* Experimente.

1.2 Forschungsgegenstand und Beitrag der Arbeit

Die Interoperation und Integration von Informationen aus verschiedenen Datenquellen, Internetdiensten und Applikationen ist ein zentrales Thema der Informatik. Der Stellenwert dieses wissenschaftlichen Gebietes hat durch die rasche Verbreitung des Internets, der Verbesserung der Kommunikationstechnologie, sowie durch die weltweite Verfügbarkeit von Informationen noch weiter an Bedeutung gewonnen (Carey et al., 1995; Naumann und Leser, 1999). So fordert beispielsweise schon Hofestädt (1999), dass Methoden zu entwickeln sind, die es ermöglichen, beliebige Daten- und Analyseressourcen flexibel und benutzergerecht zu integrieren.

Durch die Standardisierung von XML und den darauf basierenden Web Services ist man dem Ziel der Interoperabilität auf technischer Ebene einen großen Schritt näher gekommen[6]. Die Herausforderungen, die sich bei der Erstellung von Servicekompositionen durch

[6]Die technische Interoperabilität wurde dadurch erreicht, dass sich alle großen IT-Anbieter auf Protokolle wie SOAP und WSDL geeinigt haben. Es ist daher möglich, Service, die in .NET realisiert wurden, auch auf SUN ONE Plattformen zu benutzen. Dies war auch bei CORBA angedacht, doch weitestgehend nicht erreicht, da Anbieter propriätere Erweiterungen vornehmen konnten.

die syntaktische und semantische Heterogenität der Dienste ergeben, sind derweil weitestgehend ungelöst, da mit einer Vielzahl unterschiedlicher Dienste und Datenformate umgegangen werden muss (Fensel und Bussler, 2002). Aus diesem Grund benötigt die Interoperation heutzutage häufig ein hohes Maß an Benutzerwissen und Benutzerexpertise, um heterogene, autonome Systeme, die *a priori* nicht dafür konzipiert wurden, miteinander zu kombinieren und interagieren zu lassen.

In dieser Arbeit wird dieses Problem der Service-Interoperabilität in servicebasierten Workflows aufgegriffen und die Frage behandelt, wie diese weitestgehend automatisch erzielt bzw. wie der Workflowdesigner bei seiner Arbeit effektiv unterstützt werden kann. Gegenwärtige Ansätze, die diesen Aspekt adressieren, setzen entweder auf Standardisierung, beispielsweise OGC-Services (Harrison und Reichardt, 2001), oder auf eine automatische Herleitung benötigter Verbindungselemente, beispielsweise durch Schema-Matching (Rahm und Bernstein, 2001). Diese Ansätze stoßen jedoch schnell an ihre Grenzen:

- Standardisierung ist ein zeitaufwändiges Unterfangen, welches aktuellen Anforderungen nicht genügen kann. Zudem können die Anforderungen von hoch-spezialisierten Applikationen nur teilweise abgedeckt werden.
- Rein automatischen Ansätzen mangelt es an der Fähigkeit komplexe Verbindungselemente generieren zu können, die vor allem in naturwissenschaftlichen Anwendungsdomänen benötigt werden. Ferner besteht häufig keine Möglichkeit, einmal generierte Einheiten in einem anderen Workflow anpassen und wiederverwenden zu können.

Dabei wäre gerade die Anpassung und Wiederverwendung von einmal erstellten oder generierten Verbindungselementen, die zudem vorhandene Standardisierungsbemühungen ausnutzen und integrieren, ein viel versprechender Ansatz. Hierbei stellt sich die Frage, wie Wiederverwendbarkeit erreicht und vorhandene Ansätze in den Prozess der Workflow-Erstellung nahtlos integriert werden können?

Hauptbeitrag 1: Prozedur zur Erzielung der Service-Interoperabilität

In dieser Arbeit wurde eine offene und erweiterbare software-unterstützte Prozedur zur Erzielung der Service-Interoperabilität als Bestandteil der Workflow-Erstellung entwickelt, die auf Basis eines gegebenen Workflows benötigte Verbindungselemente (im Folgenden *Service-Mediatoren*) identifiziert und in den Workflow integriert. Dabei bilden die Service-Mediatoren die eigentliche Basis zur Überbrückung der Heterogenität der Dienste. Innerhalb dieser Prozedur ist vor allem der Suchprozess nach Service-Mediatoren ein essentieller aber auch problematischer Bestandteil.

Die Prozedur wird durch eine prototypische Laufzeitumgebung sowie entsprechende Benutzerschnittstellen (GUIs) realisiert und gliedert sich in Anfragegenerierung, (Q*uery Generation*), Auffinden (D*iscovery*), Anpassung (A*daptation*) und Bekanntmachung (S*torage*),

kurz QDAS. Die offene Architektur und Entwicklung der QDAS-Prozedur zur Erzielung der Service-Interoperabilität ermöglicht es, vorhandene Verfahren zum Schema-Matching in den Anpassungsprozess sowie zukünftige Retrievalverfahren in das Discovery zu integrieren. Da Service-Mediatoren insbesondere vorhandene Standards berücksichtigen können bzw. sollten, wurde erstmals ein Verfahren realisiert, das die Vorteile gängiger Ansätze integriert. ■

Die Entwicklung einer derartigen Prozedur alleine genügt nicht, um einmal entwickelte Service-Mediatoren in verschiedenen Workflows einsetzen zu können. Einerseits bedarf es verschiedener Anpassungsmöglichkeiten, um möglichst universal einsetzbare und wiederverwendbare Service-Mediatoren zu entwickeln, andererseits müssen entwickelte Service-Mediatoren mit hoher Precision und hohem Recall wiederentdeckt werden, um die Entwicklung neuer interoperabler Workflows kostengünstiger und zeiteffizienter durchzuführen.

Beide Aspekte benötigen eine Beschreibungssprache, die die Definition sowohl der Anpassungsmöglichkeiten als auch der Transformations- bzw. allgemein der Berechnungsoperationen ermöglicht. Dabei sollte neben der Syntax auch die Semantik berücksichtigt werden, ohne die eine semantische Suche nach Service-Mediatoren nur bedingt möglich ist. Gegenwärtige Standards der Web Services Technologie erfüllen diese Anforderungen nicht. Es sind derzeit keine Forschungsarbeiten bekannt, die alle diese Aspekte stringent zusammenbringen.

Hauptbeitrag 2: Die *Mediator Profile Language* (MPL)

Mit der Mediatorprofilsprache MPL wurde eine Beschreibungssprache entwickelt, die die Grundlage für das *Komponentenmodell* der Service-Mediatoren bildet. Zudem wurde eine prototypische Laufzeitumgebung realisiert, die in MPL beschriebene Service-Mediatoren explorieren und ausführen kann (*Komponentenplattform*).

MPL erlaubt es die Fähigkeiten eines Service-Mediators technisch, syntaktisch und semantisch zu spezifizieren. Die semantische Auszeichnung erfolgt hierbei über Konzepte einer Domänenontologie, die verwendete Datenformate und Elemente der Datenformate miteinander in Beziehung setzt.

Kernbestandteile des Komponentenmodells der Service-Mediatoren sind Anpassungsmöglichkeiten, die ebenfalls in MPL ausgedrückt werden können. Hierzu zählen die Konfigurierung über *zustandsbehaftete Eigenschaftsfelder* (*stateful properties*) sowie die Komposition mehrerer Service-Mediatoren (*komplexe Mediatorfunktionalitäten*). Dabei wurden die Kompositionsmöglichkeiten bewusst auf wesentliche Kontrollstrukturen beschränkt, um auf einfache Art und Weise Kompositionen beschreiben zu können und den Umgang des Benutzers mit MPL vergleichsweise einfach zu gestalten. Werden komplexere Kontrollstrukturen benötigt, können diese direkt als atomare Service-Mediatoren in einer konkreten Programmiersprache realisiert werden.

Das Besondere an MPL ist neben der semantischen Auszeichnung das Zusammenbringen von verschiedenen Anpassungsmöglichkeiten. So lassen sich Eigenschaftsfelder (Konfigurierung) mit komplexen Mediatorfunktionalitäten (Komposition) kombinieren. Auf diese Weise wird die Beschreibung konfigurierbarer, komplexer Service-Mediatoren ermöglicht. ■

Die Möglichkeit anpassbare Service-Mediatoren definieren zu können, unterstützt das Entwickeln wiederverwendbarer Softwareeinheiten. Dies allein garantiert leider noch nicht die tatsächliche Wiederverwendbarkeit der Komponenten. Hierzu bedarf es zusätzlich der Möglichkeit, aus einer Menge von Service-Mediatoren die wesentlichen, für eine gegebene Aufgabe benötigten Service-Mediatoren mit hoher Precision und hohem Recall identifizieren zu können (*Matchmaking*). Dabei sollten außerdem mögliche Kompositionen direkt vom System vorgeschlagen werden. Auch für diese Problemstellungen liefert die vorliegende Arbeit eine Lösung.

Hauptbeitrag 3: Ontologiebasiertes Discovery

Zur Suche nach Service-Mediatoren sind in dieser Arbeit verschiedene Matchmaking-Verfahren entstanden, die prototypisch realisiert wurden. Einige dieser Verfahren dienen eher der Filterung, da sie den Suchraum effizient reduzieren, andere liefern bessere Precision-Werte und erkennen neue Kompositionen von Service-Mediatoren.

Die entwickelten Verfahren und die Beschreibungssprache MPL wurden außerdem auf das Problem der semantischen Suche nach *Web Services Operationen* übertragen. Hierzu wurden WSDL-basierte Servicebeschreibungen automatisch auf MPL-basierte Beschreibungen abgebildet und anschließend für das Discovery genutzt.

Über entsprechende Benchmarks mit anschließender Messung von Precision und Recall konnte gezeigt werden, dass insbesondere ontologiebasierte Verfahren im Vergleich zu Standard-IR-Techniken, wie TF-IDF, signifikant bessere Ergebnisse liefern. Dieses Resultat unterstützt zudem die Designentscheidung, sowohl Semantik als auch Syntax eines Service-Mediators in MPL sauber abzubilden. ■

Die Service-Interoperabilität wird durch die in MPL realisierte Anpassbarkeit – und die dadurch verbesserte Wiederverwendbarkeit – der Service-Mediatoren, die entwickelten Retrieval-Verfahren und die mögliche Integration vorhandener Verfahren erleichtert, wodurch die Entwicklung neuer Workflows kostengünstiger und zeiteffizienter durchgeführt werden kann. Dies gilt vor allem dann, wenn vorhandene Workflows modifiziert werden (Austausch einzelner Services) oder einzelne Sub-Workflows Bestandteil eines neuen Workflows werden (Wiederverwendung mehrerer komponierter Services). Die Robustheit und Effektivität der entwickelten QDAS-Prozedur konnte anhand verschiedener Experimente mit anschließender Messung von Precision und Recall gezeigt werden.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt: Nach dieser Einleitung werden in Kapitel 2 die Konzepte service- und komponentenorientierter Software vorgestellt. Der gegenwärtige Status Quo der Web Services Technologie, sowie die Eigenschaften der Komponententechnologie sind Gegenstand dieses Kapitels.

In Kapitel 3 wird der Begriff der Interoperabilität im Kontext der komponenten- und serviceorientierten Software aufgegriffen und diskutiert. In diesem Rahmen wird auch das Konzept der Ontologien zur Formalisierung der Semantik eines Anwendungsgebietes behandelt. Ontologien sowie deren Beschreibungssprachen werden in dieser Arbeit intensiv genutzt. Durch ihren Einsatz kommen wir der semantischen Interoperabilität einen entscheidenden Schritt näher.

Kapitel 4 behandelt die komponentenbasierten Service-Mediatoren und stellt sie gängigen Web Services gegenüber. Es werden die softwaretechnischen Anforderungen erhoben und beschrieben, wie sich diese mit Hilfe der Komponententechnologie lösen lassen. In diesem Kapitel wird ebenfalls das Komponentenmodell vorgestellt, das aus der Auszeichnungssprache MPL und den Regeln zur Mediatorkomposition besteht.

Die Vorstellung der software-unterstützten Prozedur zur Erzielung der Service-Interoperabilität findet in Kapitel 5 statt. Dabei steht der Suchprozess im Zentrum des Kapitels. Der Suchprozess gliedert sich in die Anfragegenerierung, die aus den vorhandenen WSDL-Beschreibungen ein Anfrageprofil in MPL erstellt, dem Discovery, welches auf Basis des Anfrageprofils möglichst gute Service-Mediatoren identifiziert, und der abschließenden Anpassung der Service-Mediatoren an den konkreten Workflow.

In Kapitel 6 wird ein motivierendes Anwendungsbeispiel aufgespannt, anhand dessen die QDAS-Prozedur diskutiert wird. Weiterhin wird der Workflow des Anwendungsbeispiels sukzessiv modifiziert, um Robustheit und Effektivität der entwickelten Verfahren analysieren zu können. Die realisierten Algorithmen werden abschließend auf das Problem der semantischen Suche nach Web Services Operationen übertragen und analysiert.

Kapitel 7 fasst die Ergebnisse der Arbeit zusammen und beschreibt zukünftige Arbeiten in dem Gebiet der serviceorientierten Softwareentwicklung.

Im Anhang A der Arbeit findet ein Vergleich der Web Services Technologie zu den Middlewarekonzepten CORBA, Grid und Peer-To-Peer statt. Ferner werden zwei konkurrierende Realisierungen vorgestellt und softwaretechnisch verglichen. Abschließend werden das MPL-Schema sowie einige Beispiele in Anhang B aufgeführt. In diesem Kapitel wird ferner die Semantik von MPL hinsichtlich der Kompositionsfähigkeiten über Petri-Netze formal beschrieben und die entwickelten Benchmarks vorgestellt.

Der Fokus der Arbeit liegt auf der syntaktischen und semantischen Verknüpfung der Web Services durch geeignete Service-Mediatoren. Nichtfunktionale Anforderungen an die Laufzeitumgebung der Service-Mediatoren, wie beispielsweise Sicherheit, Performanz, Ausfallsicherheit und ähnliches, werden in dieser Arbeit nicht explizit behandelt. Die Beschreibung

und der Entwurf servicebasierter Workflows sind ebenfalls nicht Gegenstand dieser Arbeit. Sie können beispielsweise als Dokument in BPEL4WS vorliegen und durch ein entsprechendes Entwurfswerkzeug erstellt werden.

Der in dieser Arbeit entwickelte Ansatz, die Service-Interoperabilität bei servicebasierten Workflows zu erzielen, wird anhand der Biowissenschaften illustriert, ist aber nicht auf diese Anwendungsdomäne beschränkt. Fragestellung und Ansatz sind domänenunabhängig und lassen sich auf andere Gebiete übertragen, wie beispielsweise in (Radetzki et al., 2004c) für die Geowissenschaften beschrieben. Lediglich entsprechende Domänenontologien und essentiell benötigte Service-Mediatoren sind speziell für die jeweilige Anwendungsdomäne zu realisieren.

Allgemein wird jedes Kapitel durch einen *kurzen, kursiv dargestellten Abriss* eingeleitet, der einen Überblick über die Zielsetzung und die Ergebnisse des Kapitels liefert. Dies soll das zielgerichtete Lesen der Arbeit erleichtern. Da die Arbeit viele vergleichende Ansätze aus unterschiedlichen Gebieten enthält, erscheint es ungünstig diese in einem einzigen Kapitel zu beschreiben. Daher werden am Ende eines jeden Kapitels die wesentlichen Sachverhalte kurz zusammengefasst und vergleichbare Ansätze bedarfsbezogen zu den im Kapitel vorgestellten Methoden diskutiert.

Aus Gründen der Übersichtlichkeit werden in der Arbeit zwei unterschiedliche Zitierweisen verwendet, beispielsweise (Baxevanis und Ouellette, 2001) oder Baxevanis und Ouellette (2001). Beide verweisen auf dieselbe Quelle, jedoch wird letztere Zitatform gezielt eingesetzt, um Doppelnennungen der Autoren und der entsprechenden Quelle in einer Textpassage zu vermeiden.

Die in dieser Arbeit verwendeten *Definitionen* dienen neben der exakten Beschreibung auch der Begriffsklärung und sind daher nicht immer im rein mathematischen Sinne zu interpretieren.

Das hier entwickelte Framework ist unter dem Namen Interoperability and Reusability of Internet Services (IRIS) *bekannt und unter gleichnamigem Projekt entstanden. In der griechischen Mythologie ist Iris die Götterbotin. Sie wird durch den Regenbogen symbolisiert und ist die personifizierte Gottheit des Regenbogens, der sowohl Himmel wie Erde berührt. Dieses ist die Verbindung zwischen den Göttern und den Sterblichen, über die das Medium Iris Nachrichten der Götter den Menschen überbringen kann.*

„She is often shown gliding down a rainbow to deliver her messages to mortals, who looked on her as a guide and adviser. Her name also means 'rainbow', thus implying that her presence is a sign of Hope.“[a]

[a] Elysium-Gates (2001)

Kapitel 2

Konzepte service- und komponentenorientierter Software

Service- und komponentenorientierte Ansätze dienen der Modularisierung und Auslagerung von Softwarebausteinen sowie der Kosten- und Risikenreduzierung. Serviceorientierte Architekturen (SOA) bilden zudem das Workflow Management auf die allgegenwärtige Internet-Plattform ab und erlauben die Realisierung organisationsinterner oder -übergreifender Geschäftsprozesse. SOA werden als Basis für die Integration von verteilten, autonomen Daten- und Methodendiensten herangezogen. Aufgrund der Heterogenität der einzelnen Services und Komponenten gestaltet sich die Komposition jedoch als schwieriges, häufig fehleranfälliges und zeitintensives Unterfangen. Web Services treten der Heterogenität auf technischer Ebene durch Standardisierung entgegen. Semantische und syntaktische Aspekte der Heterogenität bleiben bei der Bildung komplexer Workflows ein weithin ungelöstes Problem.

One must imagine Sisyphus happy.
— Albert Camus (1913-1960), *The Myth of Sisyphus, 1955*

Dieses Grundlagenkapitel führt in die Konzepte der service- und komponentenorientierten Software ein. Anfangs wird die Komponententechnologie vorgestellt und Eigenschaften wie *Anpassbarkeit* und *Wiederverwendbarkeit* diskutiert (Abschnitt 2.1). Letztere Aspekte werden für die in Kapitel 4 entwickelten Service-Mediatoren zentral. Im Anschluss wird die Frage erörtert, was sich hinter dem Schlagwort „*Service Oriented Architecture*“ (SOA) verbirgt und wie diese neuen Technologien im Zusammenhang mit objekt- und komponentenorientierten Konzepten stehen (Abschnitt 2.2). In dem darauffolgenden Abschnitt wird die standardisierte Web Services Technologie als die derzeit gängige Realisierung vorgestellt (Abschnitt 2.3). Web Services sind lose gekoppelte, verteilt vorliegende Softwareeinheiten, die bedarfsbezogen verknüpft werden können, um ein gegebenes Geschäftsproblem zu lösen. Diese Verknüpfung geschieht häufig in Form eines Prozesses, der durch einen Workflow definiert wird. Ein kurzer Überblick über diese Begrifflichkeiten findet sich in Abschnitt 2.4.

2.1 Komponententechnologie

Schon seit langer Zeit ist bekannt, dass sich ein Produktionsprozess kostengünstiger und effizienter gestalten lässt, wenn man sich der Modulbauweise bedient. In der Industrie und den Ingenieurswissenschaften ist diese Methode daher schon lange etabliert. Die Modulbauweise ist beispielsweise beim Fahrzeugbau weit fortgeschritten. Auch in der Softwaretechnologie findet dieses Verfahren Anwendung. Im Allgemeinen lassen sich Kosten und Risiken senken, wenn komplette Applikationen durch die Komposition einzelner vorgefertigter und erprobter Softwareeinheiten realisiert werden.

Diese komponentenbasierten Ansätze sind in der Softwaretechnologie nicht neu. Die Forderung, Softwarekomponenten zur Massenproduktion in der Softwareindustrie einzusetzen, wurde erstmals von McIlroy (1968) aufgestellt. Und bereits mit der Einführung von Modulen in den Programmiersprachen Ada und Modula-2 Ende der 70er-Jahre sind die Grundstöcke für die moderne Komponententechnologie entstanden. Trotz der Etablierung der Komponententechnologie sind derweil nicht alle Fragen verstanden und gelöst. Beispielsweise gibt es keine Einigkeit über Fragen hinsichtlich des Zustands (*hat die Komponente einen Zustand oder nur ihre Instanzen bzw. sollte man zwischen Komponente und Instanzen einer Komponente überhaupt unterscheiden?*) oder der Instantiierung (*existieren Komponenten zur Laufzeit oder nur zur Compilezeit?*) (Nierstrasz und Achermann, 2003). Im Folgenden wird der Begriff der *Komponente* als Synonym für *Softwarekomponente* eingesetzt.

Eine der grundlegenden Arbeiten im Bereich der Komponentensoftware, die unter anderem die obigen Fragen adressiert, ist die von Clemens Szyperski (1998). Er definiert eine Softwarekomponente wie folgt:

> „*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 parties.*“

Nach dieser Definition sollten Komponenten ihre Kontextabhängigkeiten anderen Komponenten und Systemen explizit darlegen (*explizit context dependencies only*). Kontextabhängigkeiten beschreiben die Erfordernisse einer Komponente an ihre Einsatzplattform sowie an andere Komponenten, ohne die eine Komponente nicht ausgeführt werden kann. Das Offenlegen der Kontextabhängigkeiten erhöht die Anpassbarkeit von Komponenten sowie die Wartbarkeit von komponentenbasierter Software und ermöglicht die Integration neuer Komponenten in bereits bestehende Kompositionen (Alda und Cremers, 2003). Ohne das Wissen über die vorhandenen Kontextabhängigkeiten kann es bei der Wiederverwendung einer Komponente zu Problemen kommen: Wird eine Komponente in einem neuen Anwendungsgebiet eingesetzt, funktioniert sie eventuell nicht wie erwartet. Dies kann daran liegen, dass die Komponente Voraussetzungen annimmt, die in der originalen Umgebung gegeben waren, jedoch nicht in dem neuen Kontext (Uschold und Gruninger, 1996).

Die Aspekte der Wiederverwendbarkeit (*reusability*) und der Anpassbarkeit (*adaptability*) werden in Abschnitt 2.1.1 beziehungsweise in Abschnitt 2.1.2 näher betrachtet.

Eine Menge von Regeln und Spezifikationen über die Art und Weise, wie Komponenten erstellt und komponiert werden können, wird im Allgemeinen als *Komponentenmodell* bezeichnet. Ferner wird für das Deployment und die Installation eine spezifische *Komponentenplattform* benötigt, die das entsprechende Komponentenmodell unterstützt. Zu den bekanntesten Komponentenmodellen gehören CORBA, JavaBeans und EJBs der Java-Plattform sowie COM und Assemblies der .NET-Plattform. Auch die im nächsten Abschnitt beschriebenen Web Services kann man als *virtuelle Komponenten*[1] bezeichnen. Sie stellen eine Weiterentwicklung des bekannten komponentenbasierten Paradigmas dar. Die Weiterentwicklung besteht in einer loseren Kopplung und einer höheren Dynamik (Auffinden und Ersetzen) der Services und soll zukünftig die Bildung virtueller Organisationen unterstützen (vgl. Anhang A).

Komponentenmodul

Doch was sind die konkreten Eigenschaften einer Komponente bzw. die Unterschiede zwischen einem Komponentenmodul im Gegensatz zu einer Komponenteninstanz? Hierzu werden in (Szyperski et al., 2002) folgende Charakteristiken aufgestellt: Eine *Komponente*, genauer ein *Komponentenmodul* ist eine Einheit, die

- unabhängig eingesetzt (*deployment*),
- von Dritten mit anderen Komponenten kombiniert werden kann (*third-party composition*), und
- die keinen (extern) beobachtbaren Zustand besitzt.

Diese Eigenschaften haben direkte Konsequenzen für den Umgang mit Komponenten. Durch das Deployment als Einheit ist es nicht möglich, eine Komponente nur teilweise in einer Plattform einzusetzen. Ferner erzwingt die Kompositionsfähigkeit durch Dritte, dass Komponenten als in sich geschlossen (*self-contained*) verstanden werden müssen, über eine vertraglich zugesicherte Schnittstelle verfügen und ihre Implementierung über diese kapseln. Aufgrund der Tatsache, dass sie keinen beobachtbaren Zustand besitzen, lässt sich eine Komponente nicht von ihren Kopien unterscheiden.

[1] Diese Komponenten werden als *virtuell* bezeichnet, da sie zum einen nicht unabhängig *deployed* werden können und zum anderen nur eine Fassade vor konkret realisierten Komponenten bilden. Das heißt, die konkrete Realisierung eines Web Service geschieht gewöhnlich durch ein anderes Komponentenmodell, z.B. durch EJBs oder Assemblies (siehe Anhang A.1).

Komponenteninstanz

Dem gegenüber stehen die Objekte bzw. die konkreten *Komponenteninstanzen*. Diese zeichnen sich aus durch

- eine eindeutige Identität,
- mögliche extern beobachtbare Zustände und
- die Kapselung ihrer Zustände und ihrer Verhaltensweisen.

Somit können verschiedene Komponenteninstanzen unterschieden werden, auch wenn sie dem gleichen Komponentenmodul entstammen.

Im Folgenden ist der Begriff *Komponente* ein Synonym sowohl für den Begriff Komponentenmodul als auch für den Begriff Komponenteninstanz, wenn aus dem Zusammenhang klar wird, um welchen konkreten Begriff es sich handelt.

2.1.1 Wiederverwendbarkeit

Ein zentrales Anliegen der Komponententechnologie ist die Wiederverwendbarkeit (*reusability*) von Komponenten in verschiedenen Anwendungskontexten und somit eine Reduzierung der Entwicklungskosten und des *Time-To-Market*. Jedoch ist auch die Wiederverwendung mit Risiken verbunden, wenn dabei von bestimmten Implementierungsmerkmalen einer konkreten Komponente ausgegangen wird. Dieses tritt vor allem im so genannten *Whitebox Reuse* auf.

Beim *Whitebox Reuse* ist die Implementierung der Schnittstelle einer Komponente öffentlich, wodurch sie studiert und gegebenenfalls auch verändert werden kann, beispielsweise, wenn der Quellcode der Komponente mit ausgeliefert wird. Wird die Manipulation nicht erlaubt, spricht man allgemein vom *Glassbox Reuse*. Ein Problem beim Whitebox wie beim Glassbox Reuse ist, dass der Nutzer von einer konkreten Realisierung ausgeht, die sich jedoch bei einer neuen Version der Komponente ändern kann. Somit lässt sich nicht ohne weiteres eine neue Version statt einer alten Version einsetzen und gleichzeitig garantieren, dass das System wie gewünscht weiterarbeitet. Im Allgemeinen ist daher diese Form der Wiederverwendung nicht zu empfehlen.

Beim so genannten *Blackbox Reuse* ist bezüglich der Implementierung einer Schnittstelle nichts bekannt. Auch im Falle der Realisierung einer Komponente durch Aggregation anderer Komponenten sind diese Verknüpfungen weder bekannt noch änderbar. Die Wiederverwendung basiert allein auf der vorhandenen Schnittstelle.

Zusätzlich zum Wiederverwendungsverfahren (Whitebox, Blackbox) stellt sich die Frage nach der Wiederverwendbarkeit einer Komponente. Die Wiederverwendbarkeit einer

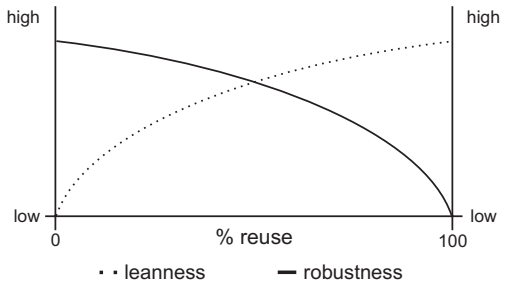

Abbildung 2.1: Grad der Wiederverwendbarkeit in Abhängigkeit der Kräfte Robustheit und Leichtigkeit.

Komponente wird sowohl durch die Anzahl ihrer Kontextabhängigkeiten als auch den Grad ihrer Autarkie (*self-containedness*) und damit der Höhe ihres „Gewichts“ beeinflusst. Diese beiden Kräfte wirken jedoch kompetitiv. Abbildung 2.1 illustriert den Zusammenhang zwischen *Robustheit* (eingeschränkte Kontextabhängigkeiten) und *Leichtigkeit* (eingeschränktes „Gewicht“) (Szyperski et al., 2002): Eine Komponente, die sich durch eine hohe Autarkie auszeichnet, ist relativ robust gegenüber Veränderungen beispielsweise anderer Komponenten, denn sie erfüllt im wesentlichen alle Voraussetzungen durch sich selbst (*self-contained*). Ein Beispiel ist eine Kartenkomponente zur Visualisierung und Verwaltung von Landkarten. Ihre Kontextabhängigkeiten sind sehr gering und sie besitzt eine große Robustheit. Auf der anderen Seite sind Leichtigkeit und Wiederverwendbarkeit gering, da die Komponente nur in speziellen Kontexten eingesetzt werden kann. Um jedoch redundante Implementierung zu verhindern, kann auf der anderen Seite eine Komponente alle nicht direkt zur eigentlichen Funktionalität gehörenden Einheiten auslagern (*out-sourcing*). Dieses erhöht die Wiederverwendbarkeit und Leichtigkeit der einzelnen Einheiten, jedoch nicht ihre Nutzbarkeit, da gleichzeitig auch die Kontextabhängigkeiten steigen. Beispielsweise besitzt die Kartenkomponente eine Komponente zur Verwaltung der einzelnen Darstellungsebenen, die auch in anderen Komponenten eingesetzt werden kann. Diese leichtere Komponente benötigt ganz bestimmte Kontextvoraussetzungen, wie Schnittstellen und Datenformate, die möglicherweise nicht jeder Client liefern kann, obwohl er die konkrete Funktionalität der Komponente benötigt. Somit resultiert eine Maximierung der Wiederverwendbarkeit in einer Minimierung der Nutzbarkeit (Szyperski et al., 2002).

Diesem Paradoxon kann durch *Standardisierung* entgegen gewirkt werden. Werden bestimmte Kontextabhängigkeiten standardisiert und dadurch allgemein zur Verfügung gestellt, dann reduzieren sie nicht mehr die Nutzbarkeit einer Komponente. Standardisierung wird entweder durch offizielle Gremien erreicht (beispielsweise OGC, W3C, OASIS, ISO) oder dadurch, dass sich die großen Anbieter einer Industrie auf bestimmte Schnittstellen und Protokolle einigen. Dies ist der Weg, der mit Web Services gegangen wurde und einen wesentlichen Erfolg dieser Technologie ausmacht.

2.1.2 Anpassbarkeit

Die Wiederverwendbarkeit einer Komponente in unterschiedlichen Anwendungskontexten kann ebenfalls erhöht werden, wenn die Komponente oder die verwendete Komponentenplattform entsprechende Möglichkeiten zur Anpassung bereitstellt. Unter dem Begriff der *Anpassbarkeit* (*adaptability*) versteht man allgemein die Eigenschaft eines technischen Systems, Benutzern die Möglichkeit zu geben, die Funktionalität des Systems entsprechend neuen Anforderungen zu verändern (Henderson und Kyng, 1991). Hierbei kann bzw. sollte das System den Benutzer möglichst geeignet unterstützen (Fischer, 2001). Der Prozess der Endbenutzeranpassung einer Softwareanwendung (beispielsweise Anpassungen der MS Word-Oberfläche) an konkrete Arbeitsbedürfnisse eines Nutzers oder einer Nutzergruppe wird auch als (Endbenutzer-) *Tailoring* bezeichnet (Mørch, 1995).

Henderson und Kyng (1991) haben drei Strategien identifiziert, um das Verhalten eines Systems den Bedürfnissen eines Benutzers anzupassen:

- Die Auswahl zwischen antizipierten Alternativen (vornehmlich Veränderungen von Parametern), wobei zu beachten ist, dass das anzupassende System herausfinden muss, welche Parameter veränderbar sind, welches Verhalten sie ändern, und welche Werte gültig sind.
- Die Konstruktion neuen Verhaltens auf Basis bereits existierender Elemente.
- Die direkte Veränderung des Artefakts, beispielsweise durch Re-Implementierung.

Eine ähnliche Klassifizierung wurde auch von Mørch (1995) vorgenommen. Er unterscheidet zwischen Konfigurierung (*customization*), Integration und Erweiterung (*extension*):

- *Konfigurierung* beschreibt die Möglichkeit ein (Präsentations-)Objekt durch Modifikation vordefinierter Konfigurationsoperationen zu verändern.
- *Integration* wird als Erstellungs- oder Kompositionsvorgang von (vordefinierten) Programmteilen hin zu einer neuen Funktionalität verstanden, die anschließend im System gespeichert wird.
- *Erweiterung* bezeichnet das Hinzufügen von Neuentwicklungen (durch Programmierung) zu einem System, um dessen vorhandene Funktionalität zu erhöhen. In diesem Zusammenhang wird auch von *Plugins* oder *Hooks* gesprochen (vgl. beispielsweise Eclipse-Plugins).

Neben der Anpassbarkeit wird häufig auch der Begriff der *Adaptivität* bzw. *Selbstanpassbarkeit* (*self-adaptability*) eingesetzt, um Systeme zu beschreiben, die sich veränderten Anforderungen anpassen. Im Unterschied zur Anpassbarkeit beschreibt die Adaptivität jedoch

die autonome Anpassung des Systems (Fischer, 2001). Das System erkennt die neuen Anforderungen, beispielsweise durch verändertes Benutzerverhalten, und reagiert mit einer Anpassung entsprechend, d.h. das System ist der Akteur einer Anpassungsaktion, während der Benutzer der Initiator der Veränderung ist.

Fischer (2001) fasst die Vor- und Nachteile dieser beiden Adaptionsvarianten zusammen. Bei den adaptiven Systemen ist der Benutzer in die Adaptionvorgänge kaum involviert und daher auch nicht oder nur wenig mit diesen belastet. Der Nachteil ist, dass der Benutzer gleichzeitig die direkte Kontrolle über die Veränderungen verliert. Ferner ist es für den Benutzer schwierig ein kohärentes Gesamtbild des Systems aufzubauen. Bei den anpassbaren Systemen werden die Veränderungen durch den Benutzer durchgeführt, weswegen sie auch unter dessen Kontrolle stehen. Zudem kennt der Benutzer seine konkreten Aufgaben und Anforderungen am besten. Auf der anderen Seite muss der Benutzer substantielle Arbeit leisten und sich erst die Adaptionsmöglichkeiten des Systems aneignen. Daher ist die Komplexität der Anpassung aus Sicht des Benutzers höher als bei adaptiven Systemen.

Da die Übergänge zwischen Adaptivität und Anpassbarkeit ungeachtet dessen fließend sind (Won, 2004), soll in dieser Arbeit keine strikte Trennung dieser beiden Adaptionsvarianten vorgenommen werden. Stattdessen wird allgemein von Anpassbarkeit gesprochen, unabhängig davon, ob das System, der Benutzer oder beide in einer kooperativen Weise die Anpassungen vornehmen.

Der Aspekt der Anpassbarkeit kann ebenfalls auf Komponentenarchitekturen angewendet werden, bei denen es neben der expliziten Erstellung neuer Komponentenaggregationen auch darum geht, das Verhalten einer existierenden Komposition zu verändern. In diesem Zusammenhang hat Won (2004) die folgenden Adaptionsmechanismen identifiziert:

- Das Ändern der Parameter einer Komponente, d.h. ihre (Re-)Konfigurierung;
- das Ändern der Komponentenzusammenstellung, d.h. das Verändern der Funktionalität einer Komposition (Applikation) durch Hinzufügen oder Entfernen einzelner Komponenten, sowie
- das Ändern der Verbindungen zwischen Komponenten innerhalb einer Komposition (dieser Aspekt der Anpassung erfordert die vorherige Auswahl geeigneter Komponenten);
- das Ändern der Implementierung einer Komponente[2], und
- das Ändern der zur Verfügung stehenden Komponenten für eine Komposition, d.h. durch Hinzufügen oder Entfernen einer Komponente zum Pool der zur Verfügung stehenden Komponenten wird die potentielle Funktionalität einer daraus zu bildenden Applikation modifiziert.

[2]Dieser Aspekt wird nur der Vollständigkeit halber aufgezählt, aber aus oben geschilderten Problemen, wie der Versionsproblematik (vgl. Whitebox Reuse auf Seite 14), nicht empfohlen.

In Kapitel 4.3 wird der Begriff der Anpassbarkeit noch einmal aufgegriffen und im Kontext der Service-Mediatoren betrachtet.

2.2 Serviceorientierte Architekturen

Eine sehr allgemeine und zugängliche Definition des Begriffs *serviceorientierte Architektur* (SOA) wurde von Jody Hunt (2003) der IONA-Gruppe gegeben:

> *„A* SOA *is the principle of information hiding applied to distributed systems, where interface contracts wrap application logic and make that logic accessible (that is, publish it) as service accessed via formal, stable interfaces. [...] A service may even be a business process that is itself the composition of other services.“*

Der Begriff wird als eine logische Konsequenz der objekt- und komponentenorientierten Programmierung auf das Gebiet der verteilten Systeme verstanden, wobei die Objekte (*Services*) und deren Anwendungslogik über vertraglich zugesicherte Schnittstellen bereitgestellt werden. Ein weiterer wichtiger Aspekt der SOA ist die Fähigkeit, komplexe Services durch Komposition anderer Services zu kreieren. An dieser Stelle werden Fragestellungen wie Interoperabilität (Schnittstellen und Daten, aber auch Prozessmodelle der Service Provider), Qualität des Services (*Quality-of-Service* – QoS) und Zuverlässigkeit (*reliability*) deutlich.

Eine etwas einschränkendere Definition liefern Booth et al. (2003) in der Beschreibung zur *Web Services Architecture* (WSA). Hier sagen die Autoren:

> *„A* SOA*, is a specific type of distributed system in which the agents are 'services'. [A] service is a software agent that performs some well-defined operation (i.e., 'provides a service') and can be invoked outside of the context of a larger application. That is, while a service might be implemented by exposing a feature of a larger application [...] the users of that server need be concerned only with the interface description of the service. [...] '[S]ervices' have a network-addressable interface and communicate via standard protocols and data formats.“*

Auffällig ist der direkte Vergleich des Service mit dem Begriff des „*Softwareagenten*“, obwohl in der Definition nicht weiter erläutert. Dieser Vergleich von Services mit Agenten wird in der Literatur häufig gezogen und begründet sich aus der Tatsache, dass die Thematik der Multi-Agentenplattformen Einzug in die Internettechnologie gesucht und gefunden hat. Und tatsächlich lassen sich viele Fragestellungen, die bereits in Multi-Agentenplattformen diskutiert wurden, in dem Gebiet der SOA wiederfinden, u. a. Discovery von Agenten (Services), Sicherheitsaspekte bei Agenten (Services), Komposition von Agenten (Services) um

ein gemeinsames Geschäftsziel zu erreichen und vieles mehr. Daher ist es nicht verwunderlich, dass viele Gruppen aus dem Gebiet der Agententechnologie ihre Arbeiten auf das neue Gebiet der SOA übertragen. Ein prominentes Beispiel ist die Übertragung von Resultaten des LARKS Systems (Sycara et al., 2002) auf OWL-S beschriebene Services (Paolucci et al., 2002).

Ferner – und dies zeigt die zweite Definition – wird häufig für die Dienste gefordert, dass diese eine wohldefinierte Schnittstelle besitzen, auf die über eine ausgezeichnete Netzwerkadresse zugegriffen werden kann, und sie außerdem über standardisierte Protokolle und Datenformate kommunizieren. Diese Anforderungen adressiert heute das Forschungsgebiet der Web Services Technologie.

Die *Serviceorientierung* fordert nicht nur, dass die Dienste über das Netzwerk ortsunabhängig verfügbar und auffindbar sein sollen, sondern liefert auch eine Abstraktionsebene, die die Komplexität der technischen Realisierung der Services maskiert. Somit kann die Geschäftsebene auf eine flexible, dynamische und kosteneffiziente Art und Weise entscheiden, wann und wo der Dienst benötigt wird, ohne sich damit beschäftigen zu müssen, wie die benötigte Funktionalität bereitgestellt wird.

Es lassen sich folgende Vorteile der serviceorientierten Software gegenüber Standardsoftware festhalten (Pallos, 2001): SOA erlaubt es, die getätigten Investitionen einer Organisation, wie Hardware, Programmiersprachen, Erfahrungen der Entwickler, etc., wirksam einzusetzen, da zum einen der Geschäftsprozess durch die Kombination von Diensten beschrieben wird, deren Implementierungsdetails hinter der Schnittstelle versteckt sind, und zum anderen, da sich neue Dienste in den Sprachen realisieren lassen, in denen die Organisation bereits das Know-How besitzt.

Durch die flexible Art und Weise, wie Services zu Geschäftsprozessen kombiniert werden können, lässt sich auch das Time-To-Market, die Kosten und die Risiken reduzieren. Das kürzere Time-To-Market ergibt sich aus dem Umstand heraus, dass immer auf vorhandene Dienste zurückgegriffen werden kann (*reuse*) und die Zahl der so neu entstehende Dienste kontinuierlich wächst. Lassen sich bestimmte Unterprozesse eines Geschäftsprozesses ebenfalls wiederverwenden, reduziert dies zudem die Kosten bei der Erstellung neuer Servicekompositionen. Das Risiko wird ebenfalls durch die Nutzung bereits erprobter und getesteter Dienste gemindert. Ferner werden die bekannten Werkzeuge, wie *Integrated Development Environments* (IDEs), Programmiersprachen und Datenbanken, bei der Implementierung von neuen Diensten genutzt.

Natürlich ist der Einsatz von Software, die rein auf (externen) Diensten aufsetzt, auch mit Risiken verbunden. Bei der Wiederverwendung eines Dienstes können beispielsweise falsche Einheiten der Daten oder falsche Geschäftsprozesse, die sich hinter der Schnittstelle befinden, angenommen werden (*„Macht der Service, was ich von ihm erwarte?“*). Ferner sind die Fragen zu erörtern, inwieweit der verwendete Dienst die geforderte Ausfallsicherheit besitzt; ob die benötigten Sicherheitsanforderungen eingehalten werden; wie effizient die Realisierung ist; oder wie stabil die Schnittstelle des genutzten Dienstes und die verwendeten Protokolle sind.

Das Anwendungsfeld für SOA ist vielschichtig, jedoch sind wesentliche Anwendungsgebiete die der *Enterprise Application Integration* (EAI) und der *Enterprise Information Integration* (EII) (Brandner et al., 2004). Die vorliegende Arbeit adressiert ebenfalls diese Gebiete, indem versucht wird, innerhalb benutzerdefinierter Geschäftsprozesse (*Workflows*) heterogene Serviceschnittstellen durch Service-Mediatoren zu verbinden (vgl. Kapitel 4).

2.3 Die Web Services Technologie

Die Web Services Technologie und XML bilden heutzutage die Grundlage für die Entwicklung von serviceorientierter Software. Diese auf der XML-Familie basierenden Technologien beschreiben nicht nur flexibel und erweiterbar Daten und Inhalte, sondern bieten auch einen standardisierten Weg, verteiltes Rechnen zu realisieren. Sie stellen eine Weiterentwicklung des Webs hinsichtlich automatisierter Verarbeitung dar (Curbera et al., 2001).

Ähnlich wie SOA bleibt auch der Begriff „*Web Services*“ schwer fassbar und es gibt keine allgemein anerkannte Definition dieses Begriffs (Foster et al., 2004b; Kossmann und Leymann, 2004). Eine wesentliche – und bis dato häufig zitierte – Definition wurde in den Anfängen von Doug Tidwell gegeben[3]:

> „Web services *are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes. [...] Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service.*“

Diese Definition bleibt recht unspezifisch und beschreibt nicht die heute gängigen Protokolle, die der Web Services Technologie angehören und den so genannten *Basistechnologie-Stack* bilden. Web Services sind auf eine Maschinen-zu-Maschinen Kommunikation ausgerichtet. Ein weiteres angestrebtes Ziel ist eine hohe Interoperabilität der Dienste. Web Services sollen miteinander kommunizieren können, unabhängig von der eingesetzten Web Services Plattform, der verwendeten Programmiersprache oder des Betriebssystems (Keidl et al., 2003). Hinsichtlich der standardisierten Protokolle, durch die Interoperabilität erreicht werden soll, wird die Definition aus der *Web Services Architecture* (WSA) (Booth et al., 2003), wie sie derzeit auch in dem W3C Web Services Glossar[4] zu finden ist, weitaus konkreter. Denn es wird sowohl auf die Ziele als auch auf Protokolle wie SOAP und WSDL hingewiesen:

> „*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 machine-processable format (specifically WSDL). Other systems interact with the Web*

[3] Doug Tidwell. *Web services – the Web's next revolution.* IBM DeveloperWorks, November 2000.
[4] Hugo Haas und Allen Brown. *Web Services Glossary.* February 2004, `www.w3.org/TR/ws-gloss/`

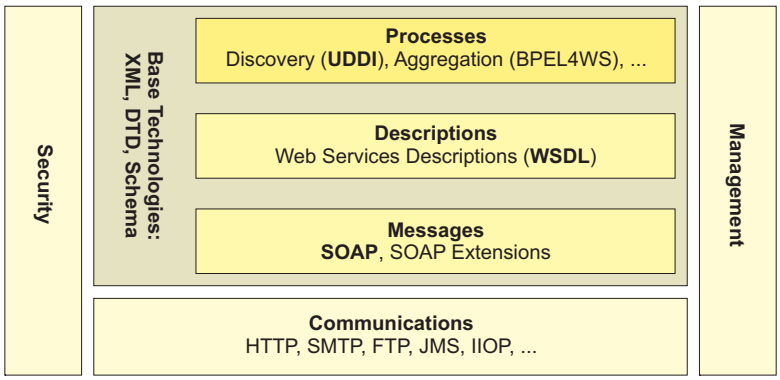

Abbildung 2.2: Web Services Technologie-Stack.

> *service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.“*

Um die babylonische Wortverwirrung zu komplettieren, sei erwähnt, dass – im Gegensatz zu oben definierter Gleichheit zwischen Service und Agent – in der WSA diese unterschieden werden. Ein Web Service wird in der WSA als abstrakte Einheit verstanden, die durch einen konkreten Agenten implementiert wird. Der Agent stellt somit eine konkrete Softwareeinheit dar, die Nachrichten empfangen und senden kann, während der Service eine abstrakte Sammlung von bereitgestellten Funktionalitäten ist. Somit kann jederzeit der Agent, der den Service realisiert, ausgetauscht und verändert werden. Der Web Service, d.h. das definierte Interface, bleibt jedoch unverändert.

Weitere Charakteristika der Web Services Technologie lassen sich wie folgt zusammenfassen (Wojciechowski und Weinhardt, 2002): Web Services sollen *Geschäftslogik kapseln*, ähnlich wie es der Fall ist für Objekte in der Objektorientierung. Sie sind *lose gekoppelt*, da Web Services nur über Nachrichten kommunizieren. Der Aufruf eines Web Service kann von jedem Ort aus geschehen, d.h. sie sind *ortstransparent*. Ferner unterstützen Web Services die *Komposition*, d.h. verschiedene Dienste können kombiniert werden, um gemeinsam ein Geschäftsproblem zu lösen bzw. um einen komplexen Service zu beschreiben. Letzterem Aspekt wird in Abschnitt 2.4 im Kontext des *Workflow Management* Rechnung getragen.

In Abbildung 2.2 werden die aufeinander aufbauenden und geschichteten Technologien, die mit der Web Services Technologie im Zusammenhang stehen, illustriert (nach Booth et al. (2003)). Wesentlich für Web Services sind dabei XML und Technologien wie XML Schema oder DTD sowie XML-Namespaces. Der so genannte Basistechnologie-Stack umfasst die Protokolle SOAP – früher das *Simple Object Access Protocol*, heute nur SOAP genannt – die *Web Services Description Language* (WSDL) und das *Universal Description, Discovery and Integration* (UDDI) Protokoll. Diese Spezifikationen werden in sehr vielen und guten Übersichtsartikeln vorgestellt (Curbera et al., 2002; Kossmann und Leymann, 2004).

Darauf aufbauend werden weitere Spezifikationen und Standards entwickelt, wie beispielsweise die *Business Process Execution Language* (BPEL4WS) zur Aggregation von Web Services (Andrews et al., 2003).

SOAP – ehemals *Simple Object Access Protocol*

Web Services werden in der Regel durch Nachrichten angesprochen und liefern ihre Ergebnisse auch in Form von Nachrichten zurück. Die WSA schreibt an dieser Stelle nicht vor, welche Kommunikationsmechanismen zu nutzen sind. Somit können Nachrichten prinzipiell jeden Kommunikationsmechanismus nutzen. Vornehmlich wird jedoch HTTP(S) eingesetzt, aber auch andere Internetprotokolle wie SMTP oder FTP können genutzt werden. Um Interoperabilität bei der Kommunikation zu erreichen, wird ein einheitliches Nachrichtenformat benötigt. Die Kodierung der Nachrichten geschieht im wesentlichen durch SOAP (Box et al., 2000). Dieses erlaubt neben dem RPC-Style (*Remote Procedure Calls*) auch asynchrones Messaging. SOAP ist sehr einfach aufgebaut und flexibel erweiterbar. Auf diese Weise können spezielle Anforderungen jederzeit integriert werden. Beispielsweise wurde der Nachrichtendienst des ebXML (*Electronic Business using eXtensible Markup Language*), der so genannte *ebXML Message Service* (ebMS), über den SOAP-Envelope integriert (Manes, 2001).

WSDL – *Web Services Description Language*

Mit WSDL (Christensen et al., 2001) besitzt die WSA ebenfalls ein Werkzeug, um die Schnittstellen der jeweiligen Dienste beschreiben zu können. In WSDL wird die Syntax des Services und seiner Operationen offenbart[5]. Zudem werden die Protokolle beschrieben, über den ein anderer Service diese Operationen ansprechen kann. Dieses beschreibt den technischen Aspekt der Schnittstelle. Über die Semantik der Daten und Operationen wird in WSDL keine Auskunft gegeben.

Die Beschreibung der Schnittstelle über eine standardisierte Sprache ist von entscheidender Bedeutung, um Interoperabilität zwischen heterogenen Serviceanbietern und -konsumenten zu ermöglichen. WSDL erlaubt Dienstanbietern, das Format der Anfragen und Resultate über verschiedene Protokolle und Kodierungen zu beschreiben, d.h. es wird ausgesagt, *was* ein Service ausführen kann, *wo* er liegt und *wie* man ihn anspricht (vgl. Abbildung 2.3 nach Kossmann und Leymann (2004)).

UDDI – *Universal Description, Discovery and Integration*

Des weiteren – jedoch nicht explizit in die WSA aufgenommen – wird UDDI (Oas, 2003) eingesetzt, um benötigte Web Services auffinden zu können. UDDI wird vom UDDI Technical Committee innerhalb von OASIS standardisiert. Ziel der UDDI Initiative ist die

[5]Namen der Operationen und Parameter, sowie verwendete Datentypen.

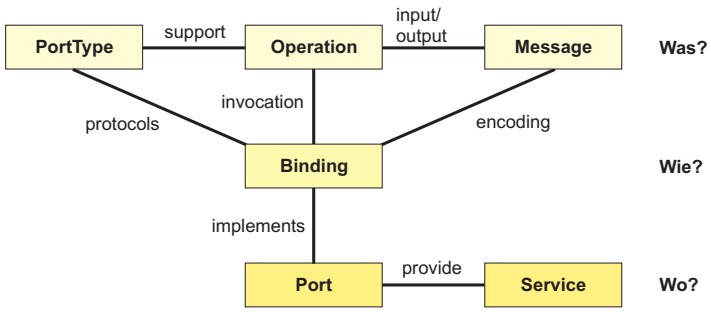

Abbildung 2.3: Elemente der WSDL Spezifikation.

Erstellung eines globalen, plattformunabhängigen Frameworks, mit dem Unternehmen einander finden (*Discovery*), ihre Internetschnittstellen offen legen (*Description*), sowie ihre Prozesse integrieren können (*Integration*). Diese Spezifikation stellt hierzu White Pages, Yellow Pages und Green Pages zur Verfügung.

In den *White Pages* werden die Namen, Beschreibungen und Kontaktinformationen der Dienstanbieter gespeichert. *Yellow Pages* gleichen einem Branchenbuch und erlauben eine Klassifizierung und Kategorisierung der Dienste über (standardisierte) Taxonomien. Hier sind aber auch eigene Kategorisierungen möglich, bzw. gegebene Taxonomien lassen sich den Wünschen der Anbieter entsprechend erweitern. In den *Green Pages* werden darüber hinaus technische Informationen über publizierte Dienste abgelegt. Diese werden in der Regel durch WSDL-Dokumente dargestellt.

UDDI selbst ist wiederum ein Web Service, der über SOAP angesprochen werden kann, und grundsätzlich die Suche nach Namen (des Service, des Anbieters, der Schnittstelle) bzw. Schlüsselwörtern unterstützt. Erweiterte Möglichkeiten wie Inferenzen oder ein flexibles Matchmaking zwischen Schlüsselwörten sind nicht realisiert (Paolucci et al., 2002).

2.3.1 Einführung in WSDL

Die vorliegende Arbeit setzt auf WSDL auf, daher hier eine kurze Einführung. Derzeit ist die Version 1.1 gültiger Standard, es wird aber bereits an der Version 2.0 gearbeitet. Die wesentlichen Konzepte einer WSDL-Schnittstellenbeschreibung in der Version 1.1 sind in Abbildung 2.3 dargelegt. Diese lassen sich prinzipiell in drei Kategorien einteilen: Schnittstelle, Bindung und Services.

Die erste Kategorie beschreibt die Schnittstelle des Service abstrakt. Hier wird spezifiziert, welche ein- und ausgehenden *Nachrichten* (*messages*) eine *Operation* (*operation*) dem Client anbietet. Jedes Nachrichtenelement ist eine Zusammenfassung von mehreren *Teilen* (*parts*), denen je ein Name und ein Typ eines Typsystems zugeordnet ist. Als Typsystem wird häufig *XML Schema Datatypes* (XSD) (Biron und Malhotra, 2001) eingesetzt. Ein

```
<message name="getNucSeqRequest"
xmlns:tns="http://www.ebi.ac.uk/XEMBL">
  <part name="format" type="xsd:string">
    <documentation>
      Input parameter that indicates the result format that should be returned. Legit
      values: Bsml or sciobj. Defaults to Bsml if format not recognised.
    </documentation>
  </part>
  <part name="ids" type="xsd:string">
    <documentation>
      A space delimited list of international Nucleotide Sequence accession numbers
      (IDs). For example: "HSERPG U83300 AC000057". Minimum number of IDs is 1.
    </documentation>
  </part>
</message>

<message name="getNucSeqResponse">
  <part name="result" type="xsd:string">
    <documentation>
      An XML formatted result in either Bsml or AGAVE format.
    </documentation>
  </part>
</message>

<portType name="XEMBLPortType">
  <operation name="getNucSeq">
    <input message="tns:getNucSeqRequest" name="getNucSeqRequest" />
    <output message="tns:getNucSeqResponse" name="getNucSeqResponse" />
  </operation>
</portType>
```

Abbildung 2.4: Port-Typ des XEMBL Web Service.

Port-Typ (*portType*) beschreibt nun eine Menge von Operationen, die kollektiv von einem Service zur Verfügung gestellt werden. Das heißt, der Port-Typ spezifiziert die eigentliche Schnittstelle des Web Service.

Wie eine Operation angesprochen wird, wird in der so genannten *Bindung* (*binding*) festgelegt. Die Bindung spezifiziert die Protokolle, die zum Nachrichtenaustausch benutzt werden, und die Kodierung der Nachrichten. Die Menge der möglichen Bindungsprotokolle wird von WSDL nicht fixiert. Stattdessen ist es möglich jederzeit neue Bindungen zu definieren. Beispielsweise wurden im Rahmen des *Web Services Invocation Frameworks* (WSIF)[6] Bindungen für Java, EJB und JMS definiert.

Der letzte Teil der WSDL-Schnittstellenbeschreibung beinhaltet die konkrete Internetadresse, unter der die abstrakte Schnittstelle implementiert wird und ansprechbar ist.

[6] `ws.apache.org/wsif/`

```
<binding name="XEMBLServiceBinding" type="tns:XEMBLPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="getNucSeq">
    <soap:operation soapAction="http://www.ebi.ac.uk/XEMBL#getNucSeq" />
    <input>
      <soap:body use="encoded" namespace="http://www.ebi.ac.uk/XEMBL"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output>
      <soap:body use="encoded" namespace="http://www.ebi.ac.uk/XEMBL"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
</binding>
```

Abbildung 2.5: Bindung des XEMBL Web Service.

Hierzu wird ein *Port* (*port*) innerhalb der WSDL-Beschreibung als eine Kombination einer Bindung und einer Netzwerkadresse definiert (Curbera et al., 2002). Ein *Dienst* (*service*) vereinigt abschließend eine Menge von konkreten Ports.[7]

> **Beispiel 2.1.** In Abbildung 2.4 ist der Port-Typ des XEMBL Web Service[8] dargestellt. Dieser Port-Typ besitzt lediglich die Operation `getNucSeq`, die eine Input- und eine Outputnachricht bereitstellt. Die Inputnachricht besteht aus zwei Einheiten, dem Teil für die Festlegung des Formates und dem Teil für die verschiedenen Akzession-Nummern.
>
> Die in Abbildung 2.5 dargestellte Bindung des XEMBL Web Service schreibt den SOAP-RPC-Style zur Interaktion mit dem Service vor, bei dem alle Nachrichten die Standard SOAP-Kodierung verwenden. Der konkrete Service `XEMBLService` aus Abbildung 2.6 beinhaltet einen einzelnen Port, der über eine SOAP-Adresse erreichbar ist.

Es fällt auf, dass WSDL-Dokumente lediglich die Syntax der Dienste abbilden können, d.h. es werden die Signaturen der Nachrichten und deren Datentypen spezifiziert. Zusätzlich kann jedes Element durch eine inhaltliche Beschreibung (*documentation*) ausgezeichnet werden. Eine höhere semantische Beschreibung, beispielsweise durch ontologische Konzepte und den Einsatz von Techniken wie dem *Resource Description Framework* (RDF)[9] oder der *Web Ontology Language* (OWL)[10] ist derzeit nicht vorgesehen.

[7] Im Verlauf dieser Arbeit werden die „Parameter“ der Operationen als „Ports“ bezeichnet, da sie es sind, die in einem Workflow verknüpft werden.

[8] `www.ebi.ac.uk/xembl/XEMBL.wsdl` (EBI, 2001)

[9] `www.w3.org/RDF/` bzw. RDF Schema – RDF(S): `www.w3.org/TR/rdf-schema/`

[10] `www.w3.org/2004/OWL/`

```
<service name="XEMBLService">
  <documentation>
    Returns full information on EMBL Nucleotide Sequences formatted as
    Bsml XML or Agave XML. I.e. returns sequence itself, cross-
    references, taxonomy, literature, full feature information, etc.
  </documentation>

  <port name="XEMBLPort" binding="tns:XEMBLServiceBinding">
    <soap:address
      location="http://www.ebi.ac.uk:80/cgi-bin/xembl/XEMBL-SOAP.pl" />
  </port>
</service>
```

Abbildung 2.6: Service-Beschreibung des XEMBL Web Service.

2.3.2 Definition eines Services

Ausgehend von den vorangestellten Betrachtungen wird in der vorliegenden Arbeit unter einem *Web Service* folgendes verstanden:

Definition 2.1 (Service, Nachricht, Port)**.** Ein *Web Service*, kurz *Service*, ist eine virtuelle Komponente mit einer eindeutigen Schnittstelle, die durch die in WSDL beschriebenen Port-Typen spezifiziert wird. Jeder Port-Typ beinhaltet eine endliche Menge von Operationen, die ihrerseits durch einen Operationsnamen, eine Input- und eine Outputnachricht, sowie eine optionale Menge von Fehlernachrichten beschrieben werden. Eine *Nachricht* besitzt neben einem Namen eine Sammlung von Parametern, die im Folgenden *Ports* genannt werden[11], wobei jeder Port durch einen Namen und einen Datentyp definiert wird. Ports der Input- bzw. Outputnachrichten werden *Inputports* bzw. *Outputports* genannt. Allgemein ist es möglich jedem Element eine inhaltliche, unstrukturierte Beschreibung über einem beliebigen Alphabet zuzuordnen. ■

Die in Definition 2.1 eingeführten „Ports“ sind nicht mit dem Tag *port* aus WSDL zu verwechseln. Ein Port in WSDL verknüpft einen Port-Typ mit einer Bindung. Die hier definierten Ports sind die Elemente, über die Services komponiert werden (siehe auch Definition 2.2).

Aus der WSDL-Beschreibung geht klar hervor, dass die in Definition 2.1 beschriebenen Ports nur durch ihre Namen und ihre syntaktischen Typen beschrieben werden. Syntaktische Typen werden durch ein Typsystem definiert. Im Falle der Web Services ist dies im wesentlichen *XML Schema Datatypes* (XSD). Über einem Typsystem lässt sich die *Untertyprelation* definieren. Ein Datentyp d' ist ein Untertyp eines Datentyps d, in Zeichen $d' \preceq d$, gdw. jede Instanz des Typs d' auch eine Instanz des Typs d ist. Die Untertyprelation kann explizit im Typsystem beschrieben werden, wie im Falle von Java durch die

[11] Das *part*-Tag in WSDL.

Schlüsselwörter *implements* und *extends*. Ist diese Relation nicht explizit im Typsystem spezifiziert, werden Mechanismen zur Herleitung benötigt. Sycara et al. (2002) beschreiben hierzu eine Menge von Inferenzregeln, um die Untertyprelation zur Laufzeit bestimmen zu können. Ein derartiges Verfahren ließe sich auch auf XSD übertragen.

2.4 Workflows und Servicekompositionen

Web Services realisieren Geschäftseinheiten, die von einer Organisation in flexibler Art und Weise kombiniert werden können, um ein gegebenes Geschäftsproblem zu lösen. Aufgrund dieser Thematik ist die Servicekomposition mit dem Begriff des *Workflow Management* eng verwandt bzw. baut auf Erfahrungen aus diesem Gebiet auf.

Workflow Management und Workflow Management Systeme

Die *Workflow Management Coalition* (WFMC) ist eine nicht profitorientierte Organisation, die sich der Aufgabe widmet, für die Workflow-Technologie ein gemeinsames Vokabular sowie grundlegende Standards zu etablieren (WFMC, 1999). Sie definiert einen Geschäftsprozess wie folgt:

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

Dabei kann sich der Prozess innerhalb einer einzelnen Organisationseinheit befinden oder über mehrere Organisationen hinweg aufgespannt sein. Ein einzelner logischer Schritt innerhalb eines Prozesses wird gewöhnlich *Aktivität* genannt, der manuell oder automatisch ausgeführt wird. Die teilweise oder vollständige Automatisierung eines Geschäftsprozesses wird *Workflow* genannt (WFMC, 1999):

> *„[A* workflow *is the] automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.“*

Workflows und die den Workflows zugrundeliegenden Geschäftsprozesse werden durch entsprechende *Prozessdefinitionen* spezifiziert. Eine Prozessdefinition besteht insbesondere aus

- einem Netzwerk von Aktivitäten und deren Beziehungen,
- Kriterien, die den Start und die Terminierung des Prozesses beschreiben,
- sowie Informationen über die Daten, die in dem Prozess benötigt werden.

Über eine Prozessdefinition kann ein *Workflow Management System* (WFMS) den Prozess überwachen und modellieren. Die Aufgaben des WFMS umfassen insbesondere

- Die Spezifikation eines Workflows,
- die Ausführung eines Workflows und die Überwachung der Ausführung, sowie
- die Administration und die Analyse eines Workflows

mit Hilfe informationstechnischer Werkzeuge und Anwendungen (WFMC, 1999).

Innerhalb der Prozessdefinition wird der Ablauf der einzelnen Aktivitäten während der Abarbeitung durch *Kontrollstrukturen* spezifiziert. Im Rahmen einer Analyse verschiedener Einsatzszenarien und WFMS haben van der Aalst und ter Hofstede (2002) über 20 derartiger Strukturen identifiziert, wobei einige jedoch eher selten Verwendung finden. Diese Strukturen haben die Autoren in Form von so genannten *Workflow Patterns* im Sinne einer Pattern-Sprache zur Verfügung gestellt (van der Aalst et al., 2003)[12]. Viele dieser Patterns lassen sich durch *Workflow Nets*, einer Anwendung der Petri-Netz Theorie, formal spezifizieren (van der Aalst, 1998). Zu den bekanntesten Pattern zählen

- die parallele Ausführung,
- die sequenzielle Ausführung,
- die synchrone oder asynchrone Zusammenführung (*AND-* bzw. *OR-Join*) und
- die Verzweigung (*Branch*).

Heutige Forschungsarbeiten im Bereich der WFMS beschäftigen sich unter anderem mit dem Aspekt des *Adaptive Workflow Management* (Reichert et al., 2003). Han et al. (1998) haben in diesem Kontext ein Klassifikationsschema aufgestellt, das verschiedene Ebenen der Workflow-Adaption aufspannt. Die Autoren unterscheiden die Ebenen

- **Domäne**: Anpassung des Workflow-Systems an einen veränderten Geschäftskontext.
- **Prozess**: Anpassung an die Evolution des Workflow-Modells und ad-hoc Modifikationen des aktuell ausgeführten Workflows.
- **Ressource**: Anpassung an Veränderungen der organisationsbezogenen Ressourcen, wie personelle Entwicklung, sowie der datenbezogenen Ressourcen, wie veränderte Daten und Datenformate.
- **Infrastruktur**: Anpassung bzw. Modifizierung der Systemkonfiguration an Gegebenheiten, wie veränderte Geschäftsumgebungen oder technische Vorgaben.

Anhand dieser Taxonomie lassen sich einerseits vorhandene Systeme klassifiziern, andererseits spezielle, Ebenen-unabhängige Lösungen realisieren.

[12] `www.workflowpatterns.com`

Workflows und Web Services Kompositionen

Die Weiterentwicklung und Übertragung der WFMS auf das Medium Internet führt konsequenterweise zur Kopplung der Web Services mit dem Gebiet des Workflow Managements. Denn Web Services sind verteilt zugreifbare Komponenten, die in flexibler Weise miteinander komponiert werden können – auch in einer Art und Weise, wie es der Entwickler zur Zeit der Erstellung nicht vorhergesehen hat – um zusammen ein gegebenes Geschäftsproblem zu lösen. Das heißt, Servicekompositionen dienen der Beschreibung von (verteilten) Geschäftsprozessen und liegen gewöhnlich als Workflow vor.

Um die Komposition zu ermöglichen, wurden verschiedene Sprachen zur Prozessdefinition entwickelt, wie beispielsweise die *Web Services for Business Process Design* (XLANG) (Thatte, 2001) und die *Web Services Flow Language* (WSFL) (Leymann, 2001). Diese beiden Sprachen sind kürzlich in der BPEL4WS Spezifikation aufgegangen (Andrews et al., 2003). BPEL4WS ist heutzutage der empfohlene Standard zur Aggregation von Web Services. BPEL4WS fasst in sich block- und graphbasierte Kontrollstrukturen zusammen, wodurch sie zu einer mächtigen und ausdruckstarken Sprache wird. Diese Ausdrucksmächtigkeit wird zum Preis einer hohen Komplexität erreicht (van der Aalst, 2003). Zudem sind nicht alle Kontrollstrukturen exakt spezifiziert, wodurch BPEL4WS Mehrdeutigkeiten aufweist (Wohed et al., 2002).

Die nachfolgenden Betrachtungen eines serviceorientierten Workflows sind unabhängig von der gewählten Prozessdefinitionssprache. Es wird von folgendem *Workflow-Begriff* ausgegangen:

Definition 2.2 (Workflow)**.** Ein *Workflow* besteht aus einer Menge von Services, deren Operationen entsprechend den Anforderungen des zu lösenden Problems verschaltet sind. Dazu spezifiziert der Workflow

- die Reihenfolge, in der die verschiedenen Operationen ausgeführt werden, den so genannten *Kontrollfluss* und
- den Datentransfer zwischen den Operationen, den so genannten *Datenfluss*.

Hinsichtlich des Datenflusses werden immer die Ports der Outputnachrichten O eines Service S mit den Ports der Inputnachrichten I eines Service S' verschaltet. Ferner muss die Outputoperation im Kontrollfluss vor der Inputoperation ausgeführt werden, damit die Daten entsprechend vorliegen. In diesem Zusammenhang bezeichnet S den *datenproduzierenden Dienst* und O die *datenproduzierende Operation* und analog S' den *datenkonsumierenden Dienst* und I die *datenkonsumierende Operation*. ■

> **Beispiel 2.2.** Abbildung 2.7 veranschaulicht einen einfachen Workflow, der aus zwei verbundenen Diensten S_1 und S_2 besteht. In diesem Workflow wird der Output der Search-Operation, hier ein annotiertes Gen, mit dem Input der BLAST-Operation[13] über den Datenfluss F_d verbunden. Über den Kontrollfluss F_c wird fest-

[13] *Basic Local Alignment Search Tool* (BLAST) (Altschul et al., 1990)

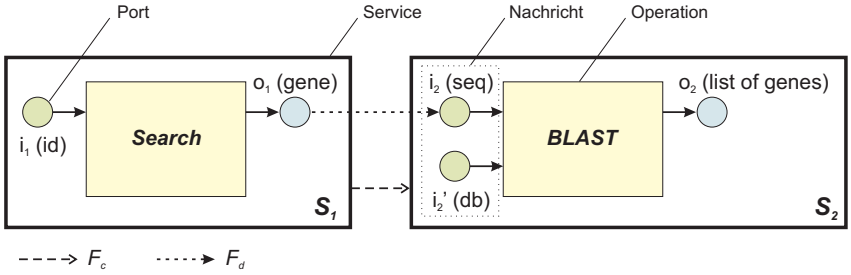

Abbildung 2.7: Darstellung eines Workflows mit zwei verbundenen Diensten.

gelegt, dass die Search-Operation vor der BLAST-Operation aktiviert wird. Dieser Workflow soll das einfache Problem lösen, gegeben den Identifikator einer Akzession im Format der Search-Operation, alle zu dem Gen mit dem Identifikator *id* homologen Gene über eine BLAST-Suche zu bestimmen.

In der Praxis lassen sich Workflows häufig durch visuelle Buildertools der verwendeten Middleware erstellen. In der Bioinformatik zu nennen wären beispielsweise Taverna (Oinn et al., 2004) des myGrid-Projektes (Stevens et al., 2003) oder BioOpera (Bausch et al., 2003). Auch die Hersteller applikationsunabhängiger Middleware-Lösungen bieten derartige Lösungen an, wie beispielsweise Microsoft's BizTalk Server[14] oder Oracle's BPEL Process Manager[15]. Diese sind exemplarisch in Abbildung 2.8 dargestellt.

Heute existierende Middleware-Lösungen unterstützen das Auffinden und Komponieren der Dienste, lösen jedoch nicht oder nur rudimentär das Problem der Heterogenität der Dienste.

2.5 Zusammenfassung

Dieses Kapitel führt in die Konzepte der komponenten- und serviceorientierten Software ein. Anpassbarkeit und Wiederverwendbarkeit werden als essentielle Anforderungen an Softwarekomponenten identifiziert. Hierdurch sollen Kosten sowie Risiken bei der Erstellung neuer Software reduziert werden. Die Serviceorientierung führt die Idee der Komponententechnologie fort, in der Art, dass weitaus loser gekoppelte, über das Internet verteilt zugreifbare Softwareentitäten definiert werden. Eine Kernfunktionalität der Serviceorientierung ist die flexible und dynamische Erstellung von Geschäftsprozessen, die über Workflows realisiert werden.

[14] www.microsoft.com/biztalk/evaluation/overview/biztalkserver.mspx
[15] www.oracle.com/technology/products/ias/bpel/index.html

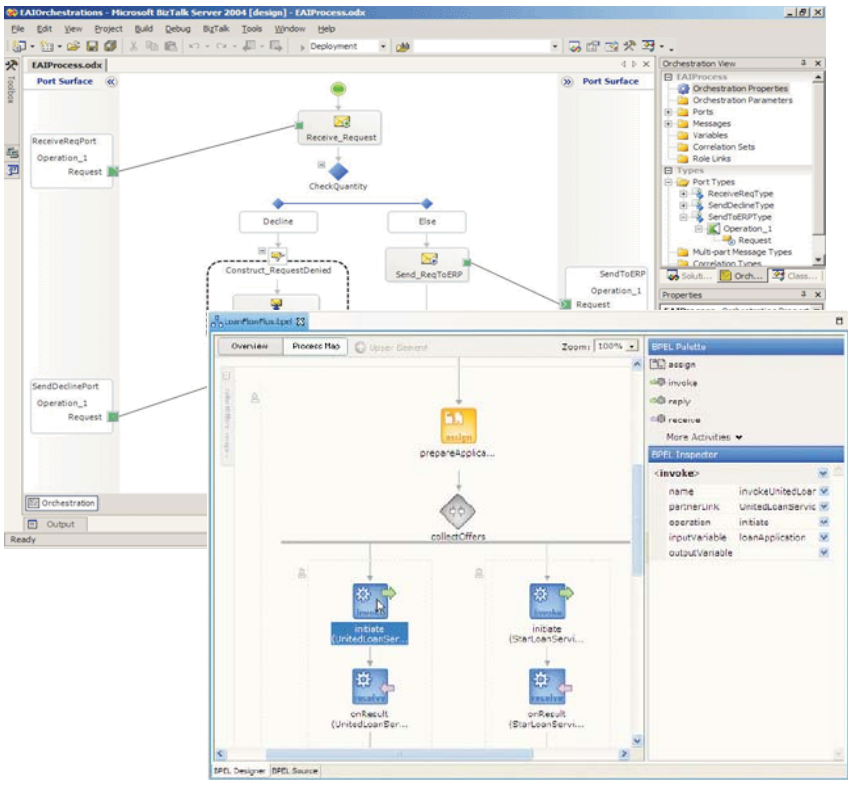

Abbildung 2.8: Beispiel des Oracle BPEL Process Manager (vorne) und des Microsoft BizTalk Server (hinten).

Die Überwindung der technischen, syntaktischen und semantischen Heterogenität innerhalb eines Workflows ist aufgrund der Autonomie der einzelnen Services ein anerkanntes Problem. Durch die Standardisierung der Web Services Technologie wird diesem Problem auf technischer Ebene entgegengewirkt. Syntaktische und semantische Fragestellungen bleiben ein offenes Forschungsfeld von hoher wirtschaftlicher Relevanz.

Kapitel 3

Interoperabilität innerhalb serviceorientierter Software

Die Erzielung der syntaktischen und semantischen Interoperabilität in service- oder komponentenorientierter Software ist ein anerkanntes Problem von großer wirtschaftlicher Bedeutung, das häufig nur manuell und unter hoher Zeit- und Kostenbelastung gelöst werden kann. Es ist umgekehrt bekannt, dass sowohl die Standardisierung von Formaten und Schnittstellen als auch der gezielte Einsatz von (generierten) Softwarebrücken diesem Problem entgegenwirken. Beide Ansätze alleine stoßen jedoch schnell an ihre Grenzen, insbesondere in wissenschaftlichen Anwendungsgebieten, in denen teilweise komplexe Softwarebausteine zur Überbrückung der Heterogenität benötigt werden und häufig keine Standards vorhanden sind. Erschwerend kommt hinzu, dass vorhandene Servicestandards keine Semantik unterstützen. Hier bietet die Semantic Web Initiative mit standardisierten Ontologiebeschreibungssprachen, wie RDF(S) und OWL, erste Ansatzpunkte.

Die in Kapitel 2 eingeführten Services sind lose gekoppelte Softwareeinheiten, die in flexibler Art in Form von Workflows verknüpft werden können. Ein zentrales Problem bei der Verknüpfung ist die Heterogenität der einzelnen Services, die sich in Syntax und Semantik ausdrückt, sowie die damit verbundene Erzielung der Service-Interoperabilität. Häufig wird letztere nur manuell unter hohem Zeit- und Kostenaufwand erreicht. Es kommt erschwerend hinzu, dass die vorhandenen Web Services Standards keine formal spezifizierte Semantik bereitstellen.

Im Rahmen des Semantic Web wird derzeit der Einsatz von so genannten *Ontologien* zur Beschreibung von Semantik diskutiert. Über sie ist es möglich, *Begriffsnetzwerke* formal zu beschreiben. In Abschnitt 3.1 wird das Konzept der Ontologie sowie vorhandene Beschreibungssprachen vorgestellt. Mit Hilfe dieser Technologie wird in dieser Arbeit die Ausdrucksfähigkeit eines Services durch semantische Annotation erweitert. Diese Annotation geschieht über Konzepte einer Domänenontologie.

Basierend auf dieser Erweiterung wird in Abschnitt 3.2 der dieser Arbeit zugrundeliegende Begriff der *Service-Interoperabilität* definiert. Abschließend werden in Abschnitt 3.3 verwandte Arbeiten (*related work*) zum Thema Interoperabilität im Kontext der service- und komponentenorientierten Software diskutiert.

3.1 Ontologien und semantische Annotation

Der Begriff der *Ontologie*, der „Lehre vom Seienden", ist der philosophischen Disziplin entliehen. In der Informatik wird der Begriff von Gruber (1993) als explizite Spezifikation einer Konzeptualisierung eingeführt:

> *„A body of formally represented knowledge is based on a* conceptualization*: the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold them. [...] An* ontology *is an explicit specification of a conceptualization."*

Eine Ontologie verkörpert bezüglich einer gegebenen Domäne eine bestimmte Sicht auf die Welt. Diese Sicht wird gewöhnlich als eine Menge von *Konzepten* (z.B. Entitäten, Attribute, Prozesse) sowie ihrer *Definitionen* und *Beziehungen* konzipiert. Daher wird dies auch *Konzeptualisierung* genannt (Uschold und Gruninger, 1996). Derzeit existiert in der Informatik keine allgemein anerkannte Definition der Ontologie (Uschold und Gruninger, 1996). So fordern Maedche und Staab (2001), dass eine Ontologie ein eingegrenztes Vokabular von Konzepten bereitstellt, wobei jedes Konzept eine explizit definierte und *maschinenverarbeitbare* Semantik besitzt.

Narayanan und McIlraith (2002) verstehen unter einer Ontologie eine formale und explizite Beschreibung von Konzepten einer Domäne, die auch *Klassen* genannt werden. Die Eigenschaften eines Konzepts beschreiben die verschiedenen Fähigkeiten und Attribute des Konzepts. Daher werden sie auch *Slots*, *Rollen* oder *Properties* genannt. Restriktionen auf Slots bezeichnet man als *Facets* oder *Rollenrestriktionen*. Eine Ontologie mit einer Menge von individuellen Instanzen der Klassen stellt eine *Wissensbasis* dar. Der Übergang von einer Ontologie zu einer Wissensbasis ist allerdings fließend. Eine Ontologie wird im Folgenden als explizite und formale Spezifikation einer Konzeptionalisierung verstanden:

Definition 3.1 (Ontologie)**.** Eine *Ontologie* beinhaltet eine endliche Menge von *Konzepten* und *Instanzen*, sowie eine Menge von benannten *Beziehungen* (*Eigenschaften*) zwischen Konzepten untereinander und zwischen Konzepten und Datentypen. Durch diese Beziehungen wird ein Konzept definiert. Jedem Konzept wird ein eindeutiger Name zugeordnet[1]. Instanzen werden Konzepten zugeordnet. Daraus folgt, dass ein Konzept eine *Klasse* von *Objekten/Individuen* des betrachteten Anwendungsgebietes definiert. ■

[1]Gewöhnlich hat der Name repräsentativen Charakter, das heißt er ist ein lesbarer Ausdruck, der das Konzept in der „realen" Welt beschreibt.

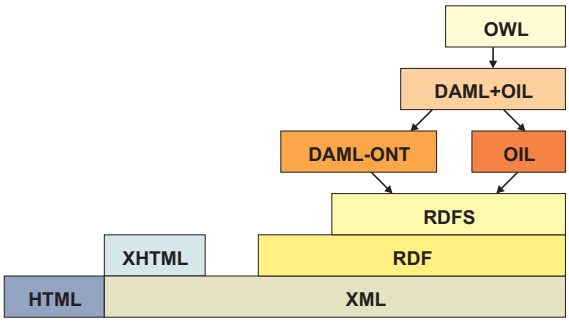

Abbildung 3.1: Ebenen der verschiedenen Beschreibungssprachen.

Konzepte entsprechen den Mengen von Dingen, die in der betrachteten Anwendungsdomäne natürlich vorkommen. Instanzen korrespondieren hingegen zu den aktuellen Entitäten, die zu Konzepten gruppiert werden können (W3C, 2004a).

Sei $\mathcal{C}$ eine endliche Menge von Konzepten einer Ontologie. Zu den bekanntesten Beziehungen zwischen Konzepten gehören *part-of* Beziehungen und *is-a* Beziehungen (*Subsumptionsrelation*). Die Subsumptionsrelation drückt aus, dass ein Konzept $c \in \mathcal{C}$ (*subsumer*) als genereller anzusehen ist, als ein Konzept $c' \in \mathcal{C}$ (*subsumee*), in Zeichen $c' \sqsubseteq c$ (Nardi und Brachman, 2004). Es gilt, dass $c' \sqsubseteq c$ gültig ist gdw. jede Instanz des Konzepts c' auch eine Instanz des Konzepts c ist. Über explizit angegebene Subsumptionsrelationen lassen sich durch Inferenz auch implizite Subsumptionsrelationen identifizieren. Diese Bestimmung gehört zu den Basisinferenzmechanismen einer Beschreibungslogik (Nardi und Brachman, 2004). Eine wesentliche Eigenschaft der *is-a* Relation ist ihre Transitivität.

Beispiel 3.1. Ontologien werden gewöhnlich für konkrete Anwendungsgebiete oder Ausschnitte eines Anwendungsgebietes definiert, so genannte *Domänenontologien*. Die *Gene Ontology*[2] ist eine derartige Domänenontologie für die Biowissenschaften. Neben Ontologien, die nur für eine bestimmte Domäne ausgelegt sind, existieren einige universelle Ontologien, wie die UNSPSC Ontologie[3] oder die WordNet Ontologie[4].

3.1.1 Beschreibungssprachen für Ontologien

Um eine Ontologie formal spezifizieren zu können, so dass sie maschinenlesbar wird, werden möglichst standardisierte Beschreibungssprachen benötigt. Vor allem im Kontext des Semantic Web sind hier Sprachen zur Beschreibung von Metadaten und Ontologien entstanden.

[2] `www.geneontology.org`
[3] `www.unspsc.org`
[4] `wordnet.princeton.edu`

RDF – *Resource Description Framework*

Die genaue Entstehungsgeschichte der verschiedenen web-basierten Sprachen zur Beschreibung von Ontologien begann mit der Auszeichnung von Web-Seiten durch Metadaten, die in RDF beschrieben werden. RDF, das *Resource Description Framework* (Lassila und Swick, 1999), setzt auf XML auf und ist ein erster Schritt in Richtung der Beschreibung von Ontologien (siehe Abbildung 3.1). RDF definiert eine syntaktische Konvention und ein einfaches Datenmodell zur Repräsentation maschinenverarbeitbarer Semantik von Daten. Hierzu nutzt das *RDF Description Model* Objekt-Attribut-Wert Tripel. Jede Instanz dieses Modells lässt sich als gerichteter oder benannter Graph interpretieren, der einem semantischen Netzwerk gleicht (Lassila, 2000).

RDFS – *RDF Schema*

Ein Defizit von RDF ist, dass weder Mechanismen zur Beschreibung der Attribute angeboten werden, noch Mechanismen zur Beschreibung von Beziehungen zwischen den Attributen und anderen Ressourcen existieren (Brickley und Guha, 2004). Daher wurde auf Basis der RDF-Spezifikation die Spezifikation des *RDF Schemas* (RDFS) (Brickley und Guha, 2004) entwickelt.

RDFS spezifiziert kein domänenspezifisches Vokabluar, sondern bietet Möglichkeiten, Klassen und Beziehungen einer Domäne zu beschreiben. Daher lässt sich RDFS – als Typsystem für RDF – mit einem Typsystem objektorientierter Sprachen wie Java vergleichen (Brickley und Guha, 2004). Ganz im Sinne der Vererbung erlaubt RDFS zudem, Klassen in Hierarchien anzuordnen. Somit definiert RDFS erste Basisprimitive zur Modellierung von Ontologien. Zu den Basisprimitiven gehören unter anderem die Konzepte der Klassen, Beziehungen und Datentypen:

- Eine *Klasse* (`rdfs:class`) in RDFS stellt ein generisches Konzept für Typen oder Kategorien bereit und ist mit Klassen in objektorientierten Sprachen verwandt.
- RDFS beinhaltet eine vordefinierte *is-a* Beziehung (`rdfs:subClassOf`) zur Bildung von Klassenhierarchien.
- Zur Charakterisierung der Klassen in RDFS werden die in RDF spezifizierten *Eigenschaften* (*properties*) eingesetzt (`rdf:Property`). Der Zusammenhang zwischen Eigenschaften und Klassen wird durch die Konzepte *Domäne* (`rdfs:domain`) und *Bereich* (`rdfs:range`) spezifiziert.
- RDFS ermöglicht den Einsatz von extern definierten Datentypen, deren Einsatz explizit angegeben werden kann (`rdfs:Datatype`). RDFS erlaubt hingegen nicht die Definition eines neuen Datentyps.

```
<rdf:RDF ...>
 <rdfs:Class rdf:ID="Person"/>

 <rdfs:Class rdf:ID="University"/>

 <rdfs:Class rdf:ID="Student">
  <rdfs:subClassOf rdf:resource="#Person"/>
 </rdfs:Class>

 <rdfs:Datatype rdf:about="&xsd;string"/>

 <rdf:Property rdf:ID="matriculatedIn">
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="#University"/>
 </rdf:Property>

 <rdf:Property rdf:ID="hasName">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="&xsd;string"/>
 </rdf:Property>
</rdf:RDF>
```

Abbildung 3.2: Beispiel eines RDF(S)-Dokuments.

Listing 3.2 illustriert ein einfaches RDF-Dokument, welches RDFS-spezifizierte Konzepte nutzt.

Im Vergleich zu Typsystemen objektorientierter Sprachen weist RDFS einige Unterschiede auf. Während objektorientierte Sprachen Klassen im Hinblick auf Eigenschaften, die ihre Instanzen haben können, definieren, werden in RDFS Eigenschaften im Sinne von Klassen von Ressourcen definiert, auf welche sie angewendet werden können (Brickley und Guha, 2004):

> **Beispiel 3.2.** Ein *Autor*-Attribut lässt sich beispielsweise mit der Domäne *Buch* und dem Bereich *Person* spezifizieren (Objekt-Prädikat-Subjekt), während im klassischen objektorientierten Design eine Klasse *Buch* entworfen würde, mit dem Attribut *Autor*, das Werte vom Typ *Person* zulässt. Es handelt sich um einen Attribut-zentrischen bzw. Klassen-zentrischen Ansatz.

Im Falle des objektorientierten Designs ist das Attribut *Autor* Bestandteil der Klasse *Buch* und wird damit nur auf Instanzen der Klasse *Buch* angewendet (*class scope*). In RDF besitzen Eigenschaften globale Sichtbarkeit (*global scope*) und können, wenn die Domäne nicht angegeben wird, von verschiedenen Klassen genutzt werden. Durch diese Freiheit können umgekehrt Situationen entstehen, in denen eine Eigenschaft falsch angewendet wird. Eine weitere Konsequenz der globalen Sichtbarkeit einer Eigenschaft ist, dass es in RDFS nicht möglich ist Eigenschaften zu spezifizieren, die je nach Domänkontext (`rdfs:domain`) verschiedene Bereiche (`rdfs:range`) erlauben. Diese Fähigkeit wird in höheren Ontologiebeschreibungssprachen zur Verfügung gestellt.

OIL, DAML-ONT und DAML+OIL

Den Bestrebungen des RDFS folgte die *Ontology Inference Layer* (OIL), eine vollständige Modellierungssprache zur Beschreibung von Ontologien. Parallel dazu wurde in einem von

der *Defense Advanced Research Projects Agency* (DARPA)[5] geförderten Programm die *DARPA Agent Markup Language* (DAML)[6] für Ontologien (DAML-ONT) entwickelt. Diese Sprachen sollten die nächste Evolution des Webs hin zum *Semantic Web* vollziehen (Hendler und McGuinness, 2000). Um die Vorteile der beiden Sprachen DAML-ONT und OIL auszunutzen, wurden die beiden Projekte bei der DARPA unter DAML+OIL zusammengefasst[7] und mündeten schließlich in OWL.

OWL – *Web Ontology Language*

Die beim W3C verfügbare *Web Ontology Language* (OWL) ist heute der neue Internetstandard zur Beschreibung von Ontologien. Sie ist aus der Integration von DAML+OIL hervorgegangen und profitiert von den dort erzielten Erfahrungen. OWL besteht aus drei Untersprachen, die unterschiedliche Ausdrucksstärken aufweisen: OWL Lite, OWL DL und OWL Full (W3C, 2004b):

- *OWL Lite* ermöglicht primär die Beschreibung von Klassifikationshierarchien mit einfachen Bedingungen. Beispielsweise kann die Kardinalität einer Eigenschaft (property) lediglich auf 0 oder 1 gesetzt werden. OWL Lite ist gut geeignet, um Taxonomien oder Thesauri auszudrücken.

- *OWL DL* bietet die maximale Ausdrucksstärke, wobei die Vollständigkeit der Berechenbarkeit (alle Schlussfolgerungen sind berechenbar) und die Entscheidbarkeit (alle Berechnungen enden in endlicher Zeit) einer definierten Ontologie erhalten bleiben. OWL DL beinhaltet alle möglichen Sprachkonstrukte von OWL, die jedoch nur mit bestimmten Restriktionen eingesetzt werden dürfen. Der Term DL suggeriert den Zusammenhang mit Beschreibungslogiken (*description logics*) (W3C, 2004a).

- Im Gegensatz zu OWL DL sind bei *OWL Full* an Benutzung der OWL Sprachkonstrukte keine Restriktionen gebunden. Beispielsweise ist es möglich eine Klasse sowohl als Kollektion von Individuen als auch selbst als Individuum zu behandeln. Auf diese Weise kann OWL Full selbst den Sprachumfang von OWL modifizieren (*Meta-Modellierung*). Es lassen sich bezüglich der Berechenbarkeit keine Garantien abgeben.

Auch in dieser Arbeit bildet OWL die Grundlage zur Beschreibung der Domänenontologien, sowie der Beschreibung der Service-Mediatoren (vgl. Kapitel 4). Der genutzte Sprachumfang überschreitet nicht die Fähigkeiten von OWL DL.

[5] `www.darpa.mil/`

[6] `www.daml.org/`

[7] *Reference description of the DAML+OIL (March 2001) ontology markup language*: `www.daml.org/2001/03/daml+oil-index`.

3.1.2 Semantische Annotation von Services

Eine zentrale Idee der Arbeit ist, die rein syntaktisch beschriebenen Ports durch semantische Annotationen (semi-) automatisch anzureichern. Die Annotation geschieht dabei über Konzepte einer Domänenontologie. Ein wesentlicher Vorteil dieser Trennung wäre, dass die semantischen Typen, d.h. die beschriebenen Konzepte, sehr einfach sein können, während gleichzeitig die syntaktischen Typen sehr komplex sein können. Vergleiche von Nachrichten können somit auf den „einfacher“ beschriebenen semantischen Typen beruhen. Daher wird der Begriff des *Ports* aus Definition 2.1 wie folgt erweitert:

Definition 3.2 (Erweiterter Port)**.** Sei $\mathcal{D}$ die Menge der Datentypen, die durch ein Typsystem beschrieben werden, und sei $\mathcal{C}$ eine endliche Menge von Konzepten, die durch eine Ontologie definiert werden. Dann ist ein (*erweiterter*) *Port* $p = (\omega_p, d_p, C_p, desc_p)$ wie folgt definiert:

ω_p ist der Portname,
$d_p \in \mathcal{D}$ ist der Datentyp des Ports,
$C_p \subseteq \mathcal{C}$ ist eine endliche Menge von Konzepten,
$desc_p$ ist eine inhaltliche, unstrukturierte Beschreibung.

Sowohl der Name als auch die Beschreibung werden durch Worte über einem beliebigen Alphabet gebildet. d_p beschreibt den syntaktischen Typ, während Elemente der Menge C_p den semantischen Typ des Ports spezifizieren. ■

Durch C_p wird eine Disjunktion von Konzepten beschrieben, die einem Port zugeordnet werden kann. Denn ausgehend von einer gegebenen Domänenontologie kann es vorkommen, dass sich ein Port nicht genau einem einzelnen Konzept zuordnen lässt, sondern verschiedenen Konzepten angehört.

> **Beispiel 3.3.** Eine BLAST-Operation könnte RNA- und Proteinsequenzen verarbeiten, nicht jedoch allgemeine Nucleotidsequenzen. C_p würde somit die Konzepte RNA- und Proteinsequenz beinhalten. Eine einzelne Annotierung über das übergeordnete Konzept Sequenz wäre in diesem Fall jedoch ungenügend.

Der erweiterte Port ermöglicht eine syntaktische und semantische Auszeichnung. Da die derzeit vorliegende WSDL-Spezifikation *keine* explizite Auszeichnung von semantischen Typen erlaubt, d.h. es gilt $C_p = \emptyset$, kann die semantische Annotation nicht innerhalb der WSDL-Schnittstellenbeschreibung des Port-Typ vorgenommen werden. Die vorliegende Arbeit löst dieses Problem durch die Entwicklung einer verbesserten Beschreibungssprache (vgl. Kapitel 4.4) und einem semi-automatischen Mapping von Ports zu Konzepten einer Ontologie (vgl. Kapitel 5.2).

3.2 Service-Interoperabilität

Mit der Einführung des erweiterten Ports lässt sich nun der für diese Arbeit relevante Begriff der Service-Interoperabilität beschreiben:

Definition 3.3 (Service-Interoperabilität)**.** Sei W ein Workflow und seien S und S' datenproduzierender bzw. datenkonsumierender Service in W. Ferner sei die Outputnachricht o einer Operation O aus S mit der Inputnachricht i einer Operation I aus S' verknüpft.

Dann sind S und S' hinsichtlich O und I *interoperabel*, wenn eine Zuordnung $\delta : o \rightarrow i$ existiert, so dass für alle nichtoptionalen (erweiterten) Ports p_i aus i ein (erweiterter) Port p_o aus o existiert, und es gilt

(1) $d_{p_o} \preceq d_{p_i}$ (*syntaktische Interoperabilität*),
(2) $\exists c_{p_o} \in C_{p_o}, c_{p_i} \in C_{p_i} : c_{p_o} \sqsubseteq c_{p_i}$ (*semantische Interoperabilität*).

Hierbei bezeichnen $d.$ und $c.$ den syntaktischen Typ bzw. einen möglichen semantischen Typ eines Ports. ■

Aus der obigen Definition geht hervor, dass alle nichtoptionalen Ports der datenkonsumierenden Operation durch Daten der datenproduzierenden Operation entsprechend der Datentypen korrekt gefüllt werden müssen (1). Ferner müssen die semantischen Konzepte der Ports einander entsprechen. Wie aber bereits im Zusammenhang mit Definition 3.2 erläutert, beschreibt die Menge C_p eines erweiterten Ports lediglich eine Disjunktion von Konzepten. Aus diesem Grund fordert Definition 3.3 (2) keine exakte Übereinstimmung der Konzepte, sondern lediglich, dass es auf jeder Seite *mindestens* ein Konzept geben muss, so dass die Subsumptionsrelation gültig ist. Auf diese Weise soll eine minimale Gewähr erzielt werden, dass die Daten auch semantisch korrekt weiterverarbeitet werden können. Zusammenfassend fordert Definition 3.3 semantische und syntaktische *Korrektheit* beim Datentransfer.

Definition 3.3 lässt sich auch auf Mengen von datenproduzierenden Services und einen einzelnen datenkonsumierenden Service erweitern. Hierzu wird lediglich ein neuer datenproduzierender Service angenommen, dessen datenproduzierende Operation alle datenproduzierenden Outputnachrichten der vorhandenen Services in sich vereint. Anschließend wird Definition 3.3 auf den so neu entstandenen datenproduzierenden Service bzw. dessen datenproduzierende Operation und den datenkonsumierenden Service angewendet. Aufgrund dieser Verallgemeinerung wird im Folgenden daher nicht mehr explizit zwischen einer Menge von datenproduzierenden Services und einem einzelnen datenproduzierenden Service unterschieden, es sei denn der konkrete Sachverhalt bedingt dies.

Definition 3.4 (Interoperabilität eines Workflows)**.** Ein Workflow W ist *interoperabel* gdw. alle in ihm kombinierten Services interoperabel sind. ■

Aus Definition 3.4 lässt sich nicht zwingend die *semantische Korrektheit der Ausführung* des Workflows ableiten, d.h. es ist nicht gesichert, dass der Workflow tatsächlich die intendierte Fragestellung des Benutzers, sprich den Geschäftsprozess, löst.

Die autonome und heterogene Landschaft der Web Services zeigt jedoch schnell, dass eine derartige Service-Interoperabilität und die damit verbundene Korrektheit nur im Falle weniger Standards und nicht im Allgemeinen gegeben ist. Das heißt, die direkte Zuordnung δ existiert *a priori* nicht oder nur in seltenen Fällen. Wenn eine Zuordnung δ existiert, dann wird man in der Praxis eher selten den Fall antreffen, dass δ beide Bedingungen gleichzeitig erfüllt. Auch die WSA sagt zur Service-Interoperabilität lediglich, dass der Requester und der Provider sich über die Semantik und die Mechanismen des Nachrichtenaustauschs einigen sollen, bevor ein Nachrichtenaustausch erfolgreich stattfinden kann (Booth et al., 2003):

> *„Often this agreement will be accomplished by the provider entity publicizing and offering both the semantics and the service description as take-it-or-leave-it "contracts" that the requester entity must accept unmodified as conditions of use.“*

Take-it-or-leave-it Verträge sind in der Regel problematisch, wenn Requester und Provider unabhängig voneinander entwickelt wurden und die eigentliche Servicekomposition durch eine dritte Organisation geschieht, da diese gewöhnlich weder auf den Requester noch auf den Provider Einfluss hat. Dies führt in der Praxis häufig dazu, dass die Konzepte aber nicht die Datentypen zusammenpassen oder dass die Syntax aber nicht die verwendete Semantik übereinstimmen. Diese Probleme entstehen, wenn beispielsweise verschiedene Datenformate eingesetzt, verschiedene Skalen verwendet oder Begriffe nicht eindeutig definiert werden.

> **Beispiel 3.4.** Die drei großen Sequenzbanken (NCBI's GenBank, EMBL und DDBJ) verwenden zur Beschreibung der genomischen Information verschiedene Datenformate. NCBI beschreibt die Gen-Information im GenBank *flat file* Format[8], während das XEMBL Projekt[9] BSML[10] und AGAVE[11] nutzt. Andererseits entsprechen die Daten dem gleichen semantischen Konzept, der annotierten Sequenzinformation (*Strukturproblem*).
>
> Längenangaben lassen sich durch den Typ *double* repräsentieren. Dieser Typ sagt jedoch nichts darüber aus, in welcher Einheit die Längenabgabe kodiert ist (*Skalenproblem*). Nukleinsäuresequenzen lassen sich in der Bioinformatik durch Zeichenketten repräsentieren. Aus dem Konzept Nukleinsäuresequenz wird jedoch nicht klar, ob diese Sequenz kodierend ist oder nicht (*Unterspezifizierungsproblem*).

[8] Kürzlich ist auch eine XML Repräsentation des GenBank Formates erschienen.
[9] `www.ebi.ac.uk/xembl/`
[10] Bioinformatic Sequence Markup Language: `www.bsml.org/`
[11] Architecture for Genomic Annotation, Visualization and Exchange: `www.agavexml.org/`

Die obigen Ausführungen machen deutlich, dass die verschiedenen Stufen der Heterogenität bei Diensten weiterhin eine wesentliche Herausforderung bei der Erstellung neuer Servicekompositionen darstellen. Auf technischer Ebene scheint durch die Standardisierung der Web Services Technologie eine Homogenisierung erreicht worden zu sein, die Interoperabilität von Diensten ermöglicht. Vielmehr lässt sich sogar sagen, dass Web Services ganz im Zeichen der Interoperabilität stehen. Auf den Ebenen der Syntax und Semantik sind die Probleme, die sich durch die Heterogenität ergeben, noch weitestgehend ungelöst. Vor allem in naturwissenschaftlichen Bereichen, wie den Bio- oder Geowissenschaften, die durch hohe Dynamik und vielfältige Analysemethoden und Datenformate charakterisiert sind, ist die Entwicklung neuer Workflows zeitaufwändig und fehleranfällig, und es bedarf einiges an manueller Arbeit und informationstechnischem Wissen bis lauffähige Servicekompositionen entstehen.

3.3 Verwandte Lösungsansätze

Um auf Datenebene die Interoperabilität der Services innerhalb eines Workflows zu erreichen, wurden verschiedene Ansätze diskutiert, die sich prinzipiell in zwei Kategorien einteilen lassen: die Standardisierung als *a priori* Ansatz zur Service-Interoperabilität, die unabhängig vom konkreten Workflow ist, und die (automatische) Mediation als softwaretechnischen a posteriori Ansatz, die vom konkreten Workflow ausgeht und entsprechend Verbindungsbausteine (z.B. Glue-Code) zwischen den Services einsetzt. Im Folgenden werden verwandte Ansätze zu diesen beiden Varianten vorgestellt.

3.3.1 Standardisierung als *a priori* Ansatz

Standardisierung findet sowohl in verschiedenen Anwendungsgebieten als auch domänenübergreifend statt. Auf technischer Ebene haben wir bereits verschiedene Gremien und Standards kennengelernt. Hier zu nennen sind beispielsweise das W3C und OASIS, die die Standardisierung der XML-Familie und insbesondere der Web Services Technologie vorantreiben. An gleicher Stelle wäre weiterhin die OMG zu nennen, die CORBA, MDA und UML spezifiziert. Diese Standards sind horizontal bezüglich der Anwendungsgebiete einzuordnen, da sie applikationsunabhängig eingesetzt werden können. Vertikal dazu entstehen in den verschiedenen Anwendungsdomänen Standards, die die Schnittstellen von Diensten, die eingesetzten Konzepte oder die verwendeten Datentypen beschreiben.

Weit vorangeschritten sind beispielsweise die Arbeiten des *Open GIS Consortium* (OGC) und der *International Organization for Standardization* (ISO). Das OGC besteht aus einer Vielzahl von Firmen, Regierungsorganisationen, universitären Einrichtungen und Forschungslaboratorien, die in den Geowissenschaften Schnittstellen für räumliche Dienste sowie ein standardisiertes Datenformat für diese Dienste, die so genannte Geography Markup Language (GML) (Ope, 2001), spezifizieren. Die verschiedenen Dienste werden unter

den *OGC Web Services* (OWS) zusammengefasst (Harrison und Reichardt, 2001). Im Zuge dieser Homogenisierung soll das Problem der Interoperabilität im Bereich der Geowissenschaften gelöst werden.

Die schnelle Verbreitung der vom OGC-spezifizierten, informationstechnologischen Standards zeigt, dass deren Spezifikation gerade zur Behandlung der syntaktischen Heterogenität ein im Geo-Bereich erfolgversprechender Ansatz ist. Durch den Einsatz von standardisierten Diensten wird eine Uniformität erreicht, die es Klienten erlaubt, verschiedene konkurrierende Anbieter eines speziellen Dienstes gegeneinander auszutauschen. Der Anwender erhält hier die Freiheit, nach Eignung, Preis, etc. zu wählen, ohne dabei eine direkte Bindung an Datenformate, Datenlieferanten oder spezielle Systemanbieter einzugehen.

Im dynamischen Umfeld wissenschaftlicher Forschungsvorhaben ist die Realisierung eines einzelnen Standards allein allerdings nicht ausreichend, da standardisierte Dienste bei weitem nicht alle Anforderungen der wissenschaftlichen Forschung abdecken können. Standards können prinzipiell nur allgemein anerkannte Aspekte, nicht jedoch spezielle Anforderungen von in wissenschaftlichen Projekten entwickelten und verwendeten, oft hochgradig individuellen Anwendungen abbilden. Ferner ist Standardisierung ein langwieriger und kostenintensiver Prozess, der aktuellen Bedürfnissen einer Anwendungsdomäne häufig hinterherläuft. Bernard et al. (2003) konnten im Rahmen geowissenschaftlicher Fragestellungen zudem zeigen, dass selbst die weitverbreiteten OWS nicht das Problem der semantischen Heterogenität vollständig adressieren, sondern eher syntaktische Aspekte behandeln.

In der Bioinformatik hat sich – im Gegensatz zur Geoinformatik – kein einheitlicher Standard zur Beschreibung von Diensten durchgesetzt. Hier existiert eine Vielzahl unterschiedlicher Beschreibungsformate, wie beispielsweise AGAVE, GAME[12] oder BSML, die von verschiedenen Anbietern genutzt werden[13].

Das Vorhandensein vieler (de-facto) Standards, die ähnliche Dinge unterschiedlich abbilden, zeigt schnell, dass ein Interoperabilitätsproblem auf Ebene der Standards entsteht. Hier bestände prinzipiell die Forderung nach nur sehr wenigen, wohl abgegrenzten Standards. Dies lässt sich aber aufgrund der verschiedenen Forschungs- und Projektfragestellungen nur schwerlich rückwirkend erreichen. Daher müssen Verfahren entwickelt werden, die einerseits die bestehenden Standards nutzen, und andererseits die vorhandene Heterogenität überbrücken. Hier setzen die folgenden Verfahren an.

3.3.2 Softwaretechnische *a posteriori* Ansätze

In der Softwaretechnologie wurden Methoden zur Erreichung der Interoperabilität entwickelt, die von einem konkreten Workflow (bzw. einer konkreten Komposition) und dessen Services (bzw. deren Komponenten) ausgehen. Diese könnte man als *a posteriori* Verfahren

[12] Genome Annotation Markup Elements
[13] Weitere XML-Sprachen finden sich unter `xml.coverpages.org/game.html`

– im Gegensatz zur Standardisierung als *a priori* Ansatz – verstehen, obwohl auch hier teilweise Standardisierung von Schnittstellen eingesetzt wird.

Adapter-Entwurfsmuster

Eine der wesentlichen Techniken, die in der Softwaretechnologie zum Einsatz kommt, ist die des Adapters. Ein *Adapter* konvertiert die Schnittstelle einer Klasse in die Schnittstelle, die vom Klienten erwartet wird (Gamma et al., 1995). Adapter stellen in der objektorientierten und komponentenbasierten Softwaretechnologie ein weitverbreitetes und häufig genutztes Konzept dar, denn sie erlauben, vorhandene Softwarebausteine mit inkompatiblen Schnittstellen gemeinsam wiederzuverwenden und zu kombinieren. Häufig wird ein Softwareadapter durch Vererbung realisiert (Gamma et al., 1995). Eine wesentliche Konsequenz dieses Vorgehens ist, dass diese Adapter zur Laufzeit fixiert sind und nicht mit anderen Klienten, d.h. Klienten mit anderen Zielschnittstellen, wiederverwendet werden können. Hier muss ein neuer Adapter programmiert werden. Es ist aber möglich, den Adapter zu nutzen, wenn die Schnittstelle des Klienten unverändert bleibt und nur ein neuer Adaptee verwendet werden soll. Der Wechsel auf einen neuen Adaptee lässt sich unter Performanzeinbußen auch dynamisch, d.h. zur Laufzeit, durchführen (Seiter et al., 1999). Somit unterstützt dieses Muster die Flexibilität und Autonomie der Adaptees, nicht jedoch die des Klienten.

Java Connector Architecture (JCA)

Bei der *Java Connector Architecture* (JCA) verschwimmt die Grenze zwischen Standardisierung und softwaretechnischem Ansatz (Sun, 2003). JCA baut ebenfalls auf Softwareadaptern auf, den so genannten *Ressource-Adaptern*. Eine Ressource kann hierbei jedes beliebige *Enterprise Information System* (EIS) sein. Diese reichen von *Legacy Datenbanksystemen* bis hin zu komplexen Systemen zur *Enterprise Resource Planning* (ERP). Das Ziel dieser Architektur ist (J2EE-) Applikationsserver unterschiedlicher Anbieter mit beliebigen auch heterogenen EIS kombinieren zu können und dabei das $n \times m$-Problem (n Applikationsserver, m EIS) durch Standardisierung der Schnittstellen des Ressource-Adapters auf ein $n + m$-Problem zu reduzieren. Jeder J2EE-konforme Applikationsserver muss die standardisierte Schnittstelle zum Einklinken (*Plug-in*) beliebiger EIS über einen Ressource-Adapter unterstützen und jeder EIS-Anbieter muss genau einen Ressource-Adapter für sein EIS zur Verfügung stellen. Die JCA erlaubt dann eine bidirektionale Verbindung zwischen dem Applikationsserver und dem angeschlossenen EIS.

Da Sun die Hoheit über die J2EE-Spezifikation hat und somit auch die JCA spezifizieren und als Standard für J2EE-konforme Applikationsserver vorgeben kann und es sich bei JCA um eine reine Java-Lösung handelt, könnte sich dieser Ansatz für Java-Plattformen durchsetzen. Ferner legt die JCA einen technischen Standard zur *Connectivity* von Ap-

plikationsservern und EIS vor. Über Syntax und Semantik der angeschlossenen EIS wird keine Aussage in der JCA getroffen.

Erweiterung des Portable Object Adapter

Einen zu der JCA verwandten Ansatz hat Shumilov (2003) mit der Realisierung eines erweiterbaren Adapters (eXtensible Database Adapter – XDA) entwickelt. Im Kontext geowissenschaftlicher Modellierungsaufgaben wurde ein Mediatoransatz auf Basis von CORBA realisiert, mit dessen Hilfe sich verteilt vorliegende, objektorientierte Datenbanken (im Speziellen ObjectStore) durch CORBA Mediatoren zugreifen lassen (Shumilov et al., 2002). Hierzu wurden auf Basis des *Portable Object Adapter* (POA) von CORBA entsprechende Zugriffsmöglichkeiten im XDA implementiert.

Auch hier legt die Spezifikation des XDA keine Syntax und Semantik der auszutauschenden Daten vor, d.h. XDA ist ein domänenunspezifisches Framework. Diese Informationen lassen sich aber durch auf XDA aufbauende domänenspezifische Adapter kodieren. Beispielsweise wurde so ein GeoToolKit/CORBA Adapter (GTA) auf Basis des XDA realisiert (Shumilov et al., 2002). GTA repräsentiert die GeoToolKit-spezifischen Klassen, die für eine verteilte Interaktion mit den persistenten GeoToolKit-Objekten über die CORBA Umgebung benötigt werden. Durch GTA wird ein verteiltes 3D/4D GIS auf Basis von GeoToolKit und CORBA realisiert. GTA ermöglicht existierende, jedoch hochgradig heterogene und hochspezialisierte Geo-Anwendungen mit verteilten GIS zu kombinieren und gleichzeitig die Wiederverwendbarkeit und Unabhängigkeit dieser Systeme zu garantieren.

Wrapper in Mediator-Wrapper-Architekturen

XDA und JCA sind eng verwandt mit so genannten *Wrappern* der Mediator-Wrapper-Architekturen, wie sie von Wiederhold (1992) eingeführt wurden. Wrapper (Software-Adapter) kapseln die verschiedenen Informationsquellen, die integriert genutzt werden sollen. Sie übersetzen Anfragen eines Klienten in das spezifische Anfrageformat der Datenquelle, so dass diese die Anfragen bearbeiten kann. Zudem transformieren Wrapper das Ergebnis einer Anfrage in das vom Klienten benötigte Datenformat. Auf diese Weise lösen Wrapper die technische Heterogenität der beteiligten Datenquellen.

Jeder Wrapper ist quellenspezifisch, d.h. das vom Wrapper unterstützte Datenschema basiert auf den vorhandenen Daten und der Zugriffsschnittstelle der Quelle. Trotzdem können Wrapper wiederverwendet werden. Beispielsweise kann ein Wrapper für ein relationales Datenbankmanagementsystem (RDMBS) auch für Quellen genutzt werden, die das gleiche RDBMS einsetzen (Leser, 2000).

Auf Basis dieser Technik haben Bilek et al. (2004) *Wrapper-Agenten* realisiert, die verschiedene Datenbanksysteme in einem Multi-Agentensystem kapseln, um die Inhalte anderen Agenten zur Verfügung zu stellen. Wrapper-Agenten im Allgemeinen kapseln für andere

Agenten einer Plattform relevante Systeme und Datenquellen, damit diese keine direkte Verbindung zu diesen Ressourcen aufnehmen müssen (Klusch und Sycara, 2001). Die Kommunikation zwischen den Agenten geschieht in dem vorgestellten Ansatz auf Basis eines gemeinsam genutzten Vokabulars. Dieses Vokabular wird mit Hilfe einer spezifischen, domänenabhängigen Ontologie definiert, die Datenbankinhalte auf Nachrichtenelemente abbildet. Auch in diesem Ansatz übersetzt der Wrapper-Agent Anfragen anderer Agenten, die in einer „agentenweiten“ Anfragesprache formuliert sind, in Anfragen an die Datenbank, die gekapselt wird, und liefert anschließend die entsprechend der gemeinsam genutzten Ontologie umgewandelten Ergebnisse zurück.

Mediatoren in Mediator-Wrapper-Architekturen

Mediatoren in der von Wiederhold (1992) eingeführten Mediator-Wrapper-Architektur transformieren Anfragen des Benutzers entsprechend der verschiedenen Wrapper, an die sie die Anfragen weiterleiten. Die Benutzeranfragen an den Mediator werden über sein ihm innewohnendes Schema gestellt. Die Teilergebnisse der Wrapper werden in den Mediatoren entsprechend der Anfrage wieder zusammengemischt. Auf diese Weise überbrücken Mediatoren die strukturelle und semantische Heterogenität der einzelnen Datenquellen. Dieser Prozess, der auch *Query Execution Planning* genannt wird, beschreibt die Hauptaufgabe eines Mediators (Leser, 2000).

Gewöhnlich realisieren Mediatoren und Wrapper die gleiche Schnittstelle. Aus diesem Grund lassen sich Mediatoren wiederverwenden und hierarchisch anordnen. Mediatoren unterer Ebenen bereiten die Informationen für Mediatoren höherer Ebenen auf. Weiterhin sind Mediatoren häufig domänenspezifisch, d.h. sie ermöglichen den integrierten Zugriff auf Daten einer speziellen Anwendungsdomäne.

Viele Forschergruppen haben die ursprüngliche Idee der Mediator-Wrapper-Architektur aufgegriffen und Varianten bzw. Erweiterungen entwickelt. Zu der Vielzahl der Systeme gehören die von Carey et al. (1995); Chawathe et al. (1994); Levy et al. (1996); Qian und Lunt (1994); Yan et al. (1997). Neuere Ansätze kombinieren die Mediator-Wrapper-Architektur einerseits mit Ontologien, um eine semantische Integration zu ermöglichen, beispielsweise (Sattler et al., 2003), andererseits mit Softwaretechnologien wie Peer-To-Peer- oder Grid-Computing, um der Flut der verschiedenen Datenquellen, die im Internet entstehen, Herr zu werden, beispielsweise (Tatarinov et al., 2003).

Da die Problematik der Integration von Informationen aus verschiedenen Datenquellen orthogonal zu dem hier betrachteten Problem der Service-Interoperabilität ist, wird in dieser Arbeit nicht tiefergehend auf die Aspekte der Mediator-Wrapper-Architekturen eingegangen.

Informationsbus

Ein *Informationsbus* wird eingesetzt, um den Informationstransfer zwischen verschiedenen Komponenten zu verwalten, wobei explizite Abhängigkeiten zwischen den einzelnen Komponenten entfernt werden. Hierzu werden Interaktionsprotokolle und Schnittstellen definiert, die jede Komponente unterstützen muss, um sich bei dem Bus anzumelden (Eskelin, 1999). Folglich lässt sich ein Informationsbus als spezialisierter Softwaremediator interpretieren (Gamma et al., 1995), der auch von einer Art „Standardisierung“ der Schnittstellen Gebrauch macht. In diesem Kontext lassen sich durch Einsatz des Adapter-Entwurfsmusters auch *a priori* inkompatible Komponenten an einen gemeinsamen Informationsbus anschließen und somit ein offenes und erweiterbares *Plug-in*-Prinzip für Komponentenplattformen realisieren (Assmann und Radetzki, 2000; Bode et al., 2002).

Das ISYS System erlaubt ein Plug-in von domänenspezifischen Komponenten durch ein erweiterbares Framework (Siepel et al., 2001). In dem geschilderten Szenario werden heterogene biologische Systeme durch Komponenten repräsentiert und an einen Informationsbus einer komponentenbasierten Client-Plattform angeschlossen. Die durch den Informationsbus definierten Schnittstellen bilden die Grundlage für alle Komponenten und müssen von diesen unterstützt werden („Standardisierung“). Dieser Bus erlaubt den angeschlossenen Komponenten, Daten und Operationen durch einen Pull- und Push-Mechanismus zu transferieren. Dabei stützen sich die verschiedenen Komponenten auf generische Objekte und generische Events. Die generischen Objekte werden im wesentlichen durch ihre Attribute bestimmt und nicht durch eine abstrakte Typdefinition. Eine Erweiterung der Attributmenge kann nur durch direkte Programmierung geschehen und somit nicht zur Laufzeit. Ferner müssen sich die verschiedenen Komponenten auf die vorhandenen Attribute beziehen, da sie sonst nicht untereinander interagieren können. Somit ist eine vollständige Unabhängigkeit und Interoperabilität der beteiligten Komponenten nur bedingt erreicht.

Komponentenklebstoff

Während ein Informationsbus einen Routing-Mechanismus zur Verfügung stellt, mit dessen Hilfe Komponenten indirekt interagieren können, bietet der *Komponentenklebstoff* (*Component Glue*) Möglichkeiten, Schnittstellen zu verschalten und dabei eine Brücke über inkompatible Komponenten zu schlagen (Eskelin, 1999). Genauer gesagt, der entwickelte Glue-Code wird eingesetzt, um die versendeten Daten inkompatibler Datenformate entsprechend zu transformieren.

Beach (1992) führt beispielsweise den Informationsbus *Bart* ein, mit dessen Hilfe unabhängige Komponenten über eine Publish/Subscribe Metapher miteinander verknüpft werden können. Die Bart-Architektur adressiert den Nachrichtentransport, den Datenaustausch und die Transformation der Daten entsprechend der verschiedenen Datenmodelle der einzelnen Komponenten. Der Glue-Code der Transformation wird deklarativ in der so genannten *Software Glue Language* (SGL), einer Prolog ähnlichen Sprache, beschrieben.

SGL erfüllt die gleiche Rolle wie eine Sprache zur Definition von Sichten bei Datenbanken. Die vorgestellten Konzepte eignen sich jedoch vornehmlich für strukturelle (syntaktische) Transformationen. Semantische Aspekte werden in SGL nicht betrachtet. Die Kernidee der Transformationsmethode ist es, die Objektrepräsentationen in Relationen umzuwandeln, auf denen anschließend die Transformationsregeln angewendet werden, bevor eine Rücktransformation in eine neue Objektrepräsentation geschieht. Dabei entsprechen die Objekte einzelnen Tupeln in einer Relation.

Der in der COBIDS Plattform verwendete Glue-Code wird im Gegensatz zur Bart-Plattform durch vollwertige Softwarekomponenten beschrieben (Radetzki et al., 2002b; Bode et al., 2004). Hierdurch ist ihre Erstellung auf der einen Seite komplexer, auf der anderen Seite werden zudem auch anspruchsvollere Berechnungen unterstützt. Die Daten werden innerhalb der Plattform durch eine generische Objektrepräsentation beschrieben, die sowohl in XML als auch in Java vorliegen kann. Diese Repräsentation lässt sich im Unterschied zur ISYS-Plattform auch zur Laufzeit dynamisch erweitern, wenn es die problemspezifische Kopplung der Anwendungskomponenten erfordert. Ein Ziel des Ansatzes ist die hohe Wiederverwendbarkeit der einzelnen Glue-Codes, sowie deren Verkettung, um höherwertige Transformationen zu ermöglichen.

Glue-Code in serviceorientierter Software

Der Ansatz des Komponentenklebstoffes wird auch in WSFL unterstützt, einer Sprache zur Beschreibung von Servicekompositionen (Leymann, 2001). Durch ein *mapping*-Tag können derartige Transformationen in Form von XSLT beschrieben werden. Der Nachteil dieses Verfahrens ist, dass diese Skripte nur schlecht wiederverwendbar sind und somit für jeden Workflow neu geschrieben werden müssen. Radetzki et al. (2002a) haben dieses Modell durch das Konzept der *Extensible Mediator Tags* (XMT) erweitert, welches im wesentlichen auf XSLT-Extensions beruht und es erlaubt, neue XML-Tags zu definieren und diesen eine Implementierung durch eine portable Sprache, wie Java oder XSLT, zu zuordnen. Obwohl diese Tags prinzipiell in anderen Workflows wiederverwendet werden können, ist ihre Anpassung, ihr Auffinden und damit ihre Wiederverwendung problematisch.

Zu den in diesem Kapitel vorgestellten und in dieser Arbeit eingesetzten Konzepten sehr ähnlich ist das *Web Service Modeling Framework* (WSMF), welches als konzeptuelles Modell zur Entwicklung und Beschreibung von Web Services und deren Kompositionen dienen soll (Fensel und Bussler, 2002; Bussler et al., 2002)[14]. Die Ziele, die durch das WSMF erreicht werden sollen, sind eine maximale Entkopplung der in einem Workflow verwendeten Dienste, sowie eine skalierbare Mediation zwischen diesen. Dabei soll die Mediation sowohl zwischen verschiedenen Dokumentstrukturen als auch verschiedenen Geschäftsprozessen stattfinden. In WSMF besteht ein Modell aus vier verschiedenen Elementen:

[14]Das Framework ist kürzlich im *Web Services Execution Environment* (WSMX) aufgegangen (Moran und Mocan, 2004)

- Ontologien bieten Terminologien für die verwendeten Elemente an.
- Zieldefinitionen definieren das Geschäftsziel, welches durch die Web Services gelöst werden soll. Diese werden im wesentlichen durch Prä- und Postkonditionen spezifiziert.
- Web Services Beschreibungen definieren wesentliche Aspekte der Web Services.
- Mediatoren sollen die verschiedenen Aspekte der Interoperabilität lösen.

Die verschiedenen Mediationen, die im WSMF adressiert werden sollen, sind: Mediation von Datenstrukturen, Mediation von Geschäftslogiken, Mediation von Protokollen für den Nachrichtenaustausch und Mediation von dynamischen Serviceaufrufen (Fensel und Bussler, 2002). Leider werden in den vorliegenden Arbeiten weder konkrete Aussagen darüber getroffen, wie die angestrebten Ziele erreicht werden sollen, noch werden Implementierungsdetails offenbart.

Das *myGrid*-Projekt ist ein seit Ende 2001 von der UK EPSRC-gefördertes Pilotprojekt, welches aus einem Konsortium englischer Universitäten und Institute besteht. Im Kern werden im myGrid-Projekt Middlewarekomponenten entwickelt, die datenintensive *in silico* Experimente in der Biologie unterstützen sollen. Hierbei stehen die Beschreibungen und Realisierung von Workflows über Web Services im Vordergrund (Stevens et al., 2003).

Die kürzlich begonnenen Arbeiten im myGrid-Projekt hinsichtlich der so genannten *Shim Services* sind mit den hier vorgestellten Arbeiten eng verwandt. Auch Shim Services stellen Glue-Code bereit, um die Interoperabilität von Diensten in einem Workflow zu erreichen. Derzeitige Arbeiten konzentrieren sich auf eine initiale Klassifikation dieser Mediatoren in Kontext bioinformatischer Fragestellungen (Hull et al., 2004). Zukünftig soll auch hier das Management von Shim Services im Rahmen von myGrid-Workflows behandelt werden.

Schema-Matching

Das Problem, Service-Interoperabilität durch Glue-Code zu erreichen, lässt sich auch als automatisches *Schema-Matching Problem* auffassen (automatische Mediation). Die *Match*-Operation erhält in diesem Kontext zwei Schemata als Eingabe und produziert daraus ein *Mapping* zwischen den Elementen der Schemata, so dass sie semantisch zueinander korrespondieren. Die Schemata werden dabei durch die Schnittstellen und Nachrichten der datenproduzierenden und datenkonsumierenden Dienste spezifiziert. In diesem Kontext stellen Rahm und Bernstein (2001) in ihrem Übersichtsartikel eine Taxonomie vor, mit dessen Hilfe sie verschiedene Ansätze zum Schema-Matching Problem diskutieren. In dieser Taxonomie werden unter anderem Schema- und Instanzlevel, Element- und Strukturlevel sowie Sprach- und Constraint-Level unterschieden. Diese Taxonomie kann auch von Benutzern verwendet werden, um eine für ihr Problem passende Match-Realisierung auszuwählen.

Die Mappings werden in diesem Artikel zum einen durch eine Menge von *Mapping-Elementen* definiert, wobei jedes Mapping-Element anzeigt, dass bestimmte Elemente eines Schemas S_1 auf bestimmte Elemente eines Schemas S_2 abgebildet werden, zum anderen dadurch, dass jedem Mapping-Element ein *Mapping-Ausdruck* zugeordnet werden kann. Ein Mapping-Ausdruck spezifiziert, wie die Elemente von S_1 und S_2 in Beziehung stehen. Beispiele sind die $=$, $\leqslant$, *is-a* oder *part-of* Beziehungen. Der Fokus ihrer Arbeit bleibt jedoch hauptsächlich auf Mapping-Elementen, die keine Mapping-Ausdrücke beinhalten. Ein Grund dafür könnte die Schwierigkeit sein, komplexe Beziehungen, d.h. komplexe Transformationseinheiten, zu generieren. Ferner bleibt folgende Aussage bestehen (Rahm und Bernstein, 2001):

> *„In general, it is not possible to determine fully automatically all matches between two schemas, primarily because most schemas have some semantics that affects the matching criteria but is not formally expressed or often even documented. The implementation of Match should therefore only determine match candidates, which the user can accept, reject or change. Furthermore, the user should be able to specify matches for elements for which the system was unable to find satisfactory match candidates.“*

Um dem Anwender die Hürde bei der Erstellung eines konkreten Workflows zu nehmen, haben Ludäscher et al. (2003) ein Verfahren entwickelt, welches den benutzerfreundlichen Entwurf eines abstrakten Workflows ermöglicht, der anschließend in einen konkret ausführbaren Workflow kompiliert wird. Dabei gehen die Autoren davon aus, dass jeder datenproduzierende oder -konsumierende Service des Workflows sowohl semantisch als auch syntaktisch ausgezeichnet ist. Services können somit semantisch ähnlich, jedoch strukturell unterschiedlich sein. Die semantische Auszeichnung geschieht über eine Ontologie, die als globales Schema dient.

Werden bei der Kompilierung des abstrakten Workflows hin zum konkreten Workflow bei kompatiblen semantischen Typen inkonsistente Datentypen erkannt, werden vordefinierte Konvertierungsregeln in den konkreten Workflow eingesetzt. Schlägt dies fehl, beispielsweise weil keine adäquate Regel vorhanden ist, werden entsprechende Fehlernachrichten erzeugt.

Auf Basis der semantischen und syntaktischen Auszeichnung wurde zudem ein Ansatz zum Schema-Matching entwickelt (Bowers und Ludäscher, 2004). Die ontologischen Informationen werden ausgenutzt, um strukturelle Datentransformationen zu generieren. D.h. auf der Grundlage der semantischen Typen sollen *Mappings* für die strukturell heterogenen Typen der Dienste generieren werden. Das Framework setzt zur Generierung der benötigten Transformationseinheiten auf so genannte *Registration-Mappings*, die als Input- oder Output-Registration-Mapping vorliegen müssen. Diese durch Regeln definierten Mappings beschreiben Assoziationen zwischen den strukturierten und semantischen Datentypen eines Dienstes.

Der entwickelte Ansatz ist von den Grundgedanken mit der hier vorgestellten Arbeit eng verwandt; ja er kann sogar fließend in die hiesige Arbeit integriert werden. Der Ansatz

von Ludäscher et al. (2003) geht davon aus, dass die vorhandenen Services sowohl semantisch als auch syntaktisch annotiert sind. Dies ist gegenwärtig nicht Bestandteil der verfügbaren Web Services Standards. Die vorliegende Arbeit stellt diesbezüglich ein Verfahren vor, dass die semantischen Typen (semi-)automatisch herleitet.

Die Herleitung struktureller Transformationseinheiten geschieht über Registration-Mappings, die dem System bekannt sein müssen. Eine Abstrahierung der generierten Transformationseinheit, sowie Inkludierung von Anpassungsmöglichkeiten, so dass Wiederverwendung ermöglicht wird, ist nicht angedacht. Zudem lassen sich gewöhnlich nur einfache Transformationseinheiten im Sinne des Schema-Matchings erzeugen. Die Kombination anpassbarer, teilweise spezialisierter Transformationseinheiten mit automatisch generierten Transformationseinheiten wäre daher ein viel versprechender Ansatz.

3.4 Zusammenfassung

Dieses Kapitel liefert die Grundlagen für die Betrachtung der Problematik der Interoperabilität in service- und komponentenorientierter Software, die sich bei der Komposition von Services bzw. Komponenten ergibt. Diese Problematik gliedert sich in drei Ebenen: Technik, Syntax und Semantik der Schnittstelle und der Daten. Durch die Entwicklung und Standardisierung der Web Services Technologie wurde der technischen Heterogenität entgegengetreten. Die Aspekte Syntax – wie werden Informationen dargestellt: Datenformate und Datentypen – und Semantik – welche Bedeutung haben die Informationen: Konzepte – bleiben jedoch weitestgehend ungelöst. Erschwerend kommt hinzu, dass durch die Web Services Technologie die Semantik eines Services nicht offenbart wird.

Eine Standardisierung der Daten und Schnittstellen scheint hier nicht ausreichend, da Standardisierung ein sehr zeit- und kostenintensiver Prozess ist, der den aktuellen Anforderungen eines Anwendungsgebietes oder Projektes nicht genügen kann. Ansätze aus der Softwaretechnologie schlagen hier Adapter, Mediatoren oder Glue-Codes vor, deren Erstellung jedoch häufig Programmierkenntnisse voraussetzt, über die nicht jeder Workflowdesigner verfügt. Interaktive Verfahren, die den Nutzer bei der Erstellung dieser Module unterstützen, sowie automatisierte Ansätze zur Generierung, lösen das Problem nur bedingt. Sie sind vor allem dann gut einsetzbar, wenn einfache Datentransformationen angewendet werden müssen. In naturwissenschaftlichen Anwendungen werden allerdings häufig auch komplexere Code-Fragmente benötigt. Codebausteine, die speziell auf ein Problem zugeschnitten sind, sind ferner nur schlecht auf andere Anwendungskontexte übertragbar, wodurch ihre Wiederverwendbarkeit sehr gering ist. Somit kann der Workflowdesigner nicht ohne weiteres auf frühere Arbeiten zurückgreifen. Aus diesen Gründen scheint gerade die Kombination von Standardisierung mit softwaretechnischen und generativen Ansätzen unter Berücksichtigung der Aspekte Anpassbarkeit und Wiederverwendbarkeit sowie maschinenverarbeitbarer Semantik (Ontologien) ein ermutigender und gangbarer Weg zu sein.

Kapitel 4

Komponentenbasierte Service-Mediatoren

Service-Mediatoren bilden eine Softwarebrücke zwischen heterogenen Diensten und schaffen auf diese Weise die geforderte Service-Interoperabilität. Der komponentenbasierte Ansatz liefert zudem die nötige Flexibilität, diese Mediatoren in unterschiedlichen Anwendungskontexten einsetzen und wiederverwenden zu können. Um die Suche nach geeigneten Service-Mediatoren für einen konkreten Workflow ermöglichen zu können, müssen die Service-Mediatoren die Introspektion ihrer Fähigkeiten unterstützen. Dieses wird durch die auf OWL aufsetzende Beschreibungssprache MPL erreicht. MPL erlaubt sowohl die syntaktische als auch die semantische Auszeichnung der genutzten Elemente, sowie die Beschreibung der verschiedenen Adaptionsmöglichkeiten eines Service-Mediators. Zu letzteren zählen die Konfigurierung über zustandsbehaftete Eigenschaftsfelder sowie die Kompositionen von Mediatorfunktionalitäten.

Meint ihr denn, es müsse Stückwerk sein,
weil man es euch in Stücken gibt (und geben muss)?
— Friedrich Nietzsche (1844-1900), *Deutscher Philosoph*

Innerhalb eines benutzerdefinierten Workflows werden autonome und meist heterogene Dienste in einen sinnvollen Kontext gesetzt, um zusammen ein gegebenes (Geschäfts-) Problem zu lösen. Die Heterogenität der Schnittstellen (Operationssignaturen, Syntax und Semantik der Daten, etc.) verhindern jedoch häufig ein direktes Koppeln dieser Dienste, d.h. die benötigte Service-Interoperabilität ist nicht erreicht (vgl. Definition 3.3).

Der im Folgenden vorgestellte Ansatz greift die softwaretechnologischen Ansätze zur Erzielung der Interoperabilität auf (Abschnitt 3.3.2) und kombiniert diese mit Anpassungsfähigkeiten der Komponententechnologie (Abschnitt 2.1). Auf diese Weise entstehen die

so genannten *Service-Mediatoren*, die sowohl hochgradig wiederverwendbar als auch von Workflowdesignern ohne Programmierkenntnis einsetzbar sind[1].

Service-Mediatoren vermitteln innerhalb eines Workflows zwischen den datenproduzierenden und datenkonsumierenden Operationen, um somit die gewünschte Service-Interoperabilität zu erzielen, d.h. aufgabenspezifische Service-Mediatoren werden zur Überbrückung der vorhandenen semantischen und syntaktischen Heterogenität der beteiligten Dienste eingesetzt.

Lassen sich für einen gegeben Workflow keine entsprechenden vorgefertigten Service-Mediatoren identifizieren, müssen neue Mediatoren erstellt werden. Hier können Ansätze der automatischen Mediation (z.B. Schema-Matching-Verfahren (Rahm und Bernstein, 2001)) gewinnbringend als spezielle Form der Anpassung integriert werden.

Im Zentrum dieses Kapitels steht daher die Frage: Was ist ein Service-Mediator und was ist er nicht? Um diese Frage zu lösen, werden anfangs die allgemeinen Services eines Workflows den Service-Mediatoren gegenübergestellt (Abschnitt 4.1) und mögliche Klassifikationen der Service-Mediatoren gegeben (Abschnitt 4.2). Anschließend werden in Abschnitt 4.3 die softwaretechnischen Anforderungen an diese spezialisierten Softwarekomponenten aufgestellt. Die genaue Spezifikation des Komponentenmodells der Service-Mediatoren findet in Abschnitt 4.4 statt. Die *Mediatorprofilsprache* (MPL), die zur Beschreibung der Fähigkeiten eines Mediators sowie als Anfragespezifikation zur Identifikation benötigter Service-Mediatoren dient, wird in Abschnitt 4.5 vorgestellt. Der darauf folgende Abschnitt 4.6 vergleicht MPL mit anderen Ansätzen aus dem Bereich der Web Services.

4.1 Service-Mediatoren und Services – Eine Gegenüberstellung

Service-Mediatoren und Services sind für die Erstellung eines korrekten Workflows essentiell. Was aber unterscheidet diese Konzepte voneinander bzw. welche Gemeinsamkeit haben sie? Tabelle 4.1 liefert hierzu eine Gegenüberstellung (Radetzki et al., 2004b). Aus technischer Sicht sind sie annähernd identisch, da beide durch Standard Web Services Technologie beschrieben und angesprochen werden. Vor allem WSDL ist hier zu nennen. WSDL alleine ist jedoch zu schwach für die im Laufe dieses Kapitels aufgestellten Anforderungen an

[1] Um sich vom Begriff *Code* in Glue-Code zu lösen und den *Übergang* von einer Servicebeschreibung zu einer anderen Servicebeschreibung zu forcieren, werden die Verbindungseinheiten in dieser Arbeit *Service-Mediatoren* oder kurz *Mediatoren* genannt. Hierzu zählen unter anderem spezielle Datentransformatoren, Filter und Komparatoren. Weitere ähnliche Konzepte und Termini, die in der Literatur verwendet werden, sind Adapter, Bridge, Connector, Component Glue oder erst kürzlich Shim. Sie sind nicht mit den Mediatoren verwandt, die Wiederhold (1992) eingeführt hat. Vergleiche auch Kapitel 3.3 über verwandte Arbeiten.

	Service	Service-Mediator
Granularität	hoch	niedrig
Nutzung	explizit	implizit
Kontext	Workflows, z.B. *in silico* Experimente	Mediation zwischen Services in einem Workflow
Lösungsraum	(Geschäfts-) Probleme, beispielsweise eines *in silico* Experiments	„mediatorbasierte“ Service-Interoperabilität
Beispiele	BLAST, Publikationsdienst	AGAVE-to-BSML-Konverter
Technische Aspekte		
Schnittstelle	Web Services (WSDL)	Web Services (WSDL) und MPL
Konfigurierung	derzeit nicht explizit unterstützt	unterstützt

Tabelle 4.1: Service-Mediatoren und Services – Eine Gegenüberstellung

die Service-Mediatoren. Beispielsweise erlaubt WSDL keine explizite Auszeichnung von semantischen Konzepten. Daher werden Service-Mediatoren ferner durch die Mediatorprofilsprache (MPL) beschrieben. Ein weiterer technischer Aspekt, der derzeit von Web Services nicht unterstützt wird, ist die Fähigkeit zur Konfigurierung über explizit ausgezeichnete Konfigurationsfelder. Dies könnte sich jedoch mit Abschluss der WSDL 2.0 Spezifikation[2] ändern, da auch hier explizit *Eigenschaften* (*properties*) aufgenommen werden sollen. Es bleibt jedoch abzuwarten, ob die Property-Konzepte von WSDL 2.0 und MPL äquivalent sind.

Services haben im Gegensatz zu Service-Mediatoren eine *höhere* Granularität. Dies liegt vor allem daran, dass sie dazu dienen, komplexe (Geschäfts-) Probleme innerhalb eines Workflows zu lösen. Häufig arbeiten Services über Datenbanken und bieten Anwendern spezifische Zugriffsmöglichkeiten auf diese Datenbanken. Service-Mediatoren auf der anderen Seite haben eine *feinere* Granularität. Sie könnte man auch als leichtgewichtige Komponenten (*light-weight components*) bezeichnen. Service-Mediatoren sollen beispielsweise flexibel aus dem Internet geladen[3] und in einen Workflow eingebaut werden können, um hier die geforderte Service-Interoperabilität zu erreichen. Sie sind alles das, was man für genau diesen Problemkontext benötigt (verschiedene Varianten von Service-Mediatoren werden weiter unten in diesem Kapitel vorgestellt).

Beispiel 4.1. In einem *in silico* Experiment gehören beispielsweise BLAST-Dienste über Nucleotid- oder Proteindatenbanken oder Recherchedienste der Klasse der Ser-

[2]Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language
[3]Verteilter versus portabler Service-Mediator wird in Abschnitt 4.3.3 behandelt.

vices an. Die Selektion bestimmter Elemente eines SWISSPROT-Eintrags wird hingegen durch einen Service-Mediator beschrieben.

Ein weiterer wesentlicher Unterschied zwischen diesen Konzepten ist ihre Nutzung. Services werden *explizit* durch Einbettung ihrer WSDL-Beschreibung in die Spezifikation eines Workflows eingebettet. Mediatoren hingegen werden weitestgehend automatisch identifiziert und in den Workflow integriert, um zwischen zwei nicht-interoperablen Diensten zu vermitteln. Ihre Nutzung ist möglichst *implizit*, d.h. sie stellen auch keinen direkten Bestandteil eines Workflows dar.

Abschließend ergibt sich die Frage, ob ein auf der Web Services Technologie basierender Dienst sowohl im obigen Sinne Service als auch Service-Mediator sein kann? Betrachten wir hierzu folgende Beispiele:

Beispiel 4.2. Sei S ein Server, der die Realtime-Kurse mehrerer Währungen liefert, dann könnte S für einen Kurs-Ticker und einen Währungsrechner eingesetzt werden. Kurs-Ticker und S sind in diesem Beispiel identisch und beschreiben einen Service im obigen Sinne, während der Währungsrechner zur Berechnung eines Betrages in eine andere Währung lediglich auf S aufsetzt. Im letzteren Fall würde man von einem Service-Mediator sprechen.

Das Beispiel illustriert, dass ein Service-Mediator auf einen regulären Service aufsetzen kann, jedoch anders eingesetzt wird und eine andere Semantik als der Service besitzt. Ein ähnliches Szenario haben Bode et al. (2004) im Rahmen geowissenschaftlicher Fragestellungen erörtert:

Beispiel 4.3. Die Rasterbilddaten eines *OGC Web Map Services* (WMS) mussten einer Visualisierungskomponente in geeigneter Art und Weise zur Verfügung gestellt werden. Dies erforderte jedoch teilweise ein Resampling bzw. Konvertieren der Bilddaten. Um diese Konvertierung verlustfrei zu ermöglichen, musste eine spezielle Transformationseinheit Metadaten über das Rasterbild entsprechend anpassen und diese dann an den WMS senden, um so ein neues Rasterbild zu erhalten. Auch hier stellt der WMS einen Service im obigen Sinne dar, während die beschriebene Transformationseinheit ein Service-Mediator ist, der auf die Fähigkeiten eines Services aufbaut.

4.2 Klassifikation verschiedener Service-Mediatoren

Wie obige Diskussion andeutet, lassen sich auch Service-Mediatoren in verschiedene Klassen und Kategorien einteilen. Eine mögliche Einteilung ist die in *deterministische* und *nichtdeterministische* Service-Mediatoren. Deterministische Service-Mediatoren liefern auf eine

konkrete Eingabe immer die gleiche Ausgabe, während das Ergebnis bei nicht-deterministischen Mediatoren im Allgemeinen nicht vorhersagbar ist. Dies ist beispielsweise der Fall, wenn ein Service-Mediator auf eine Datenbank zugreift, die unabhängig vom Mediator ihren Zustand ändern kann. Eine andere grundlegende Einteilung ist die in *domänenunspezifische* Service-Mediatoren, die in verschiedenen Anwendungskontexten eingesetzt werden können, und *domänenspezifische* Service-Mediatoren, die im Allgemeinen nur in bestimmten Anwendungsdomänen Verwendung finden.

> **Beispiel 4.4.** Der in Beispiel 4.2 vorgestellte Service-Mediator ist nicht-deterministisch, weil das konkrete Resultat der Umrechnung vom aktuellen Kurs bzw. aktuellen Zustand der Datenbank abhängig ist. Er ist ferner domänenunspezifisch, da Währungstransformationen in verschiedenen Domänen Anwendung finden, wie Finanzwesen, Touristik oder eCommerce.

Ein weiterer, erster Versuch, verschiedene Klassen von Service-Mediatoren zu identifizieren, wurde von Hull et al. (2004) im Kontext biologischer Fragestellungen durchgeführt. Sie unterscheiden primär zwei Hauptkategorien, *Filter* und *Transformatoren*. Zu der Kategorie der Filter gehören beispielsweise *Sortierer* und *Parser*. Die Untertypen der Transformatoren beinhalten *Dereferenzierer*, die einen Identifikator durch das konkret referenzierte Objekt austauschen, *Mapper*, die zwischen äquivalenten Einträgen abbilden, und *Übersetzer* oder *Konvertierer*. Die im Rahmen dieser Arbeit durchgeführte Analyse komponierter Service-Mediatoren zeigt ferner, dass auch der Bedarf besteht, komplexe Vergleiche über Daten oder Konfigurationen in der Komposition von Mediatoren ausdrücken zu können. Dies gilt vor allem in wissenschaftlichen Anwendungsgebieten. Service-Mediatoren dieser Klasse werden im Folgenden *Komparatoren* genannt (Radetzki et al., 2004a).

> **Beispiel 4.5.** Die Selektion aller Gene eines Chromosoms gehört der Klasse der Sortierer an. Ein Parser kann beispielsweise die Sequenz aus einem SWISSPROT Eintrag selektieren. Ein Dereferenzierer kann einen GenBank Identifikator durch den GenBank Eintrag ersetzen. Die Abbildung eines GenBank Identifikators auf einen EMBL Identifikator wird durch einen Mapper beschrieben. Die Konvertierung von Daten aus BSML in AGAVE gehört den Übersetzern oder Konvertierern an. Ob ein Protein durch ein Gen gebildet werden kann, lässt sich durch Service-Mediatoren der Komparator-Klasse bestimmen.

Die Definition und Bestimmung derartiger Klassifikationstaxonomien ist sinnvoll, um den Suchraum nach für einen konkreten Workflow benötigten Service-Mediatoren eingrenzen zu können (vgl. auch Kapitel 5.3).

4.3 Softwaretechnische Anforderungen

In den ersten beiden Abschnitten dieses Kapitels wurde eine Charakterisierung der Service-Mediatoren aufgestellt, um sie von allgemeinen (Geschäfts-) Diensten unterscheiden zu

können. Dieser Abschnitt fokussiert die softwaretechnischen Anforderungen an einen Service-Mediator, der ihn ebenfalls von allgemeinen Services und Komponenten unterscheidet. Zu den wesentlichen Anforderungen zählen hohe *Wiederverwendbarkeit* und die Exploration der Fähigkeiten (*Introspektion*). Allgemeine nicht-funktionale Anforderungen an Service-Mediatoren oder deren Laufzeitumgebung, wie Sicherheit, Transaktionen, Caching der Daten und Performanz, werden nicht näher behandelt, da sie nicht im Fokus dieser Arbeit liegen.

In dieser Arbeit wird für die vorgeschlagenen Service-Mediatoren ein *komponentenbasierter Ansatz* gefordert. Wird dieser Softwareansatz sorgfältig umgesetzt, kann hierdurch eine hohe Wiederverwendbarkeit der Softwarebausteine erzielt werden (vgl. Kapitel 2.1). Zu den spezifischen Anforderungen an einen komponentenbasierten Service-Mediator gehören zudem:

- Beschreibung und Introspektion der Schnittstelle,
- Anpassbarkeit an verschiedene Anwendungskontexte, insbesondere eine Kompositionsfähigkeit,
- Unabhängigkeit von Sprache, Plattform und System, und
- portable bzw. verteilte Realisierung.

Durch die Unterstützung dieser Anforderungen sollen möglichst in verschiedenen Workflows universell einsetzbare und wiederverwendbare Service-Mediatoren entstehen.

4.3.1 Beschreibung und Introspektion der Schnittstelle

Jeder Service-Mediator besitzt eine vertraglich zugesicherte, mediatorspezifische Schnittstelle, die die Fähigkeiten des Mediators beschreibt. Dies allein geht bereits aus der grundlegenden Forderung an eine Softwarekomponente hervor (Szyperski, 1998). Daher werden Mechanismen benötigt, diese Schnittstelle syntaktisch und, wenn möglich, auch semantisch zu beschreiben. Ferner sollten weitestgehend standardisierte Sprachen zur Schnittstellenbeschreibung eingesetzt werden. Um auch *a priori* unbekannte Service-Mediatoren in einem Workflow einsetzen und aktivieren zu können, wird ferner ein Introspektionsmechanismus benötigt, der die Fähigkeiten eines Service-Mediators offenbart. Unter *Introspektion* sei die Möglichkeit eines Systems oder einer Komponente verstanden, die Fähigkeiten einer Komponente zur Laufzeit explorieren zu können. Im Falle der Service-Mediatoren zählen hierzu beispielsweise die Anpassungsfähigkeiten und die Mediatorfunktionalitäten[4]. Dieser Mechanismus ist sowohl für die Identifizierung geeigneter Service-Mediatoren essentiell (vgl. Kapitel 5) als auch für die den Service-Mediator ausführende Laufzeitumgebung.

[4]Mediatorfunktionalitäten werden in Abschnitt 4.4 spezifiziert, hier können sie allgemein als Transformationsvorschriften verstanden werden.

Im Bereich der Web Services Technologie wird WSDL zur Beschreibung der Schnittstelle eingesetzt. WSDL besteht aus einem technischen Anteil, der den Zielpunkt eines Service beschreibt, sowie einem abstrakten Anteil, der den syntaktischen Port-Typ definiert. Um an dieser Stelle „das Rad nicht neu zu erfinden“ und den gegenwärtigen Standard auszunutzen, sollten auf technischer Ebene Service-Mediatoren WSDL benutzen.

Während sich mittels WSDL nur technische und syntaktische Aspekte der Schnittstellen beschreiben lassen, sollten Service-Mediatoren auch semantische Aspekte offenbaren, um eine möglichst optimale Wiederverwendung zu erreichen. Dies kann beispielsweise durch die direkte Annotation von Konzepten zu den Datentypen geschehen. Allgemein sollten die Konzepte in einer Domänenontologie durch standardisierte Beschreibungssprachen wie OWL definiert werden. Die Festlegung einer entsprechenden Ontologie sollte durch ein entsprechendes Standardisierungsgremium erfolgen.

Beispiel 4.6. Sei f^{M_1} die Mediatorfunktionalität des Mediators M_1 aus Beispiel 4.2. f^{M_1} überführt einen Geldbetrag v_1 einer bestimmten Währung c_1 in einen Geldbetrag v_2 einer anderen, beliebigen Währung c_2. Die Zielwährung c_2 sei frei wählbar. v_1 sei durch den Typ *double* repräsentiert. Dieser Typ sagt jedoch noch nichts über seine Semantik aus. In einer Ontologie, die Finanzprodukte beschreibt, könnte das Konzept des *Geldes* eingeführt werden. Dieses Konzept kann dann von f^{M_1} zur Annotation von v_1 und v_2 eingesetzt werden.

WSDL unterstützt ferner keine explizite Definition von Anpassungsfähigkeiten, die zu den wesentlichen Anforderungen an Service-Mediatoren gehören, um eine möglichst hohe Wiederverwendbarkeit zu erreichen.

Um die oben aufgeführten Einschränkungen von WSDL zu beheben, wurde auf Basis von OWL die Mediatorprofilsprache (MPL) entwickelt. Mit ihr lassen sich die Fähigkeiten eines Service-Mediators in einem Profil beschreiben. Jedem Service-Mediator muss ein derartiges Profil zugeordnet sein. MPL wird in Kapitel 4.5 vorgestellt.

4.3.2 Anpassbarkeit an verschiedene Anwendungskontexte

Aus der Forderung, eine hohe Wiederverwendbarkeit eines Service-Mediators zu erreichen, resultiert, dass im Komponentenmodell der Service-Mediatoren *Anpassungsfähigkeiten* vorhanden sind. Denn, wie bereits in Kapitel 2.1.2 dargelegt, wirkt sich die Möglichkeit der Adaption positiv auf die Wiederverwendbarkeit einer Komponente in verschiedenen Anwendungskontexten aus. Die von Won (2004) vorgeschlagenen Anpassungsfähigkeiten für allgemeine Komponentenmodelle lassen sich auch auf Service-Mediatoren übertragen und für diese weiter konkretisieren.

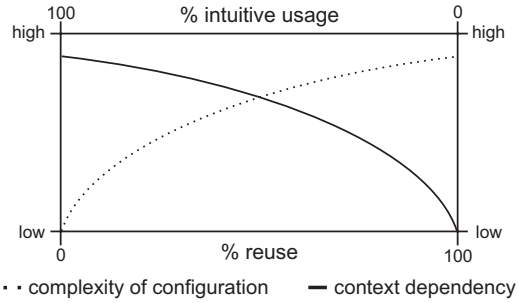

Abbildung 4.1: Grad der Wiederverwendbarkeit versus intuitive Nutzung in Bezug auf Kontextabhängigkeit und Komplexität der Konfigurierung.

Konfigurierung

Service-Mediatoren sollten spezifische, möglichst zustandsbehaftete (*stateful*) Eigenschaftsfelder haben, über die ein Benutzer oder ein System ihr Verhalten manipulieren und dem gegebenen Workflow entsprechend anpassen kann. Diese *Eigenschaftsfelder*, auch *Properties* genannt, können beispielsweise dazu dienen, ein Zielformat festzulegen oder Parameter eines Algorithmus' zu spezifizieren. Sind die Properties zustandsbehaftet, kann ihre konkrete Anpassung für einen gegebenen Workflow persistent abgelegt werden und ist somit für zukünftige Workflows, die die gleichen oder „ähnliche" Services verwenden, nutzbar. Voraussetzung ist, dass aus der Schnittstellenbeschreibung hervorgeht, (a) welche Eigenschaftsfelder vorhanden sind und (b) welche Funktionalitäten durch ein Eigenschaftsfeld moduliert werden.

> **Beispiel 4.7.** Die Funktionalität f^{M_1} aus Beispiel 4.6 besitzt beispielsweise eine Property, mit der die Zielwährung spezifiziert werden kann. Für einen Service-Mediator, der Rasterbilder verschiedener Formate in ein GIF Format konvertiert, könnte die Farbe, die bei der Konvertierung des Bildes in das GIF Format transparent dargestellt werden soll, eine Property sein.

Bei der Konfigurierung von Service-Mediatoren durch Properties ist Folgendes zu beachten: Der *Grad der Wiederverwendbarkeit* bestimmt maßgeblich die *Komplexität bei der Nutzung* des Service-Mediators, d.h. die *Intuitivität* mit der der Mediator genutzt werden kann. Abbildung 4.1 zeigt die Abhängigkeit dieser beiden Kräfte in Bezug auf die Konfigurierung.

Im Gegensatz zu Abbildung 2.1 wird hier der Grad der Wiederverwendbarkeit versus der intuitiven Benutzung des Mediators hauptsächlich durch die *Abhängigkeit zum Anwendungskontext* und der *Komplexität der Anpassung durch die Konfigurierungsoperation* bestimmt. Der Anwendungskontext wird durch den Workflow bzw. durch das zu lösende Problem gegeben.

Ein Entwickler eines neuen Mediators sollte neben den generellen *trade-offs* bei der Erstellung einer neuen Softwarekomponente (vgl. Kapitel 2.1) auch diese trade-offs kennen und sich ihrer Konsequenzen bewusst sein, um einen möglichst allgemein in verschiedenen Workflows sinnvoll wiederverwendbaren und gleichzeitig intuitiv einsetzbaren Service-Mediator zu realisieren. Denn wenn für alle Operationen, die durch Workflows verknüpft werden, neue Mediatoren realisiert werden müssen, ist der Idee der Komponententechnologie nicht Rechnung getragen (Komplexität der Konfigurierung gering, Kontextabhängigkeit sehr hoch, *aber* Wiederverwendbarkeit sehr gering). Genauso umgekehrt, wenn die Konfigurierung eines Service-Mediators einer Programmiertätigkeit gleicht, ist dem Nutzer ebenso wenig geholfen (Komplexität der Konfigurierung sehr hoch, Kontextabhängigkeit sehr gering, Wiederverwendbarkeit sehr hoch, *aber* intuitive Nutzbarkeit sehr gering).

Beispiel 4.8. Sei f^{M_2} eine Mediatorfunktionalität eines Mediators M_2, die beliebige XML-Dokumente in XML-Dokumente eines anderen, aber ebenfalls beliebigen Formates transformieren kann. D.h. f^{M_2} generiert basierend auf einem Eingangsdokument d_1 eine neue spezifische Sicht d_2 auf dieses Dokument. Um die gewünschte Transformation zu erreichen, muss der Mediator durch spezifische XQuery- oder XSLT-Regeln angepasst werden.

f^{M_2} lässt sich positiv durch seine universelle Einsetzbarkeit und Wiederverwendbarkeit charakterisieren. Zudem ist sie nur sehr schwach kontextabhängig, da die einzige Einschränkung durch die erzwungene Beschreibungssprache der Dokumente in XML gegeben ist. Auf der anderen Seite gleicht die Konfigurierung, die beim Einsetzen des Service-Mediators benötigt wird, jedoch eher einer Programmiertätigkeit. Diese ist nicht von jedem Endbenutzer bzw. von einem System direkt realisierbar. Aus diesem Grunde ist die intuitive Benutzung des Mediators beschränkt.

f^{M_1} aus Beispiel 4.6 besitzt eine höhere Kontextabhängigkeit als f^{M_2}, wodurch f^{M_1} nicht mehr ganz allgemein in allen Anwendungsgebieten eingesetzt werden kann. Durch die mögliche Konfigurierung der Zielwährung kann diese Mediator trotzdem relativ universell in entsprechend relevanten Anwendungskontexten eingesetzt werden. Aufgrund der im wesentlichen trivialen Anpassung, ist die intuitive Nutzung als hoch anzusehen. Diese Anpassung könnte beispielsweise auch durch ein „intelligenteres“ System automatisch geschehen, wenn das Ziel der Datentransformation, in diesem Fall der datenkonsumierende Service, die benötigte Währung dem System in geeignete Weise zur Verfügung stellt.

Sei f^{M_3} eine Mediatorfunktionalität eines Mediators M_3, die vorhandene XML-Formate mit biologischem Kontext in ein anderes, vorhandenes XML-Format mit biologischem Kontext übertragen kann. Hierbei seien die unterstützen Formate GAME, BSML und AGAVE. Die Anpassung besteht darin, das entsprechende Zielformat festzulegen.

f^{M_3} ist nur in Anwendungsdomänen mit rein biologischem Kontext sinnvoll. Daher ist die allgemeine Wiederverwendbarkeit schwächer als die von f^{M_1} und f^{M_2}. Die Anpassung von f^{M_3} besteht hingegen lediglich aus einer einfachen Angabe des Zielformates. Auch diese könnte ähnlich wie bei f^{M_1} durch ein speziell im biologischen Kontext arbeitendes System direkt vorgenommen werden.

Komposition von Service-Mediatoren

Die Komposition von Service-Mediatoren ist eine weitere essentielle Basisoperation der Anpassbarkeit, die Service-Mediatoren unterstützen müssen, da häufig eine mediatorbasierte Service-Interoperabilität erst durch eine aufgabenbezogene Kopplung mehrerer Service-Mediatoren erreicht werden kann. Dazu müssen die Service-Mediatoren wie bereits beim Introspektionsmechanismus diskutiert, mindestens *Blackbox Reuse* unterstützen (vgl. Kapitel 2.1.1).

Manipulation von Mediatorkompositionen

Neben der Neuerstellung von Mediatorkompositionen sollte es auch möglich sein, vorhandene Mediatorkompositionen manipulieren zu können, um sie einem neuen Workflow entsprechend anzupassen. Dies erhöht ebenfalls die Wiederverwendbarkeit von derart *komplexen* Service-Mediatoren. Dazu ist es jedoch notwendig, dass komponierte Service-Mediatoren *Whitebox Reuse* unterstützen (vgl. Kapitel 2.1.1). Aufgrund der Problematik des Whitebox Reuse-Ansatzes empfiehlt sich, erst eine Kopie des komponierten Service-Mediators anzufertigen, die anschließend angepasst wird. Auf diese Weise sind Original und Kopie unabhängig voneinander und andere Service-Mediatoren, die sich auf das Original beziehen, verlieren nicht ihre Korrektheit. Aus wirtschaftlichen / firmenpolitischen Gründen kann es jedoch auch Kompositionen von Service-Mediatoren geben, die lediglich *Blackbox Reuse* unterstützen, um beispielsweise Firmengeheimnisse nicht preisgeben zu müssen.

In Analogie zu Won (2004) zählen zu den Manipulationsfähigkeiten das Hinzufügen oder Entfernen von Service-Mediatoren einer Komponentenzusammenstellung sowie das Ändern der Verbindungen zwischen den Service-Mediatoren innerhalb einer Komposition.

Implementierung eines Service-Mediators und Änderung des Zustands des Pools zur Verfügung stehender Mediatoren

Ein neuer Workflow bzw. ein neues Anwendungsgebiet kann es bedingen, dass neue Service-Mediatoren *from Scratch* entwickelt werden müssen. Diese Mediatoren werden einem Pool der zur Verfügung stehenden Mediatoren hinzugefügt. Dieser Pool wird im Folgenden kurz *Mediatorpool* genannt. Der Mediatorpool beinhaltet die für die Komposition zur Verfügung stehenden Service-Mediatoren und bestimmt damit die potentiell mögliche Funktionalität eines daraus zu bildenden komponierten Service-Mediators. Da der Zustand des Mediatorpools Änderungen unterworfen ist, hat er indirekt Auswirkungen auf andere Anpassungsfähigkeiten.

Für eine konkrete Schnittstellenbeschreibung kann es mehrere Service-Mediatoren geben, die diese realisieren. Beispielsweise kann eine Schnittstelle durch Service-Mediatoren von verschiedenen Anbietern realisiert werden. Daher sollte der Benutzer den für sein Problem optimalen Service-Mediator selektieren können.

4.3.3 Unabhängigkeit und Verfügbarkeit

Service-Mediatoren können komplexe Berechnungsfunktionalitäten bereitstellen. Daher ist es erforderlich, Service-Mediatoren sprachunabhängig entwickeln zu können, d.h. der Entwickler sollte angesichts der Anforderungen an die Mediation eine für das Problem geeignete Programmiersprache auswählen können. WSDL unterstützt diese Unabhängigkeit durch die Erweiterungsfähigkeiten der Bindungen, d.h. es ist möglich neue Programmiersprachen durch eine neue Bindung bekannt machen zu können. Es hängt dann von der verwendeten Laufzeitumgebung ab, ob diese Bindung unterstützt wird. Derzeit sind Bindungen spezifiziert für SOAP, HTTP, EJB und Java (vgl. hierzu beispielsweise das *Web Services Invocation Framework* (WSIF)[5]).

Weiterhin ist eine Unabhängigkeit von der gewählten Plattform (z.B. Linux oder Windows) oder des gewählten Systems, d.h. der genutzten Laufzeitumgebung, zu erreichen. Dies erhöht ebenfalls die mögliche Wiederverwendung des Mediators. Auch dieses wird bereits durch WSDL ermöglicht.

Beispiel 4.9. Die Umrechnung von Koordinaten des Längen/Breiten-Koordinatensystems in Koordinaten des Grauß-Krüger-Koordinatensystems ist eine komplexe Berechnungsfunktion, die nur schwerlich mittels Sprachen wie XQuery oder Prolog umgesetzt werden kann, ohne dabei von eventuellen Erweiterungsmöglichkeiten Gebrauch zumachen. Hier werden Programmiersprachen wie C++ oder Java benötigt.

Werden spezifische Anforderungen an die verwendeten Ressourcen von einem Service-Mediator gestellt, wie spezielle Bibliotheken, Datenbanken oder die genutzte Plattform, muss sich der Service-Mediator auch bequem als verteilte Komponente realisieren und über einen *remote*-Zugriff ansprechen lassen. Die Realisierung eines mobilen und portablen Service-Mediators hat den Vorteil, den Mediator in der genutzten Laufzeitumgebung *in situ* ausführen zu können. Hierdurch lässt sich neben Performanzvorteilen auch geschickt die Netzlast bei der Ausführung des Workflows reduzieren, da auf bereits geladene Service-Mediatoren direkt zugegriffen werden kann.

Beispiel 4.10. Die mobile versus verteilte Variante der Verfügbarkeit von Service-Mediatoren lässt sich gut an den Mediatoren M_1 und M_2 illustrieren. Da gewöhnlich XSLT auf allen Plattformen vorhanden ist, könnte die Mediatorfunktionalität f^{M_2} gut als mobile Einheit realisiert werden. Wenn hingegen M^2 nur XQuery unterstützt, wäre ein verteilter Zugriff sinnvoll, da zur Zeit eine XQuery-Implementierung nicht auf allen Rechner zur Verfügung steht. Nehmen wir bezüglich des Mediators M^1 an, dass f^{M_1} von einer Bank zur Verfügung gestellt wird und auf einen nur intern verfügbaren Realtime-Kurs zugreift, dann ließe sich f^{M_1} ebenfalls nur innerhalb der Bank mit der Möglichkeit eines verteilten Zugriffs realisieren.

[5] `ws.apache.org/wsif/`

4.4 Spezifikation des Komponentenmodells

Aus den softwaretechnischen Anforderungen an Service-Mediatoren lassen sich die zwei elementare Bedingungen für Service-Mediatoren ableiten:

(a) Service-Mediatoren müssen identifiziert werden können. Daher wird ein Introspektionsmechanismus benötigt, der die Fähigkeiten eines Service-Mediators offenbart. Diesem Aspekt wird durch die *Mediatorprofilsprache* Rechnung getragen.

(b) Service-Mediatoren müssen Anpassungsfähigkeiten bereitstellen, damit sie flexibel in verschiedenen Workflows eingesetzt und wiederverwendet werden können. Diese Unterstützung wird im *Komponentenmodell der Service-Mediatoren* festgeschrieben.

Die Anforderungen nach Unabhängigkeit und Verfügbarkeit eines Service-Mediators lassen sich prinzipiell durch den Einsatz von WSDL erreichen und sollen daher nicht weiter beleuchtet werden.

Wie bereits in Kapitel 2.1 beschrieben, beinhaltet ein Komponentenmodell eine Menge von Regeln und Spezifikationen über die Art und Weise, wie Komponenten erstellt und komponiert werden können. Die Elemente des Komponentenmodells für die Service-Mediatoren beinhalten Eigenschaftsfelder, Mediatorfunktionalitäten und die Service-Mediatoren selbst. Services im Allgemeinen sind von dem hier beschriebenen Komponentenmodell völlig unberührt. Sie werden weiterhin durch WSDL beschrieben und können mittels BPEL4WS komponiert werden. Das Modell erlaubt die Erstellung neuer sowie die Anpassung vorhandener Service-Mediatoren und wird daher sowohl vom Mediator-Entwickler als auch vom Workflowdesigner eingesetzt.

4.4.1 Elemente des Komponentenmodells

Die Konfigurierung einer Mediatorfunktionalität wird, wie bereits angedeutet, durch *zustandsbehaftete Eigenschaftsfelder* (*Properties*) realisiert, mit deren Hilfe das Verhalten einer Funktionalität den Anforderungen des Workflows entsprechend modifiziert und angepasst werden kann.

Definition 4.1 (Eigenschaftsfeld)**.** Ein *Eigenschaftsfeld* (*Property*) pr ist definiert als $pr = (\omega_{pr}, d_{pr}, C_{pr}, desc_{pr})$, wobei

ω_{pr} ist der Name des Eigenschaftsfeldes (*Property-Name*),
$d_{pr} \in \mathcal{D}$ ist der Datentyp des Eigenschaftsfeldes,
$C_{pr} \subseteq \mathcal{C}$ ist eine endliche Menge von Konzepten und
$desc_{pr}$ ist eine inhaltliche, unstrukturierte Beschreibung.

Sowohl der Property-Name als auch die Beschreibung werden durch Worte über einem beliebigen Alphabet gebildet. ■

Bemerkung: Vergleicht man die Definition 4.1 mit der Definition 3.2 erkennt man schnell, dass Eigenschaftsfelder eine spezielle Form der erweiterten Ports sind. Sie werden genauso beschrieben wie erweiterte Ports, dienen aber einem anderen Zweck. Eigenschaftsfelder modulieren Verhalten, Input- und Outputports dienen der Datenübertragung.

Konfigurierbare Mediatorfunktionalitäten sind die Einheiten, die zwischen den jeweiligen heterogenen Diensten vermitteln. Sie stellen auch den komponierbaren Teil der Service-Mediatoren dar.

Definition 4.2 (Mediatorfunktionalität)**.** Eine *Mediatorfunktionalität*, kurz *Funktionalität*, f wird beschrieben durch

$$f : (i_f, \mathcal{P}_f^{\wp}) \rightarrow (o_f),$$

wobei

i_f ist die Inputnachricht von f,
o_f ist die Outputnachricht von f,
$\mathcal{P}_f^{\wp}$ ist eine endliche Menge von f-modulierenden Properties.

Die Nachrichten besitzen eine Beschreibung und eine Sammlung von erweiterten Ports. Jedem f wird ferner ein erweiterbares Tupel von Annotationen $\mathcal{A}_f$ zugeordnet. Derzeit besteht $\mathcal{A}_f$ aus den Elementen $\mathcal{A}_f = (\omega_f, desc_f, dom_f, class_f)$, wobei

ω_f ist der Name der Funktionalität f,
$desc_f$ ist eine inhaltliche, unstrukturierte Beschreibung von f,
dom_f ist die Anwendungsdomäne von f und
$class_f$ ist die Anwendungsklasse, zu der f gehört.

Die Anwendungsdomäne und -klasse einer Funktionalität f werden durch verschiedene Taxonomien beschrieben; Name und Beschreibung werden durch Worte über einem Alphabet gebildet. ■

Das Tupel der Annotationen $\mathcal{A}_f$ kann der Einschränkung des Suchraums bei der Identifizierung eines benötigten Service-Mediators dienen, beispielsweise wenn konkrete Komparatoren benötigt werden.

f spezifiziert die abstrakte Schnittstelle einer Mediatorfunktionalität. Mit $\mathfrak{f}$ wird die konkrete Realisierung einer Mediatorfunktionalität im Sinne einer Softwarekomponente bezeichnet, die der Schnittstellenspezifikation f genügt. $\mathfrak{f}_{|\cdot}$ bezeichnet eine konkrete konfigurierte Komponenteninstanz, während $f_{|\cdot}$ die Anpassung eines beliebigen Repräsentanten von f beschreibt.

Es sei nochmals daraufhin gewiesen, dass eine konkrete Realisierung $\mathfrak{f}$ einer Funktionalität f nicht-deterministisch sein kann und damit f keine Funktion im mathematischen Sinne ist (vgl. Abschnitt 4.2).

Ein *Service-Mediator* stellt nun eine Sammlung von Mediatorfunktionalitäten und den dazugehörigen Properties dar:

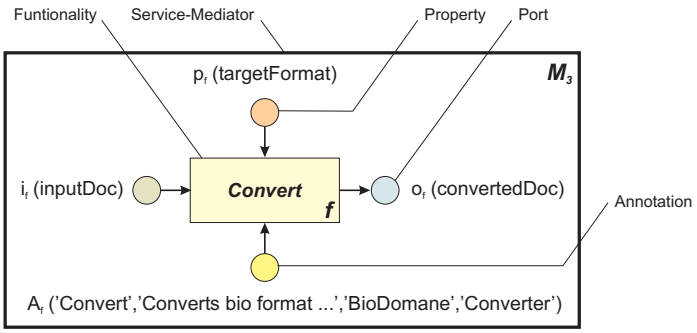

Abbildung 4.2: Schematische Darstellung der Mediatorfunktionalität f^{M_3} des Service-Mediators M_3 inklusive ihrer Properties.

Definition 4.3 (Service-Mediator)**.** Ein *Service-Mediator* M vereinigt in sich eine Menge von möglichst zusammengehörigen Mediatorfunktionalitäten und deren Properties, d.h. $M = (\mathcal{F}_M, \mathcal{P}_M^{\wp})$, wobei

$\mathcal{F}_M$ ist eine nichtleere Menge von Mediatorfunktionalitäten und
$\mathcal{P}_M^{\wp}$ ist eine Menge von Properties.

Dabei gilt für $\mathcal{P}_M^{\wp}$:

$$\mathcal{P}_M^{\wp} = \bigcup_{f \in \mathcal{F}_M} \mathcal{P}_f^{\wp}.$$

■

Zur Illustration der Konzepte betrachten wir die Mediatorfunktionalität f^{M_3} aus Beispiel 4.8:

Beispiel 4.11. In Abbildung 4.2 wird der Service-Mediator M_3 und die Mediatorfunktionalität f^{M_3} mit dem Namen *Convert* schematisch dargestellt. Eine Funktionalität besitzt genauso wie Services Input- und Outputnachrichten. Im Gegensatz zu den WSDL spezifizierten Services können die Konzepte, die den Nachrichten angehören, explizit angegeben werden. Die Konzepte entstammen einer entsprechenden Domänenontologie.

Ferner sind im Unterschied zu Services explizit Properties vorhanden. Hier ist beispielsweise die Property *targetFormat* angeben, über die das Resultatformat festgelegt wird. f^{M_3} besitzt eine Beschreibung und gehört der Anwendungsdomäne *Bio-Domäne* sowie der Klasse *Konvertierer* an.

Wie bereits oben angedeutet, besitzt der Begriff der Anpassbarkeit neben der Modifikation der Eigenschaftsfelder (Konfigurierung) weitere Facetten, zu denen auch die Komposition von Service-Mediatoren gehört. Auch hierdurch wird die Wiederverwendbarkeit von

Service-Mediatoren verbessert. Die Einheiten über die Service-Mediatoren komponiert werden können, sind die oben beschriebenen Mediatorfunktionalitäten. Es ist möglich, neue Mediatorfunktionalitäten zu schaffen, indem bereits vorhandene Mediatorfunktionalitäten adäquat miteinander in Beziehung gesetzt werden.

Das Resultat einer derartigen Komposition bildet eine *komplexe Mediatorfunktionalität*. Unzertrennbare Mediatorfunktionalitäten werden auch als *atomar* bezeichnet. Komplexe Funkionalitäten können dynamisch (*ad hoc*) zur Erreichung der Interoperabilität oder explizit (*benutzerdefiniert*) vor dem Einsatz innerhalb eines konkreten Workflows entstehen. Während im ersten Fall die resultierende Funktionalität durch den konkreten Workflow verlangt wird, entwirft im zweiten Fall der Komponentenentwickler eine neue Funktionalität, die auf vorhandenen Funktionalitäten aufbaut, ohne dabei auf einen konkreten Workflow Bezug zu nehmen. Soll dies ohne explizites Neuprogrammieren möglich sein, erfordert dies komplexe Kompositionsmöglichkeiten, wie beispielsweise Fallunterscheidungen hinsichtlich der Daten und Eigenschaften.

4.4.2 Spezifikation der Komposition

Die Spezifikation der Kompositionsmöglichkeit stellt den zweiten Teil der Spezifikation des Komponentenmodells für die Service-Mediatoren dar. Das dieser Arbeit zugrundeliegende Verständnis einer *Mediatorkomposition* ist durch die Definition von *Kompositionsgraphen* geprägt (Radetzki et al., 2004a). Die Kompositionsgraphen unterteilen sich in

- *Kontrollflussgraphen*, die den Ablauf des Prozesses, d.h. die Ausführungsreihenfolge der komponierten Funktionalitäten, steuern,
- *Datenflussgraphen*, die die Datenübertragung zwischen den Funktionalitäten innerhalb der Komposition festlegen, und
- *Eigenschaftsgraphen*, die Eigenschaftsfelder untereinander in Beziehung setzen und die Fixierung von Eigenschaftsfeldern innerhalb einer Komposition ermöglichen.

Die Definition der Kontrollflussgraphen und Datenflussgraphen sind an die Kontrollstrukturen von WSFL (Leymann, 2001) angelehnt. Im Gegensatz zu WSFL werden die Kontrollstrukturen exakt durch Petri-Netze spezifiziert, um Mehrdeutigkeiten zu vermeiden (siehe Anhang B.1). Ferner wird bewusst gefordert, in MPL lediglich einfache Kontrollprimitive zuzulassen. Beispielsweise sind keine Schleifen in einer Komposition erlaubt. Werden Schleifen benötigt, können diese direkt in einer atomaren Mediatorfunktionalität realisiert werden. Diese Designentscheidung soll zum einen den Umgang mit MPL erleichtern und zum anderen eine effiziente und konfliktfreiere Ausführung der Komposition ermöglichen.

Der graphbasierte Ansatz hat im Gegensatz zum blockbasierten Ansatz zudem den Vorteil der intuitiveren Nutzung. Der Umgang mit Kontrollstrukturen benötigt häufig Programmiererfahrung, über die nicht jeder Workflow- bzw. Mediatordesigner verfügt – ein Mediatordesigner ist nicht unbedingt der Entwickler eines atomaren Mediators.

Ein weiterer Unterschied zu WSFL ist die explizite Beschreibung von Konfigurationsfähigkeiten einer Komposition durch Eigenschaftsgraphen. Diese Fähigkeit wird derzeit von keiner der bekannten Kompositionssprachen im Servicebereich wie XLANG, WSFL oder BPEL4WS unterstützt.

Kontrollflussgraphen

Vereinfacht dargestellt, beschreibt ein Kontrollflussgraph die Reihenfolge, in der komponierte Mediatorfunktionalitäten ausgeführt werden, und die Bedingungen, unter denen die Ausführung geschieht. Ein Kontrollflussgraph legt den *Kontrollfluss* der komplexen Mediatorfunktionalität fest. In dem hier entwickelten Komponentenmodell werden drei essentielle Kontrollstrukturen unterstützt:

- die *Sequenz*, d.h. die hintereinander Ausführung von einzelnen Funktionalitäten,
- die *parallele Ausführung* unabhängiger Funktionalitäten, und
- die *Fallunterscheidung*, zur gezielten Ansteuerung einzelner Pfade im Kontrollflussgraphen.

Komplexere Strukturen wie Schleifen werden nicht zugelassen. Daher sind Kontrollflussgraphen immer azyklisch. Sie besitzen genau einen expliziten *Startpunkt* (*Source*), über den die Komposition aktiviert wird, und genau einen expliziten *Zielpunkt* (*Target*), an dem die Komposition terminiert. D.h. der Startpunkt und der Zielpunkt sind in einer Komposition eindeutig.

Die Terminierung stellt sich im Zusammenhang mit Fallunterscheidungen und der parallelen Ausführung jedoch als problematisch heraus, da es zum so genannten *Death-Path Problem* kommen kann (Leymann, 2001):

> **Beispiel 4.12.** Die Funktionalitäten A und C der Abbildung 4.3 werden direkt und parallel nach dem Startpunkt S ausgeführt. Die Ausführung von B ist hingegen von der Bedingung r abhängig, die ihrerseits von der Ausführung von A abhängig sei. (Man kann r als boolsche Variable interpretiert, die durch A auf *true* oder *false* gesetzt wird.) Ferner wird D nur dann ausgeführt, wenn p *oder* q gültig ist. p sei durch C und q durch B bestimmt. Werden r und p durch die Ausführungen der jeweiligen Funktionalitäten zu *false* evaluiert, dann kann nicht entschieden werden, ob D auszuführen ist, da B nicht ausgeführt wird. Es kommt zu einem *Death-Path* und die Komposition terminiert nicht, da der Zielpunkt T nicht erreicht wird.

Dieses Problem wird gelöst, indem zum einen gefordert wird, dass jeder bedingte Kontrollflussübergang einen logisch komplementären Kontrollflussübergang besitzen muss – einen so genannten *else-Pfad* – und durch die Einführung von *leeren Aktivitäten*, die keine Effekte

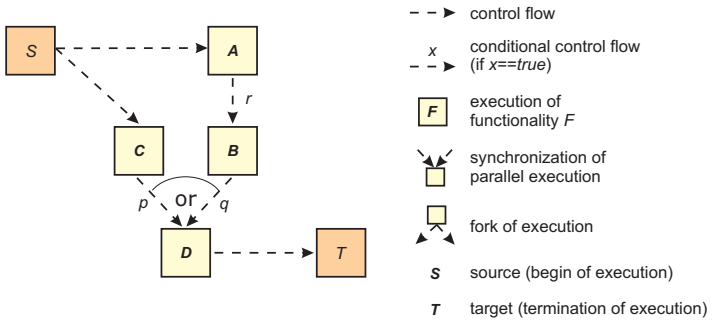

Abbildung 4.3: *Death-Path Problem* innerhalb einer Komposition. Die gestrichelten Linien beschreiben den Kontrollfluss der Komposition. Mit einem Buchstaben versehene Linien illustrieren bedingte Kontrollübergänge.

auf die Komposition haben, sondern lediglich die Kontrolle weitergeben. Auf diese Weise existiert immer mindestens ein ausführbarer Pfad von Startpunkt bis zum Zielpunkt.

Um einerseits nicht mehrere *baumelnde* Enden nach einer Aufsplittung des Kontrollflusses zu haben, und um andererseits den Zielpunkt bei der Ausführung des Kontrollflussgraphen nicht mehrfach zu erreicht, wird für Kontrollflussgraphen gefordert, dass parallele Ausführungen und Fallunterscheidungen vor dem Zielpunkt synchronisiert werden. Man spricht in diesem Zusammenhang auch von *AND-* und *OR-Synchronisationspunkten* (bzw. *AND-/OR-Synchronisationsaktivitäten*) (van der Aalst et al., 2003).

Durch diese Randbedingungen wird garantiert, dass es mindestens einen ausführbaren Pfad vom Startpunkt bis zum Zielpunkt gibt und das der Zielpunkt auch bei paralleler Ausführung exakt einmal erreicht wird.

Im Folgenden werden die Elemente des Kontrollflussgraphen im Detail spezifiziert. Die Knoten dieser Graphen werden im folgenden *Aktivitäten* genannt. Sie repräsentieren die ausführbaren Einheiten und dienen der Synchronisation und Steuerung. Genauer gesagt:

Definition 4.4 (Aktivität)**.** Die Menge der *Aktivitäten* A_c kann aus folgenden Elemente bestehen:

- Aktivitäten $a[f]$, die die Ausführung einer konkreten Komponenteninstanz $\mathfrak{f}_{|}$. anstößt,
- leere Aktivitäten $\hat{e}$, die sofort nach der Aktivierung enden, ohne ein konkretes Verhalten ausgeführt zu haben,
- AND- und OR-Synchronisationsaktivitäten, die parallele Ausführungen und Fallunterscheidungen synchronisieren (die AND-Synchronisation wartet dabei auf *alle* eingehenden Aktivitäten, während die OR-Synchronisation nur auf *mögliche* eingehende Aktivität wartet) und

- eindeutige Aktivitäten für den Startpunkt S und den Zielpunkt T des Kontrollflussgraphen. Es gilt $\forall A_c : \mathrm{S}, \mathrm{T} \in A_c$. ■

Die Kanten des Kontrollflussgraphen spezifizieren nun die Reihenfolge, in der die Aktivitäten abgearbeitet werden. Hierbei werden einfache Kanten und bedingte Kanten unterschieden:

Definition 4.5 (Aktivierungsübergänge)**.** Sei V_c eine endliche Menge von boolschen Variablen und sei $\top$ die „undefinierte Variable“, die immer den Wert *true* besitzt. Dann sei $V_c^\top = V_c \cup \{\top\}$. Sei ferner A_c die Menge der Aktivitäten und sei $E_c = A_c \times V_c^\top \times A_c$ die Menge der *Aktivierungsübergänge* (*control links*). O.B.d.A. sei $(a_1, v, a_2) \in E_c$, dann werden in Abhängigkeit von v folgende Fälle unterschieden:

(a) $v = \top$, dann wird nach Beendigung der Aktivität a_1 sofort a_2 aktiviert und ausgeführt (*einfacher Aktivierungsübergang*), und

(b) $v \neq \top$, dann erfolgt die Aktivierung von a_2 nach Beendigung der Aktivität a_1 nur, wenn v den Wert *true* angenommen hat (*bedingter Aktivierungsübergang*). ■

Nachdem die Aktivierungsübergänge allgemein definiert wurden, werden nun die oben aufgeführten Bedingungen an die Aktivierungsübergänge aufgestellt:

Definition 4.6 (Bedingungen an Aktivierungsübergänge)**.** Sei A_c die Menge der Aktivitäten, $V_c^\top$ die Menge der Variablen und E_c die Menge der Aktivierungsübergänge, dann gilt:

- Es gibt mindestens einen Aktivierungsübergang, der von der Aktivität $\mathrm{S} \in A_c$ ausgeht, d.h. $\exists a \in A_c, v \in V_c^\top : (\mathrm{S}, v, a) \in E_c$.
- Es gibt genau einen Aktivierungsübergang, der die Aktivität $\mathrm{T} \in A_c$ aktiviert, d.h. $\exists a \in A_c, v \in V_c^\top : (a, v, \mathrm{T}) \in E_c$. $\forall a' \in A_c, v' \in V_c^\top$: wenn $(a', v', \mathrm{T}) \in E_c$, dann $a' = a$ und $v' = v$.
- Es gibt keine Aktivierung in S oder aus T, d.h. $\forall a, a' \in A_c, v, v' \in V_c^\top : (a, v, \mathrm{S}) \notin E_c$ und $(\mathrm{T}, v', a') \notin E_c$.
- Für alle Aktivitäten $a[f] \in A_c$ und $\hat{e} \in A_c$ gibt es mindestens eine ausgehende Kante und genau eine eingehende Kante in E_c. Für alle AND- und OR-Synchronisationsaktivitäten gibt es mindestens eine eingehende Kante und genau eine ausgehende Kante in E_c. AND-Synchronisationsaktivitäten vereinigen parallel auszuführende Aktivitäten, während OR-Synchronisationsaktivitäten Pfade aus bedingten Aktivitätsübergängen zusammenführen.
- Es existiert $\forall a \in A_c$ ein Pfad in E_c, der mit S beginnt und mit T endet, auf dem a liegt.

- Für bedingte Aktivierungsübergänge wird gefordert, wenn $(a_0, v, a_1) \in E_c$, $v \neq \top$, dann auch $(a_0, \neg v, a_2) \in E_c$, d.h. es gibt immer einen *else*-Teil. Gehen mehrere bedingte Aktivierungsübergänge aus a_0 hervor und lassen sich diese bündeln, d.h. $(a_0, v_i, a_i) \in E_c$, $v_i \neq \top$, $1 \leqslant i \leqslant n$, dann gilt auch $(a_0, \neg(\bigvee_i v_i), a_{n+1}) \in E_c$, d.h. es existiert ein *otherwise*-Teil. ■

Zusammenfassend können Kontrollflussgraphen als Bestandteil der Mediatorkomposition definieren werden:

Definition 4.7 (Kontrollflussgraph)**.** Sei A_c die Menge der Aktivitäten, $V_c^\top$ die Menge der Variablen und E_c die Menge der Aktivierungsübergänge, wobei E_c den Bedingungen der Definition 4.6 genügt, dann beschreibt der azyklische Graph $G_c = (A_c, E_c, V_c)$ den *Kontrollflussgraphen* einer Mediatorkomposition. ■

> **Beispiel 4.13.** Der Graph aus Abbildung 4.3 genügt den Definitionen 4.4 und 4.5, jedoch nicht der Definition 4.6. Daher ist dieser Graph kein Kontrollflussgraph.

Datenflussgraphen

Der Kontrollflussgraph sagt lediglich etwas über die Aktivierungsreihenfolge aus. Über die Belegung der Variablen und den Datentransfer zwischen den einzelnen Aktivitäten wird keine Aussage getroffen. Diese Aspekte werden in den Datenflussgraphen und Eigenschaftsgraphen behandelt.

Der Datenflussgraph legt den *Datentransfer* innerhalb einer Mediatorkomposition fest, d.h. er spezifiziert, wie die datenproduzierenden und datenkonsumierenden Elemente der Mediatorfunktionalitäten verknüpft werden können. Drei mögliche Datentransfers sind in dem Komponentenmodell der Service-Mediatoren gültig:

- Verknüpfung der Outputports einer Funktionalität mit den Inputports einer im Kontrollfluss nachfolgenden Funktionalität (*Inputbelegung durch Outputs*).
- Verknüpfung des Outputports eines Komparators (spezielle Funktionalität) mit einer boolschen Variablen des Kontrollflussgraphen (*Variablenbelegung*). Hierdurch wird der Kontrollfluss modelliert (siehe Fallunterscheidungen). Ein Komparator liefert als Ergebnis immer einen boolschen Wert.
- (Vor-)belegung von optionalen oder für eine Komposition fixen Inputports einer Funktionalität durch konkrete Werte (*Inputbelegung durch Werte*).

Sei f_c die komplexe Funktionalität, die durch die Komposition entsteht. Dann beinhaltet der erste Fall auch den Aspekt, dass die Daten, die an die komplexe Funktionalität angelegt werden (Inputports von f_c) an mögliche Inputports der komponierten Funktionalitäten weitergeleitet werden können. Entsprechendes gilt für die Outputports von f_c.

Beschreiben wir nun die entsprechenden Mengen:

Definition 4.8 (Datenknoten)**.** Sei $G_c = (A_c, E_c, V_c)$ ein Kontrollflussgraph der komplexen Funktionalität $f_c : (i_{f_c}, \mathcal{P}^{\wp}_{f_c}) \rightarrow (o_{f_c})$, dann gilt für folgende Mengen:

$$\begin{aligned}
P_{in} &= \{p_{i_f} | p_{i_f} \in \mathcal{P}_{i_f}, \mathcal{P}_{i_f} \text{ Inputports von } f \text{ und } a[f] \in A_c\} \cup \\
&\quad \{p_{o_{f_c}} | p_{o_{f_c}} \in \mathcal{P}_{o_{f_c}}, \mathcal{P}_{o_{f_c}} \text{ Outputports von } f_c\}, \\
P_{out} &= \{p_{o_f} | p_{o_f} \in \mathcal{P}_{o_f}, \mathcal{P}_{o_f} \text{ Outputports von } f \text{ und } a[f] \in A_c\} \cup \\
&\quad \{p_{i_{f_c}} | p_{i_{f_c}} \in \mathcal{P}_{i_{f_c}}, \mathcal{P}_{i_{f_c}} \text{ Inputports von } f_c\}, \\
P^{comp}_{out} &\subseteq P_{out}, \forall p_{o_f} \in P^{comp}_{out} : \mathit{class}_f = \mathit{comparator}, \text{ und} \\
W &\quad \text{ist eine Menge von Werten.}
\end{aligned}$$

Die Vereinigung $A_d = P_{in} \cup P_{out} \cup V_c \cup W$ definiert die Menge der *Datenknoten*. ■

P_{in} und P_{out} beinhalten die Inputports bzw. Outputports der verschiedenen Funktionalitäten. P^{comp}_{out} ist eine Teilmenge von P_{out}, die nur die Outputports der Komparatoren beinhaltet, über die der Kontrollfluss modelliert wird. W ist eine Menge von Werten, die für die Belegung der Inputports genutzt wird.

Nun gilt es die Verknüpfung zwischen den Datenknoten herzustellen, über den die drei möglichen Datentransfers von Seite 71 geregelt werden:

Definition 4.9 (Datenübertragungen)**.** Seien die Datenknoten aus Definition 4.8 gegeben. Dann bezeichnet die Menge E_d mit

$$\begin{aligned}
E_d &= P_{out} \times P_{in} && (\textit{Inputbelegung durch Outputs}) \\
&\cup\ P^{comp}_{out} \times V_c && (\textit{Variablenbelegung}) \\
&\cup\ W \times P_{in} && (\textit{Inputbelegung durch Werte})
\end{aligned}$$

die Menge der möglichen *Datenübertragungen* (*data links*). ■

Nicht alle über E_d beschreibbaren Datenübertragungen sind sinnvoll. Daher werden die möglichen Datenübertragungen weiter eingeschränkt:

Definition 4.10 (Gültige Datenübertragung)**.** Seien die Datenknoten aus Definition 4.8 gegeben und sei E_d eine Menge von Datenübertragungen. Für eine *gültige* Datenübertragung $(p_{f_1}, p_{f_2}) \in P_{out} \times P_{in} \subseteq E_d$ muss gelten, dass es einen Pfad in E_c von $a[f_1]$ (bzw. S) nach $a[f_2]$ (bzw. T) gibt. S bzw. T gelten im Falle das $f_1 = f_c$ bzw. $f_2 = f_c$. ■

Die Bedingung besagt, dass die Aktivität $a[f_1]$ (bzw. S) einer datenproduzierenden Funktionalität f_1 (bzw. f_c) auf dem gleichen Pfad von S nach T liegt wie die Aktivität $a[f_2]$ (bzw. T) einer datenkonsumierenden Funktionalität f_2 (bzw. f_c) und das f_1 vor f_2 ausgeführt und beendet wird. Dies beinhaltet auch das $a[f_1] \neq a[f_2]$ ist.

Abschließend muss die Korrektheit bei der Datenübertragung gesichert werden, d.h. dass die übergebenen Daten an den entsprechenden Inputports korrekt verarbeitet werden können:

Definition 4.11 (Korrekte Datenübertragung)**.** Seien die Datenknoten aus Definition 4.8 gegeben und sei E_d eine Menge von Datenübertragungen. Eine Datenübertragung (p_1, p_2) bzw. (w, p) ist *korrekt* im Sinne der Datentypen und Konzepte, wenn

(a) $(p_1, p_2) \in P_{out} \times P_{in} \subseteq E_d$ gilt:

(1) $d_{p_1} \preceq d_{p_2}$ und
(2) $\exists c_{p_1} \in C_{p_1}, c_{p_2} \in C_{p_2} : c_{p_1} \sqsubseteq c_{p_2}$ oder

(b) $(w, p) \in W \times P_{in} \subseteq E_d$ und w vom Datentyp d_w ist und den Konzepten C_w angehört, gilt:

(1) $d_w \preceq d_p$ und
(2) $\exists c_w \in C_w, c_p \in C_p : c_w \sqsubseteq c_w$. ■

Die beiden obigen Fälle korrespondieren zu der Definition der Service-Interoperabilität (Def. 3.3 auf Seite 40). Auch hier wird gefordert, dass Semantik und Syntax übereinstimmen müssen.

Über den Datenflussgraph wird nun der Datentransfer einer Mediatorkomposition geregelt:

Definition 4.12 (Datenflussgraph)**.** Seien die Datenknoten aus Definition 4.8 gegeben und sei E_d eine Menge von gültigen und korrekten Datenübertragungen, dann bezeichnet der Graph $G_d = (A_d, E_d)$ den *Datenflussgraphen* der Mediatorkomposition. ■

Eigenschaftsfelder (*Properties*) komplexer Funktionalitäten

In den beiden vorangegangenen Abschnitten wurden Kontroll- und Datenfluss einer komplexen Funktionalität spezifiziert. Nun werden die Eigenschaftsfelder komplexer, sowie komponierter und teilweise angepasster Funktionalitäten behandelt.

Eigenschaftsfelder modellieren das Verhalten einer Mediatorfunktionalität. In einer Mediatorkomposition können sich Eigenschaftsfelder sowohl auf die komplexe Funktionalität selbst als auch auf die komponierten Funktionalitäten beziehen. Es werden in dieser Arbeit drei Varianten unterschieden:

- Die Modellierung des Kontrollflusses einer Komposition durch Eigenschaftsfelder und Komparatoren, die die Fallunterscheidungen steuern. Das heißt, die Werte der Eigenschaftsfelder dienen als Input für Komparatoren (*Inputbelegung durch Eigenschaftsfelder*). Somit können nicht nur Outputdaten von komponierten Funktionalitäten den Kontrollfluss verändern.
- Variable Konfigurierung komponierter Funktionalitäten durch Eigenschaftsfelder der komplexen Funktionalität (*Variable Konfigurierung*).

- Feste Konfigurierung komponierter Funktionalitäten durch konkrete Werte, wenn dies die konkrete Komposition erfordert (*Feste Konfigurierung*). D.h. die Komposition erfordert zur Durchführung ganz konkret angepasste komponierte Funktionalitäten.

Die Unterstützung der drei Varianten geschieht mit Hilfe so genannter Eigenschaftsgraphen. Diese stützen sich auf folgende Knoten:

Definition 4.13. Seien $G_c = (A_c, E_c, V_c)$ Kontrollflussgraph und $G_d = (A_d, E_d)$ Datenflussgraph einer komplexen Funktionalität $f_c : (i_{f_c}, \mathcal{P}^{\wp}_{f_c}) \rightarrow (o_{f_c})$.

$$\begin{aligned} P^{comp}_{in} &\subseteq P_{in}, \forall p_{i_f} \in P^{comp}_{in} : class_f = comparator, \\ \mathcal{P}^{\wp} &= \bigcup_{f,a[f]\in A_c} \mathcal{P}^{\wp}_{f}. \end{aligned}$$

Sei $A_{\wp} = \mathcal{P}^{\wp}_{f_c} \cup \mathcal{P}^{\wp} \cup P^{comp}_{in} \cup W$ die Zusammenfassung der einzelnen Mengen. ■

$\mathcal{P}^{\wp}$ ist die Menge der Properties der komponierten Funktionalitäten, während P^{comp}_{in} die Inputports der Komparatoren vereinigt.

Über diese Knoten lassen sich nun die drei möglichen Zuordnungen für Eigenschaftsfelder einer Mediatorkomposition definieren:

Definition 4.14 (Property-Verbindung)**.** Seien die Elemente aus Definition 4.13 geben. Dann bezeichnet $E_{\wp}$ mit

$$\begin{aligned} E_{\wp} &= \mathcal{P}^{\wp}_{f_c} \times \mathcal{P}^{\wp} && (\textit{Variable Konfigurierung}) \\ &\cup \mathcal{P}^{\wp}_{f_c} \times P^{comp}_{in} && (\textit{Inputbelegung durch Eigenschaftsfelder}) \\ &\cup W \times \mathcal{P}^{\wp} && (\textit{Feste Konfigurierung}) \end{aligned}$$

die Menge der *Property-Verbindungen* der Mediatorkomposition. ■

Die Property-Verbindung $(pr_1, pr_2) \in \mathcal{P}^{\wp}_{f_c} \times \mathcal{P}^{\wp} \subseteq E_{\wp}$ beschreibt, dass eine Property pr_2 einer komponierten Funktionalität durch eine Property pr_1 der komplexen Funktionalität bestimmt wird. Die Konfigurierung ist variabel, da erst bei der Anpassung der komplexen Funktionalität die interne Funktionalität ebenfalls angepasst wird.

Im Gegensatz dazu spezifiziert eine Property-Verbindung der Form $(w, pr_f) \in W \times \mathcal{P}^{\wp}_{f} \subseteq E_{\wp}$, dass die Property pr_f einer komponierten Funktionalität f den Wert w annimmt ($f_{|w}$). Diese Konfigurierung ist fest und nicht von außen änderbar.

Eigenschaftsfelder können zudem das Verhalten der komplexen Funktionalität durch Property-Verbindungen $(pr, p_{in}) \in \mathcal{P}^{\wp}_{f_c} \times P^{comp}_{in} \subseteq E_{\wp}$ modellieren. (pr, p_{in}) beschreibt, dass der Wert einer Property einem konkreten Inputport eines Komparators übergeben wird. Dieser kann den Wert analysieren und entsprechend den Kontrollfluss über Fallunterscheidungen verändern.

Die Definition 4.11 der Korrektheit von Datenübertragungen wird analog auf Property-Verbindungen übertragen. In diesem Zusammenhang wird von *korrekten* Property-Verbindungen gesprochen:

Definition 4.15 (Korrekte Property-Verbindungen)**.** Für die *Korrektheit* der Property-Verbindungen muss gelten, dass $\forall(pr_1, pr_2) \in E_\wp$ und pr_1 vom Datentyp d_{pr_1} ist und den Konzepten C_{pr_1} angehört[6], dass

(1) $d_{pr_1} \preceq d_{pr_2}$ und
(2) $\exists c_{pr_1} \in C_{pr_1}, c_{pr_2} \in C_{pr_2} : c_{pr_1} \sqsubseteq c_{pr_2}$.

■

Ausgehend von korrekten Property-Verbindungen wird nun der Eigenschaftsgraph einer Mediatorkomposition definiert:

Definition 4.16 (Eigenschaftsgraph)**.** Seien die Elemente aus Definition 4.13 geben und sei $E_\wp$ eine Menge von korrekten Property-Verbindungen, dann definiert der Graph $G_\wp = (A_\wp, E_\wp)$ den *Eigenschaftsgraphen* (*Property-Graphen*) der Mediatorkomposition. ■

In Kapitel 5.4 wird ein konkretes Beispiel für eine Mediatorkomposition angegeben.

4.5 Die Mediatorprofilsprache

Die *Mediatorprofilsprache* (*Mediator Profile Language* MPL) ermöglicht, die Fähigkeiten eines Service-Mediators entsprechend der in Abschnitt 4.3 diskutierten Anforderungen beschreiben und explorieren zu können (Introspektion). MPL basiert auf der standardisierten Ontologiebeschreibungssprache OWL. OWL stellt Konzepte zur Verfügung, die sich zur Definition eines entsprechenden Schemas ausnutzen lassen. Hierzu zählen Klassen, Objekt- und Datentypbeziehungen (siehe auch Kapitel 3.1).

Innerhalb MPL wird prinzipiell die Schnittstelle des Mediators, die sowohl semantisch als auch syntaktisch ausgezeichnet werden kann, von der Anpassung, die sich aus der Komposition und dem Setzen von Eigenschaftsfeldern ergibt, unterschieden. Die folgenden Abschnitte beschränken sich auf die Darstellung der entscheidenden Konzepte inklusive ihrer Beziehungen, da eine vollständige Beschreibung einen zu großen Umfang einnehmen würde. Die vollständige Schemadefinition findet sich im Anhang B.2.

4.5.1 Schnittstelle eines Service-Mediators

Die Kernelemente der Schnittstelle eines Service-Mediators sind durch die Definitionen 3.2, 4.1, 4.2 und 4.3 gegeben. Diese werden auf die entsprechenden Konzepte in MPL abgebildet.

[6] d_{pr_1} und C_{pr_1} müssen explizit aufgeführt werden, da pr_1 aus W sein kann.

Mediatorprofil

Im Zentrum eines Mediatorprofils steht das entsprechende Konzept `Mediator Profile`. Dieses Konzept beinhaltet generelle Informationen über die Mediatorkomponente. Hierzu zählen der Mediatorname (`mediatorName`), der Ort der WSDL-Beschreibung (`wsdlLocation`), Informationen über den Entwickler bzw. die Entwicklerfirma (`developerInformation`), sowie die thematische Klassifizierung des Mediators (`mediatorCategory`), beispielsweise ein geo- oder biologischer Anwendungskontext. Die Klassifizierung muss sich hierbei an einer gegebenen Taxonomie orientieren (wie NAICS[7]). Die Angabe der URI des WSDL-Dokumentes ist notwendig, um die Realisierung des Mediators zur Laufzeit bestimmen und ausführen zu können. Abbildung 4.4 illustriert diese Konzepte inklusive ihrer Beziehungen zueinander[8].

Zudem stellt das Konzept `Mediator Profile` Informationen über die Fähigkeiten des Mediators bereit. Die Fähigkeiten werden, wie bereits diskutiert, in zwei Klassen unterteilt, Properties und Funktionalitäten. Diese können über die Beziehungen `mediatorProperty` bzw. `mediatorFunctionality` zugegriffen werden (vgl. Definition 4.3).

In Zukunft können in dem Mediatorprofil auch Informationen über die Qualität des Mediators angegeben werden, wie beispielsweise der Preis für die Benutzung, die Genauigkeit der Transformation, Geschwindigkeit der Berechnung, Ausfallsicherheit und vieles mehr. Viele dieser Aspekte werden heutzutage in so genannten *Service Level Agreements* (SLA) niedergeschrieben[9].

Funktionalitäten

Die Funktionalitäten eines Service-Mediators werden durch das Konzept `Mediator Functionality` repräsentiert. Jede Funktionalität wird durch eine Menge von Ports spezifiziert, die über die Beziehungen `inputPort` und `outputPort` zugeordnet werden. Wie aus Abbildung 4.5 ersichtlich unterstützt das Konzept `Mediator Functionality` ferner die geforderten Annotationen einer Funktionalität der Definition 4.2. Dem Aspekt der komplexen Funktionalität wird durch die Konzepte `Complex Functionality` und `Functionality Activity` Rechnung getragen, die im Abschnitt 4.5.2 näher betrachtet werden.

[7] North American Industry Classification System (NAICS), `www.census.gov/epcd/www/naics.html`

[8] Die Kardinalitäten der Beziehungen sind aus Übersichtlichkeitsgründen in den verschiedenen Grafiken nicht eingezeichnet. Diese können im Anhang B.2 nachgeschlagen werden.

[9] Ein Service Level Agreement ist ein (messbarer) Vertrag zwischen einem Serviceanbieter und seinem Kunden, der die Qualität der Dienste, die der Serviceanbieter liefert, spezifiziert.

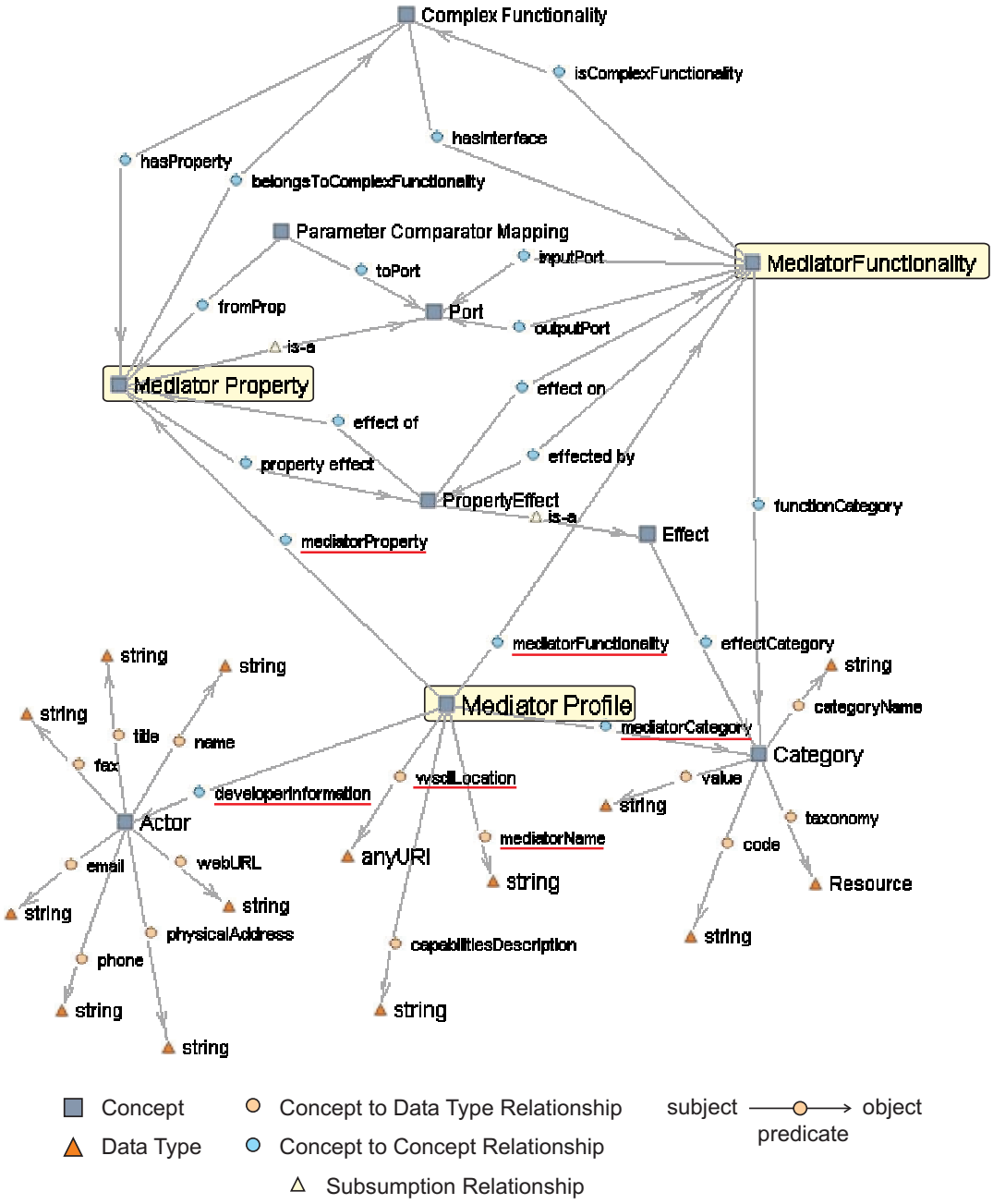

Abbildung 4.4: Kernkonzepte des Mediatorprofils.

Eigenschaftsfelder

Das Konzept `Mediator Property` beschreibt die Klasse der Eigenschaftsfelder (vgl. Definition 4.1). Dieses Konzept ist mit dem Konzept der Mediatorfunktionalitäten bidirektional verbunden, um eine Zuordnung zwischen diesen Konzepten zu ermöglichen (vgl. Definition 4.2). Während die Funktionalitäten durch eine Menge von Ports genauer spezifiziert werden, gehören die Properties hingegen der Klasse der Ports an. Genauer gesagt, sie bilden einen Untertyp der Klasse `Port` über eine entsprechende *is-a* Beziehung.

Ports, Parameter und boolsche Variablen

Das Konzept `Port` bildet die Definition 3.2 der erweiterten Ports ab. Ports sind ihrerseits vom Konzept `Parameter` abgeleitet, welches einen Namen (`parameterName`), eine Beschreibung (`parameterDescription`) und einen Datentypen besitzt. Letzterer wird durch die Beziehung `rangeOf` spezifiziert. Ports besitzen ferner die Möglichkeit der *semantischen* Annotation, die mittels der Relation `hasConcept` kodiert wird. Diese verweist auf das Konzept `Concept`, welches seinerseits auf Konzepte einer Domänenontologie verweist (`conceptRef`). Da unter Umständen ein Konzept einer Ontologie nicht exakt den semantischen Typ des Ports beschreibt, sind hier feinere Abstufungen möglich (Radetzki und Cremers, 2004)[10].

Wie in der Spezifikation des Komponentenmodells in Kapitel 4.4 beschrieben, werden Fallunterscheidungen im Kontrollfluss durch bedingte Aktivierungsübergänge definiert (Definition 4.5). Diese beziehen sich auf boolsche Variablen. Aus diesem Grund spezifiziert MPL das Konzept `Variable`. Diese sind als Unterklasse der Parameter definiert. Im Gegensatz zu Ports werden diese jedoch nicht semantisch näher beschrieben, da sie zur Zeit lediglich zur Fallunterscheidung eingesetzt werden. Abbildung 4.6 stellt die Konzepte zusammengefasst dar.

Anwendung des Mediatorprofils

Die in dem Profil kodierte Information kann nicht nur genutzt werden um den Introspektionsmechanismus zu realisieren, sondern sie wird auch benötigt um geeignete Mediatoren zu identifizieren. Des weiteren kann sie aus softwaretechnischer Hinsicht verwendet werden, um darauf aufbauend den `PortType` des WSDL-Dokumentes bzw. Proxy-Klassen für bestimmte Programmiersprachen wie Java zu generieren. Soll umgekehrt ein bereits realisierter Service als Mediator zur Verfügung gestellt werden, lässt sich das Profil ebenfalls bezüglich der syntaktischen Ausprägung vollständig und bezüglich der semantischen Ausprägung teilweise automatisiert herleiten (vgl. Kapitel 5.2). Gewöhnlich nutzen Mediatordesigner visuelle Werkzeuge zur Erstellung eines Mediatorprofils, d.h. sie kommen mit MPL nicht direkt in Berührung. Dies gilt auch für die Entwicklung von Mediatorkompositionen. Kapitel 6.3 geht auf diese Aspekte näher ein.

[10] Hier werden zur Zeit vier mögliche Grade unterstützt: 'exactly'; 'less exactly'; 'weak'; 'very weak'.

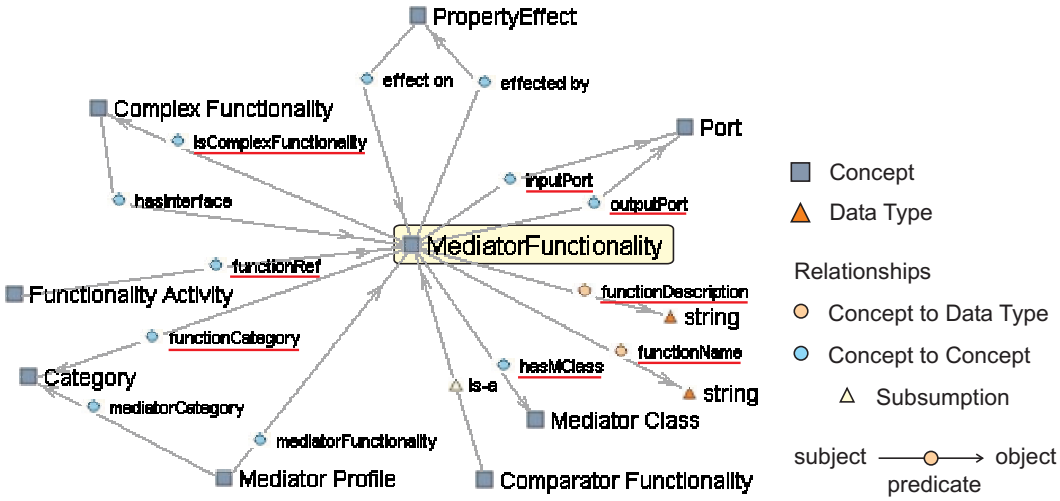

Abbildung 4.5: Annotation der Mediatorfunktionalität und Zusammenhang mit komplexen Funktionalitäten.

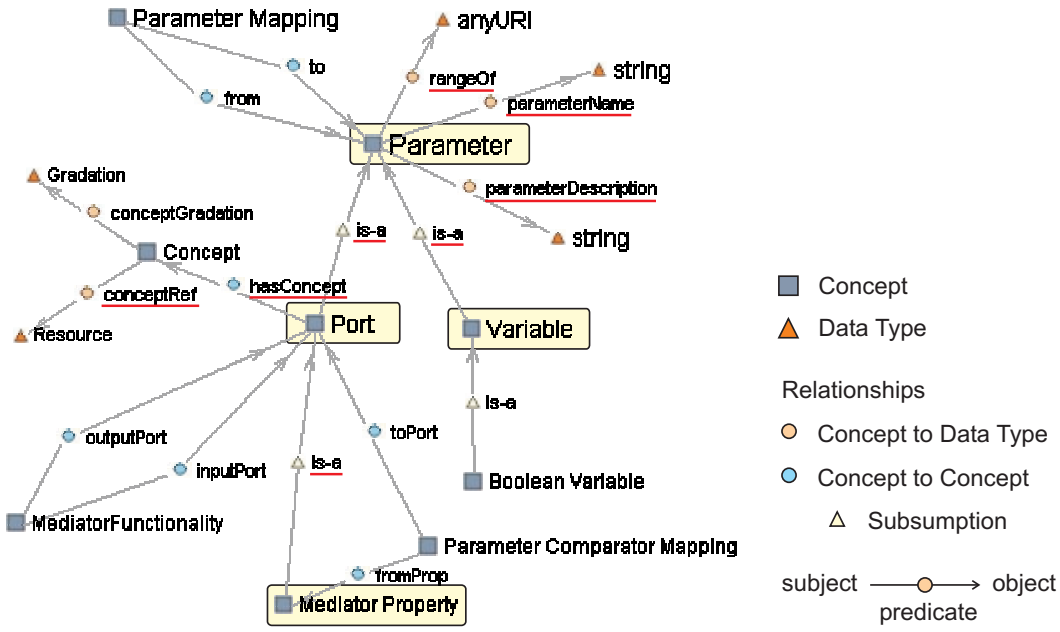

Abbildung 4.6: Ports, Parameter und Variablen in MPL.

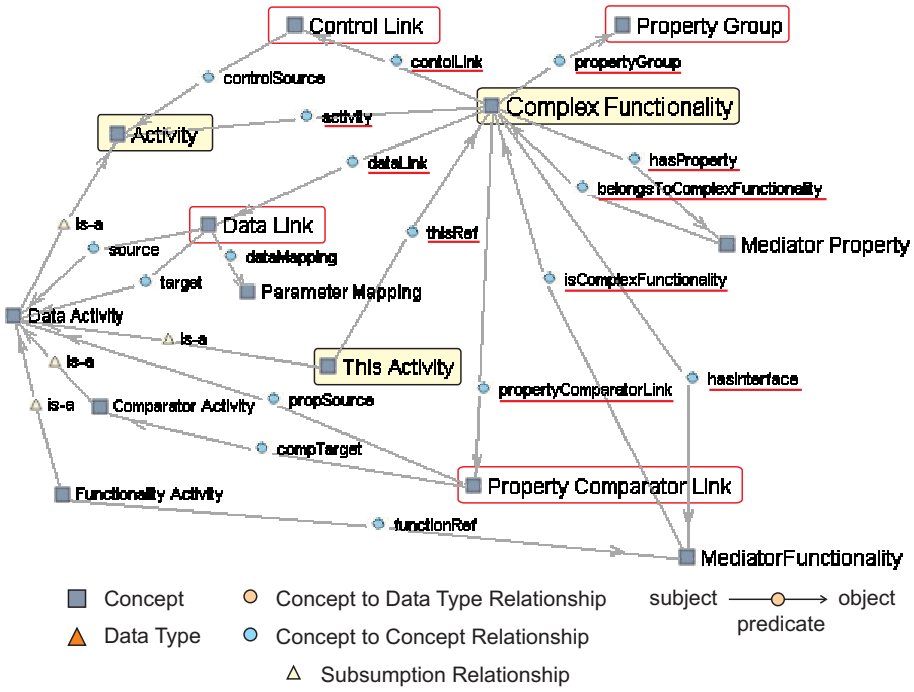

Abbildung 4.7: Konzepte zur Beschreibung einer komplexen Funktionalität.

4.5.2 Komposition und Konfigurierung komplexer Funktionalitäten

Die Kompositions- und Konfigurierungsmöglichkeiten einer komplexen Funktionalität sind im Komponentenmodell der Service-Mediatoren niedergeschrieben und durch Kontrollfluss-, Datenfluss- und Eigenschaftsgraphen spezifiziert (Definitionen 4.4 bis 4.16). Die Definitionsmöglichkeiten dieser Graphen sind entsprechend in MPL wiederzufinden.

Komplexe Funktionalitäten

Komplexe Funktionalitäten werden über Graphenstrukturen gebildet. Abbildung 4.7 stellt hierzu die entsprechenden Konzepte dar. Im Zentrum steht das Konzept `Complex Functionality`, welches eine Menge von Aktivitäten (`Activity`) beinhaltet inklusive einer *this*-Referenz (`This Activity`), welche die komplexe Funktionalität in den Graphen repräsentiert. Die this-Referenz steht beispielsweise für den Start- und Endpunkt des Kontrollflussgraphen (vgl. Definition 4.4).

Kontroll- und Datenflussgraphen werden durch die Konzepte `Control Link` bzw. `Data Link` abgebildet. Die Behandlung der Property-Verbindungen findet mittels der Konzepte `Property Comparator Link` sowie `Property Group` statt.

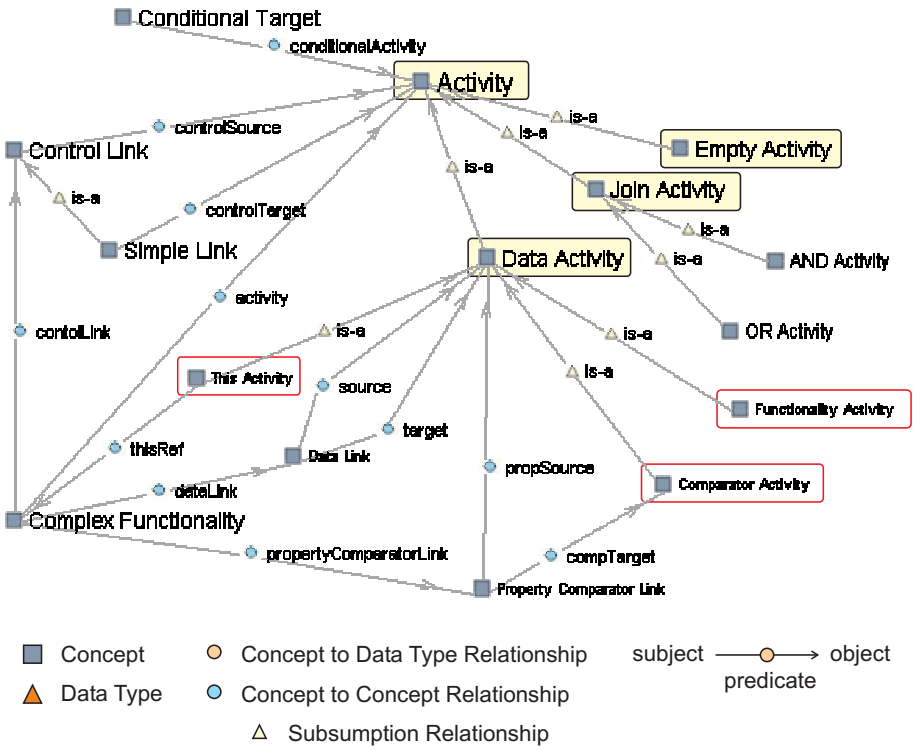

Abbildung 4.8: Klassifikation der verschiedenen Aktivitäten.

Die Schnittstelle einer komplexen Funktionalität wird genau wie die einer atomaren durch eine Mediator Funktionalität beschrieben (`hasInterface` und `isComplexFunctionality` Relationen). Gleiches gilt für die Eigenschaftsfelder der komplexen Funktionalität. Diese werden durch die Beziehungen `hasProperty` und `belongsToComplexFunctionality` ausgedrückt.

Aktivitäten

Aus der Definition 4.4 wird deutlich, dass neben dem Konzept der `This Activity` weitere Aktivitätsklassen existieren. Hierzu zählen beispielsweise die Synchronisationsaktivitäten (`Join Activity`) und die leere Aktivität (`Empty Activity`). Neu in der Klassifikation der Aktivitäten sind die datenverarbeitenden Aktivitäten, die unter dem Konzept `Data Activity` zusammengefasst werden. Hierzu zählen die bereits erwähnte `This Activity`, sowie allgemeine Funktionalitätenaktivitäten (`Functionality Activity`) und Komparatoraktivitäten (`Comparator Activity`). Aufgrund der unterschiedlichen Semantik werden letztere Aktivitäten getrennt behandelt. Die vollständige Klassifikation wird in Abbildung 4.8 beschrieben.

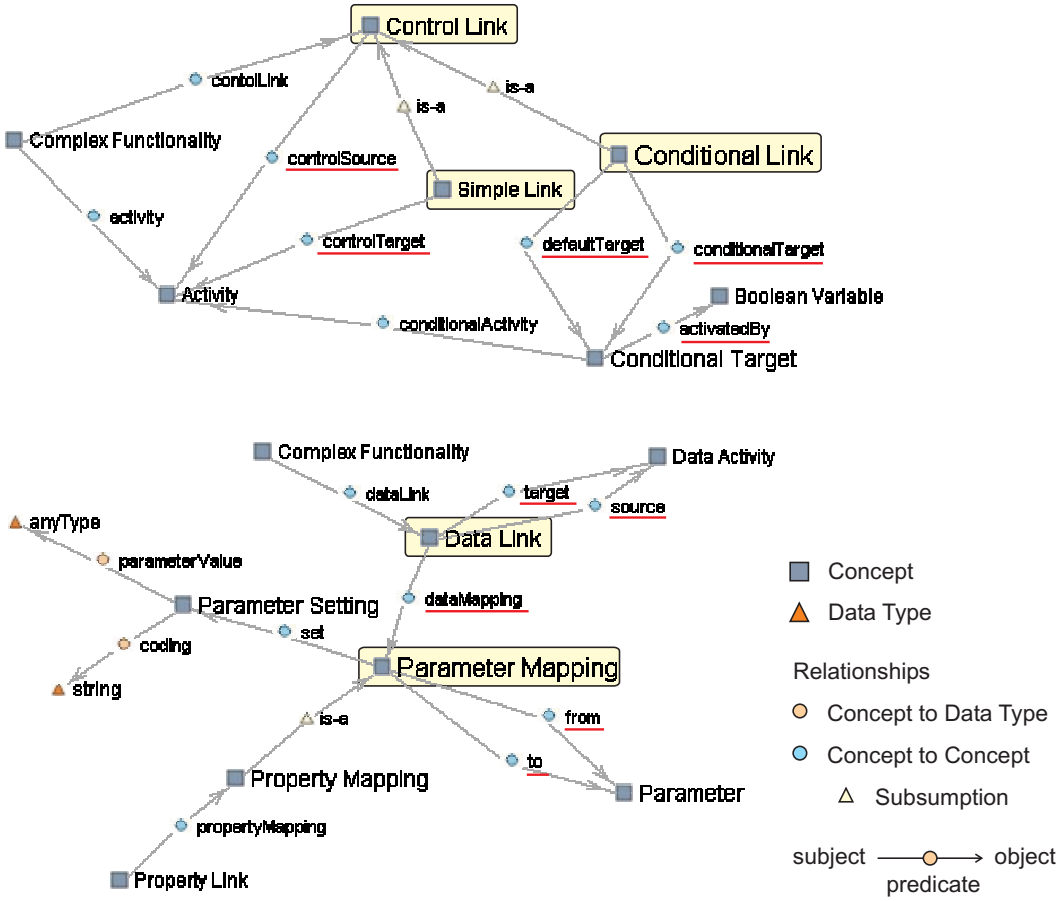

Abbildung 4.9: Konzepte zur Beschreibung des Kontrollflusses (oben) und des Datenflusses (unten).

Aktivierungsübergänge

Kontrollflussgraphen werden über einzelne Kontrollkanten definiert und spiegeln sich im Konzept `Control Link` wieder. Dieses Konzept entspricht den in Definition 4.5 beschriebenen Aktivierungsübergängen. Wie dort definiert, werden einfache, d.h. nicht-konditionale, Kanten (`Simple Link`) und konditionale Kanten (`Conditional Link`) unterschieden. Letztere besitzen immer eine *default*-Senke (`defaultTarget` Relation) und mindestens eine konditionale Aktivität (`conditionalTarget` Relation). Im Gegensatz dazu besitzt eine einfache Kante genau eine Zielaktivität, die über die Beziehung `controlTarget` angegeben wird. Generell beinhaltet eine Kontrollkante eine Aktivitätsquelle (`controlSource` Relation). Konditionale Kanten werden über eine boolsche Variable aktiviert (`activatedBy` Relation). Abbildung 4.9 (oben) stellt die entsprechenden Konzepte vor.

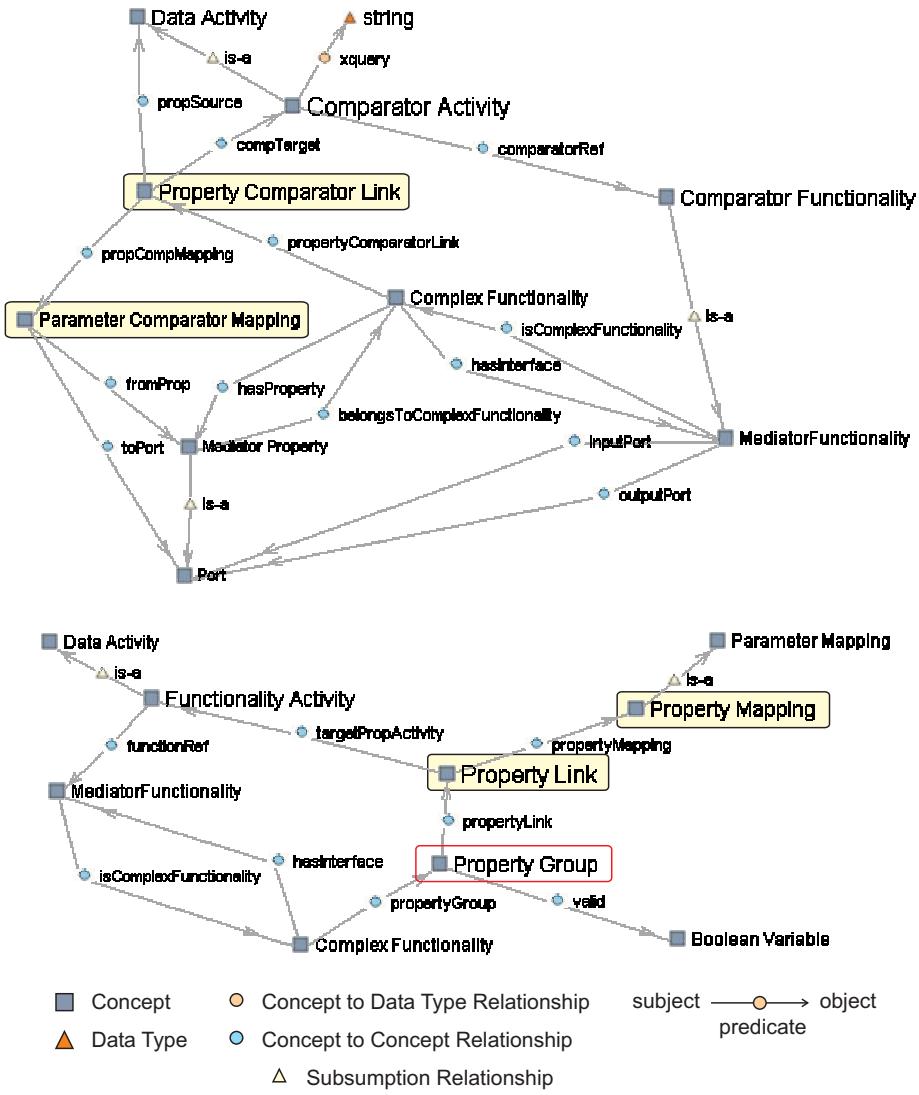

Abbildung 4.10: Konzepte zur Beschreibung des Komparator-Verbindungen (oben) und der Property-Verbindungen (unten).

Datenknoten und Datenübertragungen

Der untere Teil der Grafik 4.9 illustriert die Konzepte zur Beschreibung der Datenflussgraphen. Hier steht das Konzept `Data Link` im Vordergrund. Die Datenquelle und die Datensenke werden über die Beziehungen `target` und `source` bestimmt. Man beachte, dass die konkrete Datenquelle und Datensenke dem Konzept `Data Activity` angehören muss.

Über das Konzept `Parameter Mapping` werden die einzelnen Parameter der Datenquelle und Datensenke aufeinander abgebildet. Diese geschieht über die Beziehungen `from` und `to`. Zusätzlich ist es möglich Parameter fest mit Werten zu belegen (`Parameter Setting`).

Zusammenfassend spiegeln die Konzepte `Data Link` und `Parameter Mapping` die Definition 4.9 wieder.

Property-Verbindungen

Abbildung 4.10 visualisiert die Konzepte zur Beschreibung der Eigenschaftsgraphen, wie sie in Definition 4.16 spezifiziert sind. Im oberen Teil werden die Property-Verbindungen zwischen Properties einzelner Datenaktivitäten mit Ports von Komparatoren verschaltet. Dies geschieht im wesentlichen durch die Konzepte `Property Comparator Link` und `Parameter Comparator Mapping`.

Der untere Teil der Abbildung 4.10 schildert die Property-Verbindungen von Properties in einer komplexen Funktionalität. Diese werden mittels der Konzepte `Property Link` und `Property Mapping` definiert, wobei ein `Property Mapping` ein Spezialfall der `Parameter Mappings` ist. Man beachte, dass `Property Links` zu Gruppen zusammengefasst werden (`Property Group`), deren Gültigkeit durch eine boolsche Variable festgesetzt wird. Diese Designentscheidung ergibt sich aus der Tatsache, dass durch bedingte Kontrollflüsse Property-Verbindungen und Belegungen ungültig sein können. Daher werden diese Verbindungen über die gleichen Variablen wie die bedingten Kontrollkanten gesteuert.

4.6 Verwandte Lösungsansätze

Das Konzept der Eigenschaftsfelder ist an die Anpassungseigenschaften des Komponentenmodells der JavaBeans angelehnt. Ähnliche Konzepte finden sich auch in C#, Grid Services und in dem sich in Entwicklung befindlichen WSDL 2.0. Bei Grid Services und WSLD 2.0 dienen die Properties jedoch nicht der Verhaltensmodifikation[11]. Im Gegensatz zu den Properties der JavaBeans werden diese in MPL inklusive ihrem Auswirkungsziel explizit angegeben.

Die entwickelten Kompositionsgraphen sind an die Kontrollstrukturen von WSFL (Leymann, 2001) angelehnt, über die Web Services graphbasiert komponiert werden können. Diese Sprache ist von ihrer Ausdruckskraft mächtiger, da in MPL bewusst nur zentrale Kontrollstrukturen erlaubt sind. Im Gegensatz dazu unterstützt MPL die Konfigurierung komponierter Einheiten, ein Aspekt der in WSFL keine Betrachtung findet. Ein weiterer Unterschied zu WSFL ist, das WSFL das *Death-Path Problem* nicht explizit löst, sondern einen Backtracking-Algorithmus vorstellt, mit dessen Hilfe eine Laufzeitumgebung das *Death-Path Problem* auflösen kann.

[11] Näheres hierzu im Anhang A.

Neben WSFL wurden im Kontext der Web Services Technologie weitere verschiedene Ansätze zur Beschreibung von Web Services Kompositionen vorgeschlagen. Derzeitiger Hauptkandidat für die Orchestration ist BPEL4WS, welches aus XLANG und WSFL hervorgeht. Daher stellt sich BPEL4WS als eine Mischung aus graph- und blockbasiertem Prozessmodell dar. Dieses führt zu einer ausdrucksstarken Sprache, die gleichzeitig aber sehr komplex ist (van der Aalst, 2003). Ferner konnte gezeigt werden, dass einige Konstrukte nicht exakt spezifiziert sind und BPEL4WS deswegen Mehrdeutigkeiten aufweist (Wohed et al., 2002). Neben der Komplexität von BPEL4WS unterstützt diese Sprache genau wie WSDL weder eine semantische Annotation noch das Konzept der Properties zur Konfigurierung des Verhaltens, weswegen diese Sprachen zur Beschreibung von Service-Mediatoren unzureichend sind.

Im Zuge des *Semantic Web* ist die Beschreibungs- und Kompositionssprache OWL-S entstanden, die aus DAML-S hervorgeht (Martin et al., 2004). Diese Sprache erlaubt die Beschreibung eines Web Services als atomare und kombinierte Einheit. Letzteres wird durch ein entsprechendes Prozessmodell ermöglicht. Weiterhin ist es möglich die technische Realisierung, d.h. wie der Service angesprochen wird, in einem so genannten *Grounding* zu spezifizieren. Die Schnittstellenbeschreibung geschieht im so genannten *Service Profile*, welches zu dem hier entworfenen *Mediator Profile* Gemeinsamkeiten aufweist. So sind beispielsweise die Konzepte zur Beschreibung des Anbieters sowie der Kategorie des Services an OWL-S angelehnt. Auch OWL-S entwickelt sich ständig weiter, so werden seit Version 0.7 ferner Beschreibungen so genannter *Profil-basierte Klassenhierarchien* unterstützt, über die einem Parameter eine *Produktklasse* zugeordnet werden kann. Über diese Produktklassen ist es prinzipiell möglich, ähnlich wie bei MPL, Parameter auch semantisch auszuzeichnen. Aus diesen Gründen korrespondieren beide Sprachen, so dass der hier vorgestellte Ansatz von OWL-S beschriebenen Services profitieren kann. Leider ist OWL-S kein Standard der Web Services Technologie, noch ist davon auszugehen, dass sich OWL-S kurzfristig durchsetzen wird. Die Orchestration der Dienste in OWL-S wird durch das *Process Model* definiert, welches sich als Blockstruktur darstellt und die Beschreibung komplexer Workflows ermöglicht.

Obwohl die neueren Versionen von OWL-S prinzipiell eine semantische Annotation ermöglichen, ist OWL-S für die Beschreibung von Service-Mediatoren ungeeignet, da es keine Konfigurationsmöglichkeiten der atomaren oder komplexen Services bereitstellt. Außerdem werden die Semantik und die Syntax eines Parameters nicht sauber voneinander getrennt, sondern der Typ eines Parameters durch eine URI spezifiziert, die wahlweise auf eine Produktklasse oder einen Datentyp verweisen kann. Aus diesem Grund lassen sich Produktklassen eher als komplexe Typen (Klassen) einer objektorientierten Sprache auffassen. Durch die saubere Trennung dieser beiden Einheiten (Semantik und Syntax) in MPL können jedoch verschiedene syntaktische Typen einem einzelnen semantischen Konzept zugeordnet werden und umgekehrt. Hierdurch werden die Nutzungsmöglichkeiten der Sprache größer. Beispielsweise kann wahlweise nach syntaktischen, semantischen oder beiden Merkmalen zusammen gesucht werden, wenn ein Service-Mediator benötigt wird.

Die definierten Anpassungsmöglichkeiten können vom Workflowdesigner eingesetzt werden, um die Service-Interoperabilität durch die Wiederverwendung eines Service-Mediators zu erzielen. Der Entwickler eines neuen Service-Mediators kann sie gleichermaßen nutzen, um auf Basis existierender Service-Mediatoren neue Service-Mediatoren zu erstellen, die unabhängig von einem konkreten Workflow sind. Dieses Prinzip wird von Alda et al. (2004) im Rahmen komponentenbasierter Peer-To-Peer-Systeme weiter diskutiert.

4.7 Zusammenfassung

Service-Mediatoren überbrücken die syntaktische und semantische Barriere zwischen heterogenen Services. Doch was unterscheidet einen Service-Mediator von einem Service? Dieser Frage geht dieses Kapitel unter anderem nach. Weiterhin werden konkrete softwaretechnische Anforderungen an Service-Mediatoren erhoben, denen durch die Spezifikation eines Komponentenmodells Rechnung getragen wird. Zu den elementaren Anforderungen gehören die syntaktische und semantische Beschreibung der Fähigkeiten und die damit verbundene Bereitstellung eines Introspektionsmechanismus', sowie die Fähigkeit der Anpassung.

Sowohl die Exploration der Fähigkeiten als auch die Anpassung durch Konfigurierung und Komposition werden mittels der OWL-basierten *Mediatorprofilsprache* (MPL) ermöglicht. Im vorgeschlagenen Komponentenmodell geschieht die Konfigurierung eines Service-Mediators mit Hilfe so genannter *Eigenschaftsfelder* (*Properties*), die im Mediatorprofil beschrieben werden. Diese werden inklusive ihrem Auswirkungsziel, d.h. der konkreten Mediatorfunktionalität, explizit angegeben.

Gegenstand der Komposition sind die so genannten *Mediatorfunktionalitäten*, die durch die Spezifikation von *Kompositionsgraphen* zu komplexen Einheiten aggregiert werden können. Diese Graphen unterteilen sind in Kontrollflussgraphen, Datenflussgraphen und Eigenschaftsgraphen, die durch MPL unterstützt werden. Beim Entwurf der Kontrollflussgraphen wurden bewusst nur wenige, zentrale Kontrollstrukturen erlaubt, um die Anwendung weitestgehend intuitiv und effizient zu gestalten. Durch die entwickelten Eigenschaftsgraphen wird auch die Konfigurierung komplexer Mediatorfunktionalitäten über Eigenschaftsfelder ermöglicht.

Kapitel 5

Identifizierung und Anpassung von Service-Mediatoren

Die Identifizierung und Anpassung von Service-Mediatoren, die innerhalb eines gegebenen Workflows die Service-Interoperabilität realisieren sollen, geschieht durch eine semiautomatische, software-unterstützte Prozedur. Diese Prozedur gliedert sich in vier verschiedene Phasen. Ziel der Prozedur ist es, Informationen aus den vorhandenen WSDL-Beschreibungen der einzelnen Services zu extrahieren (Anfragegenerierung), die eine Suche nach geeigneten Service-Mediatoren ermöglichen (Discovery). Identifizierte Service-Mediatoren werden in einem weiteren Schritt den konkreten Bedürfnissen des Workflows angepasst (Adaption) und abschließend gespeichert, um zukünftigen Workflows zur Verfügung zu stehen (Bekanntmachung).

A theory can be proved by experiment;
but no path leads from experiment to the birth of a theory.
— Albert Einstein (1875-1955), *Theories of Relativity*

The noblest search is the search for excellence.
— Lyndon Baines Johnson (1908-73), *36. US President, Demokrat*

Unabhängig vom eingesetzten Verfahren zur Erreichung der Service-Interoperabilität in einem Workflow bedarf es grundsätzlich einer Analyse des Workflows mit dem Ziel problematische Datenflüsse zu identifizieren. Anders ausgedrückt, es müssen die kombinierten Dienste eines Workflows erkannt werden, die nicht der Definition 3.3 genügen. Aus dieser Analyse ergeben sich Aussagen darüber, welche Anforderungen ein konkreter Service-Mediator oder jeder andere Glue-Code erfüllen muss. Anschließend muss der identifizierte Service-Mediator bzw. allgemein der entwickelte oder teilweise generierte Glue-Code in den Workflow entsprechend eingesetzt werden.

Die in dieser Arbeit entwickelte semiautomatische, *software-unterstützte Prozedur* (*software-aided procedure*) bricht dieses Vorgehen weiter auf und stellt dabei eine Zerlegung des Problems vor, die es einerseits erlaubt, existierende Ansätze zur automatischen Generierung benötigter Service-Mediatoren zu integrieren (vergleiche Matching-Ansätze in Kapitel 3.3), und es andererseits ermöglicht, die gesamte Prozedur auf andere Middlewareplattformen, wie beispielsweise Grid Services oder CORBA Services, zu portieren. Die entwickelte Prozedur ist ferner applikationsunabhängig.

Dieses Kapitel liefert in Abschnitt 5.1 einen Überblick über die software-unterstützte Prozedur zur Erzielung der Service-Interoperabilität, die abkürzend *QDAS-Prozedur* genannt wird. Anschließend werden die verschiedenen Phasen der QDAS-Prozedur ausführlich diskutiert. Die Diskussion beinhaltet die Herleitung des Anfrageprofils (Q*uery Generation*) in Abschnitt 5.2, die Entdeckung (D*iscovery*) der benötigten Service-Mediatoren in Abschnitt 5.3 und die konkrete Anpassung (A*daptation*) der ausgewählten Service-Mediatoren in Abschnitt 5.4. Diese drei Phasen werden unter dem Stichwort *Suchprozess* zusammengefasst. Die endgültige *Bekanntmachung* (S*toring*) der angepassten oder neu erstellten Service-Mediatoren wird in dieser Arbeit nicht näher behandelt. Die prototypische Umsetzung der QDAS-Prozedur wird in Abschnitt 5.5 besprochen. In Abschnitt 5.6 werden Herausforderungen der verschiedenen Phasen und verwandte Lösungsansätze erörtert. Das Kapitel endet mit einer kurzen Zusammenfassung. In Kapitel 6 werden die hier theoretisch vorgestellten Methoden anhand eines exemplarischen *in silico* Experiments praktisch vorgestellt.

5.1 Eine software-unterstützte Prozedur zur Erzielung der Service-Interoperabilität

Die Erzielung der Service-Interoperabilität mittels atomarer oder teilweise komponierter Service-Mediatoren wird erst durch eine semiautomatische, software-unterstützte Prozedur (QDAS-Prozedur) ermöglicht. Sie unterstützt den Workflowdesigner bei der Erstellung eines interoperablen Workflows. Die durch Service-Mediatoren erreichte Interoperabilität wird im Folgenden auch mediatorbasierte Service-Interoperabilität genannt:

Definition 5.1 (Mediatorbasierte Service-Interoperabilität)**.** Seien W, S und S' gegeben wie in Definition 3.3, jedoch hinsichtlich O und I nicht interoperabel. Dann sind S und S' hinsichtlich O und I *mediator-interoperabel*, wenn ein Service-Mediator M mit einer Mediatorfunktionalität $f : (i_f, \mathcal{P}_f^{\wp}) \rightarrow (o_f)$ existiert, und S und M hinsichtlich O und f interoperabel ist, sowie gleichzeitig M und S' hinsichtlich f und I interoperabel ist. ■

In obiger Definition beschreibt f die Schnittstelle der Mediatorfunktionalität. Diese wird durch eine konkrete Implementierung $\mathfrak{f}$ realisiert, die die eigentliche Transformation bereitstellt. Da sich die Definition 5.1 auf die Definitionen 3.3 und 3.4 stützt folgt, dass ein

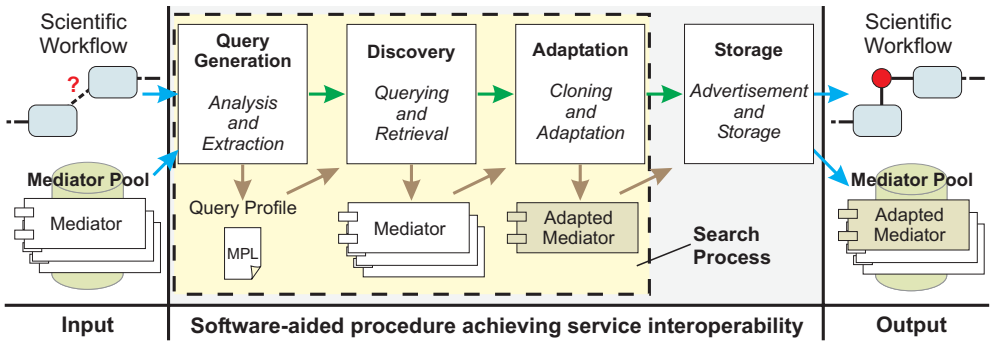

Abbildung 5.1: Schematisches Ablaufdiagramm der QDAS-Prozedur zur mediatorbasierten Service-Interoperabilität mit der Unterteilung in den komplexen Suchprozess (Anfragegenerierung, Discovery, Anpassung) und die Bekanntmachung.

Workflow, dessen komponierte Services alle mediator-interoperabel sind, auch interoperabel ist. Auch hier gilt jedoch, dass aus der Interoperabilität des Workflows sich nicht die semantische Korrektheit der Ausführung des Workflows ableiten lässt (vgl. Abschnitt 3.2). Definition 5.1 lässt sich ebenfalls auf eine Menge von datenproduzierenden Operationen und einer einzelnen datenkonsumierenden Operation erweitern.

Notation: Für den Sachverhalt, dass eine Mediatorfunktionalität f die mediatorbasierte Service-Interoperabilität zwischen zwei Operationen O und O' zweier Services erreicht, wird im Folgenden auch die abkürzende Schreibweise $O \rightsquigarrow_f O'$ verwendet. Auch hier gilt die Erweiterung, dass O eine Vereinigung mehrerer Operationen darstellen kann. Damit lässt sich zusammenfassend folgende Aufgabenstellung für die QDAS-Prozedur formulieren:

> *Gegeben ein Workflow W mit Datenfluss F_d, dann ist für alle verknüpften O und O' in F_d, die nicht interoperabel sind, ein Service-Mediator M gesucht, der die Funktionalität f realisiert, so dass $O \rightsquigarrow_f O'$ gilt.*

Die gesamte QDAS-Prozedur, mit deren Hilfe die mediatorbasierte Service-Interoperabilität eines gesamten Workflows erreicht werden soll, sowie ihre Unteraufgaben werden im Folgenden standardmäßig durch die Faktoren *Input-Verarbeitung-Output* beschrieben. Abbildung 5.1 illustriert die komplette QDAS-Prozedur.

Der *Input* der kompletten QDAS-Prozedur besteht aus dem entworfenen Workflow W, der über einen Kontrollfluss F_c und einen Datenfluss F_d spezifiziert ist, sowie einer Menge von Service-Mediatoren, die in dem *Mediatorpool* registriert sind. Initial ist dies eine Menge von essentiell benötigten Mediatoren eines Anwendungsgebietes. In diesem Pool werden lediglich die Fähigkeiten, d.h. die Profile, der einzelnen Service-Mediatoren registriert, nicht jedoch die Service-Mediatoren selber. Diese bleiben im Internet verteilt verfügbar.

Bei der *Verarbeitung* der kompletten QDAS-Prozedur wird der Datenfluss F_d analysiert. Für alle datenkonsumierenden Operationen O_k, die bezüglich ihrer datenproduzierenden Operationen O_p nicht interoperabel sind, wird die QDAS-Prozedur durchlaufen. Ziel der QDAS-Prozedur ist, einen geeigneten Service-Mediator zu identifizieren und diesen den Anforderungen entsprechend anzupassen, so dass O_k und O_p mediator-interoperabel werden und der Workflow abschließend interoperabel ist. Diese Mediatoren werden am Ende der QDAS-Prozedur im Mediatorpool abgelegt und in den Workflow eingebettet.

Der *Output* der kompletten QDAS-Prozedur besteht somit aus einer neuen Ausprägung des Mediatorpools, in dem speziell adaptierte beziehungsweise neu erstellte Service-Mediatoren eingefügt wurden. Die Änderung der Ausprägung des Mediatorpools ergibt sich nur dann, wenn in einem Workflow nicht-interoperable Operationen identifiziert werden, die in früheren Anwendungen der QDAS-Prozedur noch nicht betrachtet wurden und kein anpassbarer Service-Mediator existiert, mit dessen Hilfe die Operationen mediator-interoperabel werden. Der Output besteht ferner aus einem angepassten Workflow W', in dem diese Mediatoren eingebaut wurden. Die Einbettung ist ohne eine Erweiterung der Beschreibungssprachen für Servicekompositionen möglich, da Service-Mediatoren technisch ebenfalls durch WSDL beschrieben werden (vgl. Kapitel 4).

Die QDAS-Prozedur lässt sich prinzipiell in vier voneinander entkoppelte Phasen unterteilen. Durch diese Aufteilung wird die Erweiterbarkeit und Ersetzbarkeit der für die verschiedenen Phasen entwickelten Softwarekomponenten ermöglicht. Beispielsweise lassen sich als spezielle Form der Anpassung Ansätze zur automatischen Mediation integrieren. Dieses resultiert in einer offenen und wartbaren Prozedur (Radetzki et al., 2004c). Zwischen den einzelnen Phasen besitzt der Nutzer die Möglichkeit in die Abarbeitung der QDAS-Prozedur einzugreifen und Outputs der Phasen zu modifizieren. Als Basis nutzen die Softwarebausteine, die die einzelnen Phasen realisieren, zum einen die Mediatorprofilsprache (MPL) (siehe Kapitel 4.5) und zum anderen eine gemeinsame Datenbasis (*Mediatorpool*), in der die Mediatorprofile der registrierten Service-Mediatoren abgelegt werden. Folgende Phasen werden unterschieden:

1. *Anfragegenerierung*: Als erster Schritt bei der Suche nach geeigneten Service-Mediatoren im Mediatorpool wird vom System zunächst aus den Schnittstellenbeschreibungen der Operationen O_k und O_p, die in WSDL vorliegen, eine Beschreibung der benötigten Mediatorfunktionalität eines Mediators generiert (*Anfrageprofil*).

2. *Discovery*: Ein spezieller Retrievalprozess gleicht die Beschreibung des Anfrageprofils mit der Beschreibung der Mediatoren des Pools ab und schlägt geeignete Mediatoren vor. Hierbei wird auch eine mögliche Komposition unabhängiger Mediatoren berücksichtigt.

3. *Adaption*: Die Anpassung der konkret benötigten Mediatorfunktionalität erfolgt durch den Benutzer in einem vom System unterstützten Prozess. Gerade die komponentenorientierte Realisierung der Mediatoren schafft hier die benötigte Flexibilität und

unterstützt die Wiederverwendung einzelner Softwarebausteine in unterschiedlichen Kontexten.

4. *Bekanntmachung*: Die Funktionalität der neu geschaffenen oder auch nur gezielt angepassten Mediatoren wird im Mediatorpool registriert und steht für andere Nutzungskontexte zur Verfügung. Dieses bildet den Übergang zu einer neuen Ausprägung des Mediatorpools.

Die ersten drei Phasen bilden einen eigenständigen *Suchprozess*, der ausführlich in den folgenden Abschnitten der Arbeit diskutiert wird. Die anschließende Bekanntmachung von Service-Mediatoren wird nur der Vollständigkeit halber erwähnt, jedoch nicht weiter in dieser Arbeit vertieft.

5.2 Anfragegenerierung – Beschreibung benötigter Service-Mediatoren

Der software-unterstützte Prozess beginnt mit der Analyse des Workflows. Genauer gesagt, sei W der betrachtete Workflow mit Datenfluss F_d, dann wird für alle datenproduzierenden und datenkonsumierenden Operationen in F_d, die nicht interoperabel sind, eine Anfrage an den Mediatorpool generiert. Hierzu werden die vorhandenen Schnittstellenbeschreibungen der korrespondierenden Services, die in WSDL vorliegen, herangezogen. Die generierte Anfrage sollte die Fähigkeiten beschreiben, die ein konkreter Service-Mediator erfüllen muss, um zwischen den Operationen die mediatorbasierte Service-Interoperabilität zu erreichen. Hierbei sollten ebenfalls syntaktische als auch semantische Aspekte berücksichtigt werden.

Anfragen werden – entsprechend der Fähigkeiten der Service-Mediatoren – ebenfalls in MPL in Form von *Anfrageprofilen* ausgedrückt. Das Anfrageprofil beschreibt die Anforderungen an die konkreten Service-Mediatoren. Ein Service-Mediator mit dem gleichen oder einem ähnlichen Profil wird als Antwort auf die Anfrage verstanden. Kurz gesagt, die Aufgabe der Anfragegenerierung besteht für ein gegebenes W darin, für alle datenproduzierende Operationen O_p und datenkonsumierende Operationen O_k, die nicht interoperabel sind, Beschreibungen q mit $O_p \rightsquigarrow_q O_k$ zu generieren.

Probleme bei der Anfragegenerierung

Wie bei der Vorstellung von WSDL erläutert, erlaubt dieser Standard innerhalb des PortTyps nur die Auszeichnung von syntaktischen Aspekten der Schnittstelle (vgl. Kapitel 2.3.1). Lediglich textbasierte, inhaltliche Beschreibungen können zusätzlich den verschiedenen Elementen angehängt werden. Eine semantische Auszeichnung durch entsprechende Konzepte einer Ontologie wird durch WSDL nicht unterstützt. Die verwendeten Namen der verschiedenen Operationen, Input- und Outputports sind häufig zusammengesetzt und wenig

aussagekräftig. Wenn sie zusammengesetzt werden, findet man jedoch häufig den Fall, dass keine korrekten Worte verwendet, sondern verschiedene Abkürzungen benutzt werden (Dong et al., 2004). Die syntaktischen Datentypen der Ports sind ferner irreführend. In der Praxis werden beispielsweise häufig *generische Typen* verwendet, d.h. eine Serviceoperation liefert einen String, der jedoch ein komplexes XML Dokument kodiert. Dieses lässt sich anhand der rein syntaktischen Schnittstelle jedoch nicht erkennen. Diese Gegebenheiten gestalten die Anfragegenerierung, die sowohl syntaktische als auch semantische Informationen kodieren soll, als schwierig.

Semantische Konzepte und linguistische Methoden

Input der Anfragegenerierungsphase ist der Workflow inklusive der WSDL-Dokumente teilnehmender Services.

In der *Verarbeitung* analysiert der hier vorgeschlagene Algorithmus zur Entwurfszeit des Workflows die verschiedenen WSDL-Dokumente. Im ersten Schritt des Algorithmus wird die syntaktische Information (Namen und Datentypen der Ports) sowie verfügbare inhaltliche Beschreibungen in das Anfrageprofil übernommen. Um ein Mapping zwischen der Domänenontologie und den Ports herzustellen, werden diese Informationen in einem zweiten Schritt weiter analysiert. Portnamen werden, wenn möglich, aufgetrennt, d.h. eine Trennung findet statt, wenn in einem Portnamen Großbuchstaben verwendet werden. Ferner werden die Dokumentationen analysiert, wobei normalisierte Terme identifiziert werden. Hier kommen Techniken wie Stopwortlisten und Wortstammbildung (*stemming*) zum Einsatz, die u.a. auch im Information Retrieval verwendet werden. Die so identifizierten Terme werden anschließend mit den Konzepten der Ontologie verglichen (*concept matching*) und bei Übereinstimmung werden Mappings definiert.

Der Benutzer kann zusätzlich entscheiden, ob linguistische Methoden beim Mapping eingesetzt werden sollen. Beispielsweise können Synonyme, Hyponyme oder Hyperonyme sowie Taxonomien mit gängigen Abkürzungen eingesetzt werden. Werden über linguistische Methoden Konzepte identifiziert, werden diese schwächer bewertet (vgl. Kapitel 4.5.1). Ferner kann jederzeit der Benutzer eigenständig das Anfrageprofil erweitern bzw. modifizieren.

Gegenwärtig werden weder eine Anwendungsdomäne noch eine Anwendungsklasse durch den Ansatz identifiziert. Auch die Properties möglicher Service-Mediatoren werden ignoriert, da sie gewöhnlich nicht direkt relevant für das Discovery sind. Zukünftige Erweiterungen könnten jedoch eine maximale Anzahl von Properties vorschreiben oder den Grad der Komplexität bei der Anpassung mit berücksichtigen.

Die Verarbeitung liefert als *Output* ein Anfrageprofil q mit $q : (i_q, \emptyset) \rightarrow (o_q)$ und Annotationen $\mathcal{A}_q = (\omega_q, desc_q, dom_q, class_q)$, welches in MPL spezifiziert ist. Dieses beschreibt Anforderungen an konkrete Mediatoren. Die folgende Discovery-Phase gleicht dieses Profil anschließend mit Profilen konkreter Service-Mediatoren ab.

5.3 Discovery – Identifizierung geeigneter Service-Mediatoren

In dieser Phase der Prozedur müssen Service-Mediatoren identifiziert werden, die entweder exakt die Anforderungen des Anfrageprofils erfüllen (*exact match*) oder die anstelle eines exakt übereinstimmenden Service-Mediators zwischen die teilnehmenden Services einsetzt werden können (*plug-in match*). Weiterhin sollten auch abgeschwächte Formen des Discovery unterstützt werden (*relaxed match*), wie die nachfolgende Problematik aufzeigt. Der Discovery-Prozess nach geeigneten Service-Mediatoren bzw. deren Mediatorfunktionalitäten kann allgemein auch als *Matchmaking-Problem* aufgefasst werden (Sycara et al., 1999).

Probleme beim Matchmaking

Es ist eher unwahrscheinlich einen Service-Mediator zu identifizieren, der direkt eingesetzt werden kann (*plug-in match*), noch einen Service-Mediator zu identifizieren, der exakt den Anforderungen genügt (*exact match*). Dieses hat verschiedene Gründe: Zu allererst ist das hergeleitete Anfrageprofil nicht exakt sonder unscharf. Dazu kommt, dass auch die Spezifikationen der Fähigkeiten eines Service-Mediators eine gewisse Unschärfe aufweisen. Ein Grund dafür sind teilweise komplexe Berechnungen innerhalb der Mediatorfunktionalitäten, die eine rein auf logischen Bedingungen basierende Beschreibung fast unmöglich machen. Ferner werden Services und Service-Mediatoren durch unabhängige Provider erstellt, wodurch sie nicht direkt aufeinander abgestimmt sind. Es ist eher wahrscheinlich, dass eine Komposition verschiedener Service-Mediatoren zusammen das Anfrageprofil *matchen*. Zusammenfassend werden daher *relaxed matchings* in Kombination mit der Komposition von Service-Mediatoren benötigt, die durch das Discovery direkt ermittelt werden sollten.

Discovery-Einheit

Die in dieser Arbeit entwickelte *Discovery-Einheit* beinhaltet verschiedene Matchmaking-Verfahren, die kombiniert, iterativ und rekursiv miteinander ausgeführt werden können, um möglichst zum Anfrageprofil identische Service-Mediatoren zu identifizieren (Radetzki und Cremers, 2004). Die Auswahl der Verfahren beruht auf Ergebnissen des Komponenten- und Agenten-Matchmakings (Klusch und Sycara, 2001; Zaremski und Wing, 1997) (vgl. auch Kapitel 5.6). Werden zukünftig neue Algorithmen im dem Bereich des Information Retrieval entwickelt, können diese problemlos in den Prozess eingebettet werden. Folgende Algorithmen wurden entworfen und realisiert:

- *Anwendungsdomänen-Matchmaking*: Liefert Mediatorfunktionalitäten, die einer konkreten Anwendungsdomäne angehören.

- *Anwendungsklassen-Matchmaking*: Liefert Mediatorfunktionalitäten, die eine konkrete Realisierungsform, wie Komparatoren oder Parser, unterstützen.
- *Keyword-Matchmaking*: Liefert Mediatorfunktionalitäten, die bestimmte Schlüsselwörter in ihrer inhaltlichen, unstrukturieren Beschreibung besitzen.
- *Signatur-Matchmaking*: Liefert einzelne oder komponierte Mediatorfunktionalitäten, die die entsprechende Signatur, d.h. Datentypen und Portnamen, bereitstellen.
- *Konzept-Matchmaking*: Liefert einzelne oder komponierte Mediatorfunktionalitäten, die die entsprechenden Konzepte der Domänenontologie bereitstellen.

Jedes Verfahren besitzt ein entsprechendes Distanzmaß, das unterschiedlich gewichtet werden kann. Eine Normierung und Kombination der Distanzen über eine kombinierte Ausführung findet dabei nicht statt.

Die einzelnen Verfahren unterscheiden sich in der Genauigkeit und den Berechnungskosten. Generell gilt, je geringer die Genauigkeit, desto geringer die Berechnungskosten. Verfahren mit geringerer Genauigkeit können gut zum Filtern eingesetzt werden. Einerseits kann hierdurch der Suchraum effektiv reduziert werden, andererseits können irrelevante Service-Mediatoren frühzeitig entfernt werden, beispielsweise Service-Mediatoren die nicht der gesuchten Anwendungsdomäne angehören, wodurch sich auch die Genauigkeit nachfolgender Retrievalverfahren erhöhen kann. Weiterhin gilt, dass komplexe *Matchmakings*, die neue Kompositionen von Mediatorfunktionalitäten identifizieren, lediglich einmal ausgeführt werden, da das Resultat, d.h. die neue komplexe Mediatorfunktionalität, als eigenständige Mediatorfunktionalität gespeichert wird und anschließend für zukünftige Retrieval-Durchgänge direkt verfügbar ist.

Komplexes Matchmaking: Komposition der Service-Mediatoren

Das komplexe Matchmaking liefert neue Kompositionen von Mediatorfunktionalitäten. Die Grundlage für die Identifizierung neuer Kompositionen sind die konkreten, registrierten Mediatorfunktionalitäten, die durch ihre Input- und Outputports *Operationsnetzwerke* aufspannen. Diese Netzwerke müssen bei der Registrierung der Mediatorfunktionalitäten hergeleitet werden. Die Portnamen, Datentypen und Konzepte der einzelnen Funktionalitäten dienen als Basis für die Herleitung. Im Allgemeinen ist die Herleitung nicht trivial, da einerseits Untertyprelationen und Subsumptionsrelationen beachtet werden müssen, andererseits unterschiedliche Portnamen für semantisch äquivalente Ports verwendet werden und vice versa.

Abbildung 5.2 zeigt beispielhaft ein derartiges Operationsnetzwerk. Die verschiedenen Farben der Input- und Outputports repräsentieren unterschiedliche Portspezifikationen. Diese Unterschiede treten aufgrund verschiedener Portnamen, Datentypen und/oder Konzepte auf. Eine Verbindung zwischen zwei aufeinander folgenden Funktionalitäten muss ferner

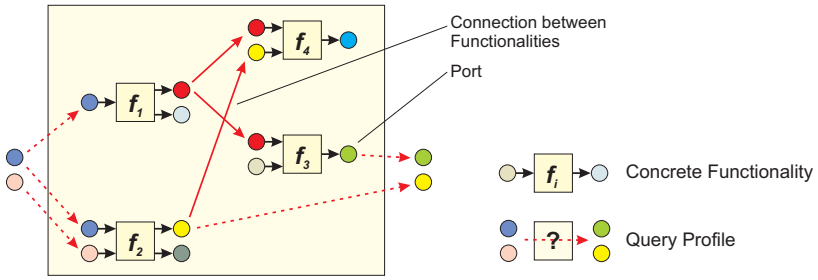

Abbildung 5.2: Schematische Darstellung des komplexen Matchmakings. Die verschiedenen Farben illustrieren unterschiedliche Portspezifikationen (verschiedene Portnamen, Datentypen und/oder Konzepte).

nicht vollständig sein. Es kann dazu kommen, dass Inputports nicht komplett belegt werden (siehe f_3) oder Resultate an Outputports von den nachfolgenden Funktionalitäten nicht benötigt werden (siehe f_2). Nicht alle Inputports einer Funktionalität müssen durch eine einzelne vorherige Funktionalität belegt werden, sondern sie können auch durch Outputports vieler Funktionalitäten bestimmt werden (siehe f_4). Es kommt zu einer Aufteilung der Inputports (*Portzerlegung*).

Beim komplexen Matchmaking muss das Teilnetz identifiziert werden, welches die Anforderungen des Anfrageprofils am besten erfüllt. Das identifizierte Teilnetz sollte zudem minimal sein, d.h. es sollte keine unnötigen Funktionalitäten enthalten. Beispielsweise ist f_4 zur Beantwortung der Anfrage irrelevant, da es zur Lösung nicht beiträgt.

Das zurückgegebene Teilnetz bestimmt im wesentlichen den Datenfluss zwischen den einzelnen Mediatorfunktionalitäten. Durch diesen Datenfluss wird zudem ein erster Kontrollfluss induziert. Die so entstehenden Mediatorkompositionen müssen teilweise durch den Benutzer angepasst werden. Dies ist vorallem dann notwendig, wenn spezielle Konfigurierungen der einzelnen Funktionalitäten anzugeben sind oder wenn bedingte Aktivierungsübergänge in der Komposition benötigt werden. Letztere lassen sich nicht anhand der Operationsnetzwerke erkennen.

Retrievalvarianten

Neben der Kombination der verschiedenen Matchmaking-Verfahren lassen sich folgende Retrievalvarianten unterscheiden:

- *Automatisches Retrieval*: Bei der automatischen Variante nimmt der Benutzer keinen Einfluss auf das Retrieval. Das heißt, es werden Standardgewichte der Matchmaking-Verfahren angenommen und nur die Informationen eingesetzt, die direkt durch die Anfragegenerierung im Anfrageprofil enthalten sind.

- *Kooperatives Retrieval*: Das kooperative Retrieval zeichnet sich dadurch aus, dass der Benutzer die Gewichtungen den Bedürfnissen seines Anwendungskontextes anpassen kann. Auf diese Weise lässt sich auch die Anzahl verwendeter Mediatorfunktionalitäten nach oben beschränken. Zudem kann der Benutzer Schranken angeben, ab welchen konkrete Funktionalitäten nicht mehr betrachtet werden sollen. Hiermit lässt sich beispielsweise der Suchraum für nachfolgende Matchmaking-Verfahren effektiv reduzieren. Zu dem kooperativen Retrieval zählt auch das Hinzufügen weiterer Informationen, wie Schlüsselwörter oder semantischer Konzepte. Der eigentliche Retrievalprozess verläuft anschließend wie beim automatischen Retrieval.

- *Interaktives Retrieval*: Im Gegensatz zum kooperativen Retrieval greift der Benutzer beim interaktiven Retrieval aktiv in den Suchprozess ein. Das heißt, der Benutzer kann einen eher iterativen Retrievalprozess starten, bei dem er konkrete Funktionalitäten als zu ignorieren oder als speziell zu betrachten auszeichnet. Dieses ist vor allem bei den komplexeren Matchmaking-Algorithmen, die neue komplexe Funktionalitäten identifizieren können, sinnvoll. Hier könnte beispielsweise der Benutzer eine konkrete Funktionalität aus einer identifizierten Komposition herausnehmen und anschließend für diese eine andere Funktionalität suchen lassen. Weiterhin bietet dieses Retrieval alle Möglichkeiten des kooperativen Retrievals.

- *Manuelles Retrieval*: Beim manuellen Retrieval kann der Benutzer vorhandene Strukturen nutzen, um selbständig nach geeigneten Funktionalitäten zu suchen. Zu den Strukturen zählen vor allem die Domänenontologie, aber auch die Taxonomie der Anwendungsdomäne und der Anwendungsklassen. Diese Retrievalvariante zeichnet sich vor allem durch das explorative Verhalten des Benutzers aus (*browsing*). Weiterhin stehen dem Benutzer alle Möglichkeiten des interaktiven Retrievals zur Verfügung.

Der *Input* dieser Phase der software-unterstützten Prozedur besteht somit aus dem Anfrageprofil q mit $q : (i_q, \emptyset) \rightarrow (o_q)$ und Annotationen $\mathcal{A}_q = (\omega_q, desc_q, dom_q, class_q)$ sowie dem Mediatorpool. Die *Verarbeitung* nutzt die verschiedenen Matchmaking-Verfahren zur Suche und liefert als *Output* mögliche Service-Mediatoren als Kandidaten, die das Anfrageprofil q *matchen*. Im Folgenden werden die einzelnen Verfahren vorgestellt.

5.3.1 Anwendungsdomänen-Matchmaking

Workflows gehören gewöhnlich einer speziellen Anwendungsdomäne an. Daher werden häufig Mediatorfunktionalitäten benötigt, die der gleichen Anwendungsdomäne entstammen, d.h. sie teilen sich den gleichen Anwendungskontext. Somit gilt, ist dom_W die Anwendungsdomäne des Workflows, dann ist $dom_q = dom_W$.

Die verschiedenen Anwendungsdomänen lassen sich gewöhnlich auf eine Taxonomie abbilden, die einer Baumstruktur gleicht. Innerhalb dieses Baums verfeinern Kinderknoten

den Anwendungskontext der Elternknoten. Der Wurzelknoten vereinigt alle Anwendungsdomänen. Einige Service-Mediatoren lassen sich keiner konkreten Anwendungsdomäne zuordnen, sondern verlaufen horizontal zu diesen, d.h. sie sind domänenunspezifisch und in verschiedenen Kontexten sinnvoll einsetzbar (vgl. Kapitel 4.2). Aus diesem Grund wird eine orthogonale Domäne $dom_\perp$ eingeführt (*general purpose service-mediators*). Sie ist nicht direkt mit dem Wurzelknoten gleichzusetzen. Würden domänenunspezifische Service-Mediatoren dem Wurzelknoten angehängt und nicht einer entsprechenden orthogonalen Domäne, würden diese bei jedem Retrievalgang identifiziert werden. Sie wären von domänenspezifischen Service-Mediatoren nicht direkt unterscheidbar. Die orthogonale Domäne erlaubt somit eine spezifischere Selektion durch den Matchmaking-Algorithmus.

Der Algorithmus zum Anwendungsdomänen-Matchmaking vergleicht die konkreten Service-Mediatoren mit der Domäne des Workflows, die im Anfrageprofil durch dom_q spezifiziert werden kann. Dazu greift dieses Verfahren auf eine gegebene, möglichst standardisierte, taxonomische Baumstruktur zu, wie beispielsweise NAICS[1]. Da in dieser Arbeit im wesentlichen wissenschaftliche Workflows betrachtet werden, wurde eine prototypische, rein wissenschaftlich angelegte Taxonomie definiert, mit deren Hilfe Service-Mediatoren in verschiedene wissenschaftliche Gebiete eingeteilt werden können. Neben dem exakten *matchen* unterstützt dieses Verfahren auch die Subsumptionsrelation von taxonomischen Termen.

Die Distanz $d_{\mathtt{DM}}$ einer konkreten Funktionalität $\mathfrak{f}$ mit dem Profil f, die einer Domäne dom_f angehört, lässt sich hinsichtlich der angefragten Domäne dom_q wie folgt definieren:

$$d_{\mathtt{DM}} = \begin{cases} 0, & dom_f \sqsubseteq dom_q \\ \omega \cdot l', & dom_f \sqsupset dom_q \\ u, & dom_f = dom_\perp \end{cases} \tag{5.1}$$

wobei l' die „normierte“ Länge des Pfades von dom_q nach dom_f in der taxonomischen Baumstruktur und ω einen Gewichtungsfaktor bezeichnet; es gilt $u \in \{0, \infty\}$ je nachdem, ob domänenunabhängige Funktionalitäten betrachtet bzw. nicht betrachtet werden sollen.

Die direkte Länge l des Pfades als Indiz für den „semantischen“ Abstand zur gesuchten Anwendungsdomäne zu nehmen, birgt die Gefahr, dass bestimmte Anwendungsdomänen detaillierter aufgeschlüsselt sein können als andere, und somit die Länge nicht den tatsächlichen Abstand wiedergibt. Man kann sich diesem Problem für Baumstrukturen nähern, indem man über die Gesamtlänge eines Pfads von der Wurzel bis zu einem Blatt die Länge der Pfade zwischen zwei Domänen „normiert“. Genauer gesagt, sei l die Länge des Pfades von der Wurzel bis zu einem Blatt, wobei die Domänen dom_q und dom_f auf diesem Pfad liegen, und sei k die Länge des Pfades von dom_q nach dom_f. Dann ist $l' = k/l$. Für komplexe Graphen, wie sie bei Ontologien auftreten, ist dieses Vorgehen nicht ohne weiteres übertragbar, da beispielsweise nicht immer eine einzelne Wurzel vorhanden ist (vgl. Kapitel 5.3.5).

Allgemein ist dieses Vorgehen lediglich eine Näherung, da auch hier unter Umständen nicht der exakte Abstand zwischen verschiedenen Domänen erfasst wird. Ein weiterer Weg wäre,

[1] `www.census.gov/epcd/www/naics.html`

die verschiedenen Domänen manuell in Detaillierungsebenen einzuteilen, über die dann eine Normierung im obigen Sinne vorgenommen wird.

Im Sinne des kooperativen Retrievals kann der Benutzer ω, u und eine Schranke für $d_{\mathtt{DM}}$ angeben. Zusätzlich können im Sinne des interaktiven Retrievals direkt Funktionalitäten aus der Resultatliste selektiert werden. Diese generelle Herangehensweise ist für alle Matchmaking-Verfahren identisch.

5.3.2 Anwendungsklassen-Matchmaking

Dieser Matchmaking-Algorithmus profitiert von unterschiedlichen Anwendungsklassen, wie sie beispielsweise in Kapitel 4.2 vorgestellt wurden. Jede konkrete Mediatorfunktionalität f kann einer Klasse $class_f$ zugeordnet werden. Da diese Zuordnung unscharf sein kann, ist der Grad der Zuordnung ebenfalls anzugeben. Derzeit werden die Grade durch ein $i \in I = \{0$='exactly'; 1='less exactly'; 2='weak'; 3='very weak'$\}$ unterstützt, welches zusätzlich angegeben werden kann. Die Einteilung der Funktionalitäten in Anwendungsklassen ist orthogonal zu der Einteilung in Anwendungsdomänen.

Mit diesen Informationen lässt sich nun die Distanz $d_{\mathtt{AM}}$ zwischen der Anwendungsklasse $class_q$ des Anfrageprofils q und der Anwendungsklasse $class_f$ einer konkreten Mediatorfunktionalitäten f für dieses Retrievalverfahren definieren. Für alle $class_q \equiv_i class_f$:

$$d_{\mathtt{AM}} = \omega \cdot i, \tag{5.2}$$

wobei $i \in I$ gilt und ω einen Gewichtungsfaktor repräsentiert.

Das Resultat dieses Matchmaking-Verfahrens ist eine Menge von konkreten Funktionalitäten, die alle der gleichen Anwendungsklasse $class_q$ angehören. Diese Art des Retrievals ist immer dann sinnvoll, wenn eine konkrete Anwendungsklasse benötigt wird, wie beispielsweise ein Komparator für einen bestimmten Vergleich in einer Mediatorkomposition.

5.3.3 Keyword-Matchmaking

Die verschiedenen Beschreibungen *desc.* der konkreten Mediatorfunktionalitäten werden bei der Bekanntmachung extrahiert und separat analysiert. Das Verfahren zum Keyword-Matchmaking beruht auf dem Vektorraum-Modell, über das der Ähnlichkeitsgrad zwischen verschiedenen Beschreibungen ermittelt werden kann (Salton, 1971). Damit ist ein Ranking der Resultatmenge möglich. Diese Technik ist aus dem Gebiet des Information Retrievals wohlbekannt und soll daher hier nur kurz skizziert werden.

Zunächst werden durch Stopwortlisten und Wortstammbildung (*stemmer*) Terme aus den einzelnen Beschreibungen extrahiert. Anschließend wird die *inverse document frequency* (IDF) für alle Terme k_i gebildet: $idf_i = \log(N/n_i)$, wobei N die Anzahl aller Beschreibungen

ist und n_i die Anzahl der Beschreibungen ist, die den Term k_i enthalten (Jones, 1972). Weiter sei $freq_{i,j}$ die Frequenz des Terms k_i in der Beschreibung $desc_j$. Dann lässt sich die normalisierte Termfrequenz $f_{i,j}$ definieren durch

$$f_{i,j} = \frac{freq_{i,j}}{\max_l freq_{l,j}}, \tag{5.3}$$

wobei l über alle Terme in $desc_j$ und $f_{i,j} = 0$, wenn k_i nicht in $desc_j$ vorkommt.

Die Gewichtung eines Terms lässt sich durch die *term frequency-inverse document frequency* (TF-IDF) definieren: $\omega_{i,j} = f_{i,j} \cdot idf_i$. Damit kann die Ähnlichkeitsdistanz $d_{\texttt{KM}}$ zwischen dem Beschreibungsvektor $\vec{d_j}$ der Beschreibung $desc_j$ und dem Anfragevektor $\vec{q}$ der Beschreibung $desc_q$ des Anfrageprofils q definiert werden durch

$$d_{\texttt{KM}} = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{t} \omega_{i,j} \cdot \omega_{i,q}}{\sqrt{\sum_{i=1}^{t} \omega_{i,j}^2} \cdot \sqrt{\sum_{i=1}^{t} \omega_{i,q}^2}}, \tag{5.4}$$

wobei t der Anzahl der Terme entspricht.

5.3.4 Signatur-Matchmaking

Im Folgenden verstehen wir unter der *Signatur* einen syntaktischen Aspekt, der die Namen der Funktionalitäten sowie die Anzahl, Namen und Datentypen der Ports beinhaltet. Genauer gesagt:

Definition 5.2 (Signatur)**.** Sei f eine Mediatorfunktionalität mit $f : (i_f, \mathcal{P}_f^{\wp}) \rightarrow (o_f)$ und sei ω_f der Name der Funktionalität laut Definition 4.2. Dann besteht die *Signatur* von f aus ω_f sowie der Anzahl und den syntaktischen Elementen der erweiterten Ports der Nachrichten i_f und o_f. Die syntaktischen Elementen eines Ports p mit $p = (\omega_p, d_p, C_p, desc_p)$ gemäß Definition 3.2 beinhalten den Namen ω_p und den Datentyp d_p. ■

Suchalgorithmen, die auf der Signatur des Anfrageprofils q aufsetzen, können einerseits rein auf den Datentypen der Ports arbeiten, andererseits zusätzlich die Namen der Ports und Funktionalitäten in Kombination mit den Datentypen berücksichtigen. Im ersten Fall führt dies gewöhnlich zu einer schlechteren *Precision*, während im letzteren Fall die *Precision* steigt auf Kosten des *Recall*, da nun das *Matching* sehr streng ist und Namensgleichheit bei den Portnamen voraussetzt. Im Kontext dieser Arbeit soll lediglich die zweite Variante vorgestellt werden, da die erste Variante eine Vereinfachung dieser darstellt.

Das entwickelte Verfahren berücksichtigt sowohl die aufgetrennten Portnamen[2] in Kombination mit den Datentypen, wobei gleichzeitig linguistische Methoden, wie Synonyme, als

[2]Portnamen werden an den Großbuchstaben aufgetrennt.

auch die Untertyprelation ($\preceq$) sowie beliebige Portreihenfolgen unterstützt werden. Ferner werden auf dieser Basis auch mögliche Kompositionen von Mediatorfunktionalitäten beim *Matchmaking* erkannt und berücksichtigt. Der Algorithmus besteht aus zwei Teilalgorithmen. Der erste Teilalgorithmus wird bei der Registrierung eines neuen Service-Mediators beim Mediatorpool verwendet und berechnet Verbindungen zwischen bereits vorhandenen Mediatorfunktionalitäten. Hierdurch wird das vorhandene Operationsnetzwerk erweitert. Der zweite Teilalgorithmus nutzt die gespeicherten Verbindungen des Operationsnetzwerkes, um eine konkrete Anfrage zu erfüllen.

Syntaktische Registrierung

Bei der Registrierung werden alle Portnamen der verschiedenen Funktionalitäten des neuen Service-Mediators mit den bereits vorhandenen Portnamen unter Berücksichtigung der Auftrennung und der Synonym-Relation verglichen (*Äquivalenz der Portnamen*). Die Reihenfolge der verschiedenen Ports ist dabei beliebig. Gleichzeitig werden die entsprechenden Datentypen verglichen. Hierbei werden die Substitution der Ports und die Untertyprelation berücksichtigt.

Sei im Folgenden f_* die neu zu registrierende Funktionalität eines Service-Mediators mit den Inputports $\mathcal{P}_{i*}$ und Outputports $\mathcal{P}_{o*}$. Sei ferner o.B.d.A. f eine bereits registrierte Funktionalität mit den Inputports $\mathcal{P}_i$ und Outputports $\mathcal{P}_o$. Dann werden die möglichen Verbindungen von $\mathcal{P}_o$ nach $\mathcal{P}_{i*}$ bzw. $\mathcal{P}_{o*}$ nach $\mathcal{P}_i$ unter Beachtung der Äquivalenz der Portnamen überprüft. Betrachten wir lediglich den ersten Fall, da der zweite Fall analog verläuft.

Betrachtung der Untertyprelation Seien o.B.d.A. $p_* \in \mathcal{P}_{i*}$ und $p \in \mathcal{P}_o$ zwei Ports, die der Äquivalenz der Portnamen genügen. Dann wird untersucht, ob die Datentypen sich hinsichtlich der *Untertyprelation* verknüpfen lassen. Dass heißt, es werden die Fälle

(1) $d \preceq d_*$ und

(2) $d \succ d_*$

geprüft, wobei d bzw. d_* die Datentypen von p bzw. p_* sind. Gilt Fall (1), dann lassen sich beide Ports verbinden. Im Fall (2) ist eine Verknüpfung eventuell möglich, aber nicht garantiert. Gilt keiner der beiden Fälle, dann ist keine syntaktische Verknüpfung möglich.

Der Schnitt $\mathcal{P}_{i*} \cap \mathcal{P}_o$ beinhaltet die Ports, die eine Verbindung aufgrund der Untertyprelation (Fall 1 und Fall 2) und der Äquivalenz der Portnamen ermöglichen. Sei n die Anzahl der Elemente dieses Schnitts. Sei $I_\preceq$ der Index, der die Anwendung der Untertyprelation bewertet. Dann ist $I_\preceq = \sum_{1 \leqslant j \leqslant n} l_j$, wobei l_j wie folgt berechnet wird:

$$l_j = \begin{cases} 0, \text{ falls } d_j \preceq d_{j*} \\ t, \text{ falls } d_j \succ d_{j*} \text{ und } t \text{ die Pfadlänge in der Typhierarchie ist.} \end{cases}$$

Betrachtung der Substitution der Ports Zur Bewertung der Verknüpfungen einer Funktionalität mit verschiedenen anderen Funktionalitäten, sind ferner die minimale Anzahl *benötigter* und *fehlender Ports* von Interesse (*Substitution der Ports*). Benötigte Ports sind die Ports, die eine konkrete Funktionalität zur Ausführung braucht. Hingegen beziehen sich fehlende Ports auf die Ports, die durch eine konkrete Funktionalität nicht abgedeckt werden können. Diese beiden Maßzahlen seien durch die Indexe $I_{\texttt{req}}$ und $I_{\texttt{miss}}$ erfasst. Dann gilt:

$$I_{\texttt{req}} = |\mathcal{P}_{i*} - (\mathcal{P}_{i*} \cap \mathcal{P}_o)| \text{ und } I_{\texttt{miss}} = |\mathcal{P}_o - (\mathcal{P}_{i*} \cap \mathcal{P}_o)|\,.$$

Syntaktisches Retrieval

Die während der Registrierung berechneten, unscharfen Verknüpfungen werden in der Retrievalphase eingesetzt, um *relaxed signature matches* zu ermöglichen. Das aufgespannte Operationsnetzwerk wird zudem eingesetzt, um Portzerlegungen und die Herleitung von Kompositionen zu ermöglichen.

Betrachtung der Portzerlegung Ausgehend von den Inputports des Anfrageprofils q wird eine minimale Anzahl von Funktionalitäten gesucht, die zusammen alle Inputports von q verarbeiten können (*Portzerlegung*). Damit werden alle Inputports des Anfrageprofils berücksichtigt. Sei k die minimal benötigte Anzahl. Ist eine Zerlegung möglich, kann dies die Vermutung nahelegen, dass einzelne Ports kontextunabhängig voneinander sind. Sei $I_{\texttt{part}}$ der Index, der den Grad der Zerlegung misst, dann ist $I_{\texttt{part}} = k - 1$, da mindestens eine Funktionalität benötigt wird.

Betrachtung der Komposition Werden durch die Anwendung einer einzelnen konkreten Funktionalität nicht alle Outputports von q direkt erreicht, wird versucht, weitere konkrete Funktionalitäten zu identifizieren, die mit den bereits ausgewählten konkreten Funktionalitäten kombiniert werden können. Der Index $I_{\texttt{comp}} = x - 1$ misst die Anzahl der so komponierten Funktionalitäten.

Das Resultat des Verfahrens zum Signatur-Matchmaking liefert eine konkrete Funktionalität bzw. eine Komposition von konkreten Funktionalitäten mit minimaler Distanz $d_{\texttt{SM}}$. Diese ergibt sich aus den verschiedenen Indexwerten:

$$\begin{aligned} d_{\texttt{SM}} \;=\; & \omega_{\texttt{comp}} \cdot I_{\texttt{comp}} + \omega_{\texttt{part}} \cdot I_{\texttt{part}} + \\ & \omega_{\texttt{miss}} \cdot I_{\texttt{miss}} + \omega_{\texttt{req}} \cdot I_{\texttt{req}} + \omega_{\preceq} \cdot I_{\preceq}, \end{aligned} \tag{5.5}$$

wobei die verschiedenen $\omega_{\cdot}$ Gewichtungsfaktoren repräsentieren.

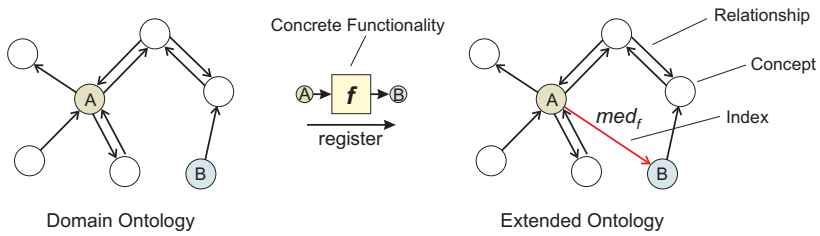

Abbildung 5.3: Ontologiebasierte Indexierung der Mediatorfunktionalitäten.

5.3.5 Konzept-Matchmaking

Das Verfahren zum Konzept-Matchmaking ist dem Verfahren zum Signatur-Matchmaking sehr ähnlich, besitzt aber auch fundamentale Unterschiede. Auch dieses Verfahren besteht aus zwei Teilalgorithmen, die sich in Registrierung und Retrieval aufspalten lassen. Während bei der syntaktischen Registrierung die *Äquivalenz der Portnamen* eingesetzt wird, um eine höhere Genauigkeit zu erzielen, ist dies für das semantische Retrieval nicht nötig. Gewöhnlich sind die semantischen Auszeichnungen eines Ports durch Konzepte einer Domänenontologie akkurater und aussagekräftiger als die verwendeten Portnamen. Sei f eine Mediatorfunktionalität mit $f : (i_f, \mathcal{P}_f^{\wp}) \rightarrow (o_f)$ und sei o.B.d.A. p ein erweiterter Port der Nachrichten i_f oder o_f mit $p = (\omega_p, d_p, C_p, desc_p)$ gemäß Definition 3.2, dann bezieht sich der Algorithmus zum Konzept-Matchmaking lediglich auf die Konzepte C_p der einzelnen Ports. In dem Algorithmus zum Konzept-Matchmaking wird ferner anstelle der Untertyprelation des Signatur-Matchmakings die Subsumptionsrelation eingesetzt.

Semantische Registrierung

Die Konzepte C_p einer konkreten Mediatorfunktionalität entstammen einer vorhandenen Domänenontologie. Diese Ontologie wird während der Registrierungsphase eines neuen Service-Mediators zur *Indexierung* der im Mediator enthaltenen Funktionalitäten eingesetzt. Genauer gesagt, zwischen den durch eine Funktionalität abgebildeten Konzepten wird eine entsprechende Relation *med.* in die Domänenontologie eingefügt, die auf die entsprechende Funktionalität verweist. Abbildung 5.3 veranschaulicht das Prinzip der *ontologiebasierten Indexierung*. Eine Funktionalität f, die zwischen den Konzepten A und B vermittelt, wird registriert. Hierzu wird eine Verbindung med_f zwischen den Konzepten A und B in die Ontologie eingetragen. Diese Verbindung verweist auf f. Die so *erweiterte Ontologie* spannt nun das Operationsnetzwerk für die semantische Suche auf. Im Gegensatz zu dem Operationsnetzwerk, welches bei der Registrierung des Verfahrens zum Signatur-Matchmaking aufgespannt wird, besitzt der Graph, der durch die erweiterte Ontologie beschrieben wird, keine „Lücken“ zwischen zwei Konzepten.

Da die Zuordnung eines Ports einer konkreten Funktionalität zu einem Konzept der Domänenontologie nicht exakt sein muss, erlaubt die Mediatorprofilsprache eine entsprechende

Abstufung, wie sie auch bereits für die Anwendungsklassen eingeführt wurde (vgl. Kapitel 4.5 und 5.3.2). In solchen Fällen sollen die ähnlichsten Konzepte genutzt und der Grad der Zugehörigkeit erfasst werden. Dieser Ansatz reduziert auch die Refactoring-Iterationen der Domännenontologie. Der Grad der Zugehörigkeit wird durch den bereits weiter oben definierten Bereich I ausgedrückt. Für jeden Port $p \in \mathcal{P}_i \cup \mathcal{P}_o$ einer konkreten Funktionalität ergibt sich damit der Grad der Abstufung g_p mit

$$g_p = \frac{1}{|C_p|} \sum_{c \in C_p} i_c,\ i_c \in I \text{ und } i_c \text{ Grad der Zugehörigkeit des Ports } p \text{ zum Konzept } c.$$

Die semantische Ähnlichkeit einer konkreten Mediatorfunktionalität f bezüglich der Konzepte der Domänenontologie lässt sich dann wie folgt definieren:

$$\omega_f = \sum_{p \in \mathcal{P}_i \cup \mathcal{P}_o} g_p.$$

Semantisches Retrieval

In der Retrievalphase wird die erweiterte Domänenontologie für den Matchmaking-Prozess in der Art ausgenutzt, dass der aufgespannte Graph traversiert wird, um einen Pfad zwischen den Konzepten der Ports des Anfrageprofils q zu bestimmen. Pfade spezifizieren dabei mögliche Kompositionen von Mediatorfunktionalitäten. Startknoten werden durch die Konzepte der Inputports, Zielknoten durch die Konzepte der Outputports von q definiert.

Während dieses Konzept-Matchmaking sind die Kanten des Graphen unterschiedlich zu bewerten, je nachdem, ob sie *is-a*, *med.*, *part-of* oder andere Beziehungen ausdrücken: *is-a* Beziehungen weisen auf die Subsumptionsrelation und *med.* auf indexierte Service-Mediatoren hin. *part-of* und andere Beziehungen zwischen zwei Konzepten ohne eine gleichzeitige *med.* Beziehung besagen, dass die Konzepte zwar semantisch zusammenhängen, aber zur Zeit kein Service-Mediator vorhanden ist, der diesen Zusammenhang „auflösen“ kann. Dies führt im Retrievalprozess zum Einsatz von *Proxy-Funktionalitäten*.

Betrachtung der Subsumptionsrelation Die Subsumptionsrelation ($\sqsubseteq$) kann ausgenutzt werden, um konkrete Mediatorfunktionalitäten zu identifizieren, die nicht direkt die Konzepte der Ports des Anfrageprofils sondern ein Unter- oder Oberkonzept anbieten. Sei c_* ein Konzept eines Inputports $\mathcal{P}_{i*}$ der Anfrage q und sei c ein Konzept eines Inputports $\mathcal{P}_i$ einer konkreten Mediatorfunktionalität f. Dann erfüllt f die Anforderungen von q, wenn $c_* \sqsubseteq c$. Ist $c_* \sqsupset c$, ist die Verknüpfung möglich, aber die korrekte semantische Verarbeitung nicht garantiert. Gilt keins von beiden, erfüllt f bzgl. c die Anforderung von q bzgl. c_* nicht.

Ähnlich zur Bewertung der Untertyprelation bei der syntaktischen Registrierung lässt sich auch die Anwendung der Subsumptionsrelation bewerten. Sei $I_{\sqsubseteq}$ der entsprechende Index.

Der Schnitt $\mathcal{P}_{i*} \cap \mathcal{P}_i$ beinhaltet in diesem Kontext nur die Ports, deren Konzepte über die Subsumptionsrelation verbunden sind, und sei m die Anzahl dieser Elemente. Dann ist $I_{\sqsubseteq} = \sum_{1 \leqslant j \leqslant m} h_j$, wobei h_j sich ergibt durch

$$h_j = \begin{cases} 0, \text{ falls } c_{j*} \sqsubseteq c_j \\ t, \text{ falls } c_{j*} \sqsupset c_j \text{ und } t \text{ die Pfadlänge in der Ontologie ist.} \end{cases} \tag{5.6}$$

c_j und c_{j*} entsprechen den Konzepten zweier Ports aus $\mathcal{P}_i$ bzw. $\mathcal{P}_{i*}$. Analog lässt sich dieses für die Outputports beschreiben.

Wie bereits in Abschnitt 5.3.1 diskutiert, ist die Verwendung der reinen Pfadlänge in einer Ontologie ein erster Versuch, die Distanz zwischen Konzepten, die der Subsumptionsrelation genügen, zu berechnen. Werden Ontologien verwendet, die in ihren Konzepten unterschiedliche Detailierungsgrade aufweisen, dann ist dieses Verfahren problematisch. Eine „Normierung“ über die Tiefe ist nicht möglich, da gewöhnlich komplexe Graphen durch eine Ontologie aufgespannt werden und nicht nur einfache Baumstrukturen.

Betrachtung der Proxy-Funktionalitäten Neben den *is-a* und *med.* Beziehungen können weitere Beziehungen zwischen den Konzepten der erweiterten Ontologie wie beispielsweise *part-of* Beziehungen existieren. Auch diese Informationen werden in dem Verfahren zum Konzept-Matchmaking genutzt.

Lässt sich beispielsweise bei einem Retrievalprozess zwischen zwei gegebenen Konzepten c und c' keine konkrete Mediatorfunktionalität finden, obwohl die beiden Konzepte in Beziehung zueinander stehen, wird an dieser Stelle eine so genannte *Proxy-Funktionalität* f_p angenommen, die einen Inputport mit Konzept c und einen Outputport mit Konzept c' besitzt, und der Retrievalprozess wird mit f_p fortgesetzt. Für derartige Proxy-Funktionalitäten generiert der Ansatz anschließend das entsprechend benötigte Mediatorprofil. Die Implementierung findet durch den Benutzer oder im Sinne eines Schema-Matchings durch ein System statt (vgl. den folgenden Abschnitt über Anpassung). Sei $I_{\texttt{proxy}}$ der Index, der die Anzahl der so eingefügten Proxy-Funktionalitäten erfasst.

Das Verfahren zum Konzept-Matchmaking liefert als Ergebnis eine konkrete Mediatorfunktionalität oder eine Komposition von Mediatorfunktionalitäten mit minimaler Distanz $d_{\texttt{CM}}$. Die Indexe für die Komposition, die benötigten und fehlenden Ports, sowie die Portzerlegung werden äquivalent zum Signatur-Matchmaking verwendet. Damit lässt sich $d_{\texttt{CM}}$ definieren durch

$$\begin{aligned} d_{\texttt{CM}} \;=\; & (\omega_{\texttt{comp}} + \omega_f) \cdot I_{\texttt{comp}} + (\omega_{\texttt{part}} + \omega_f) \cdot I_{\texttt{part}} + \omega_{\texttt{proxy}} \cdot I_{\texttt{proxy}} + \\ & \omega_{\texttt{miss}} \cdot I_{\texttt{miss}} + \omega_{\texttt{req}} \cdot I_{\texttt{req}} + \omega_{\sqsubseteq} \cdot I_{\sqsubseteq}, \end{aligned} \tag{5.7}$$

wobei die verschiedenen ω. Gewichtungsfaktoren repräsentieren.

Neben dem semantischen Retrieval durch dieses Matchmaking-Verfahren, kann die Domänenontologie auch zum manuellen *browsen* genutzt werden. Das heißt, der Benutzer navigiert durch die Ontologie, um selbst geeignete Mediatorfunktionalitäten zu identifizieren.

5.4 Adaption – Anpassung von Service-Mediatoren

Durch den Discovery-Prozess der vorherigen Phase erhält der Benutzer eine Liste von atomaren bzw. komplexen Service-Mediatoren, die teilweise im vorliegenden Workflow sinnvoll eingesetzt werden können. Diese liefern den *Input* für die Phase der Anpassung. In diesem Abschnitt steht die konkrete Anpassung der identifizierten Service-Mediatoren im Vordergrund. Durch die Anpassung der Mediatoren soll die Service-Interoperabilität innerhalb des Workflows garantieren werden (*Verarbeitung*). Das Resultat der Anpassung liefert als *Output* einen konfigurierten Service-Mediator bzw. eine neue Komposition von Service-Mediatoren.

Probleme bei der Adaption

Die durch das Retrieval gewonnenen Service-Mediatoren und Kompositionen von Service-Mediatoren erzielen die geforderte Service-Interoperabilität nicht zwingend. Zum einen liegt dies daran, dass sowohl das Discovery als auch die Anfragegenerierung unscharf sind, zum anderen, dass während der Discovery-Phase teilweise Service-Mediatoren „identifiziert“ werden, die nicht im Mediatorpool vorhanden sind: Proxy-Funktionalitäten.

Semiautomatische und manuelle Adaption

In Kapitel 4.3.2 wurden die konkreten Anpassungsmöglichkeiten der Service-Mediatoren bereits vorgestellt. Hier werden sie nun aus Sicht der QDAS-Prozedur weiter diskutiert.

- *Selektion*: Die Selektion einer atomaren oder komplexen Funktionalität ist der erste Anpassungsschritt in der Adaptionsphase. Das Discovery der vorherigen Phase liefert hierzu eine abgestufte Liste von Funktionalitäten, die die Anforderungen des Anfrageprofils weitestgehend erfüllen. Diese Selektion geschieht anfangs manuell unter Berücksichtigung der definierten Schranken und Gewichte. Werden später in einem anderen Teilworkflow exakt die gleichen zwei Services miteinander verbunden, geschieht die Selektion automatisch aufgrund des vorhandenen Vorwissens. Sind die Services sehr *ähnlich*[3], wird eine entsprechende Selektion dem Nutzer vorgeschlagen.

- *Konfigurierung*: Die Konfigurierung ist eine weitere Möglichkeit einen Service-Mediator den konkreten Bedürfnissen entsprechend anzupassen. In dem vorgestellten Ansatz werden zustandsbehaftete Eigenschaftsfelder für diesen Zweck eingesetzt. Die Zustandsbehaftung erlaubt getätigte Konfigurationen persistent abzulegen, wodurch

[3]Die Phase der Anfragegenerierung liefert hinsichtlich der genutzten Konzepte *äquivalente* Anfrageprofile. Äquivalent heißt in diesem Zusammenhang, dass die Konzepte der Input- bzw. Outputports eines Profils über *is-a* Beziehungen mit den Input- bzw. Outputports eines anderen Profils verbunden sind.

sie für zukünftige Workflows direkt zugänglich sind. Derzeit wird die Konfigurierung manuell durchgeführt. Zukünftig könnte durch eine geeignete Annotierung der Property-Werte über die Domänenontologie die Konfigurierung teilweise automatisiert vonstatten gehen.

- *Identifizierung und Modifikation von Kompositionen*: Durch den Discovery-Prozess werden mögliche Kompositionen von Funktionalitäten auf Basis der syntaktischen oder semantischen Information vorgeschlagen bzw. bereits vorhandene, komplexe Funktionalitäten aus dem Mediatorpool identifiziert. Ferner kann ein Experte eigenständig neue Kompositionen erstellen, ohne dabei einen konkreten Workflow zu betrachten. Hierzu kann er die verschiedenen Matchmaking-Verfahren nutzen, um geeignete Service-Mediatoren für seine Komposition zu ermitteln. Wird eine Komposition vorgeschlagen oder aus dem Pool identifiziert, kann der konkrete Workflow es bedingen, diese entsprechend den Anforderungen zu modifizieren. Da komplexe Funktionalitäten ebenfalls in anderen Kompositionen teilnehmen können und so hierarchische Kompositionen entstehen, muss bei der Modifikation einer bereits im Mediatorpool abgelegten Funktionalität das so entstehende *fragile base class problem* berücksichtigt werden (Szyperski et al., 2002). Aus diesem Grund werden die Service-Mediatoren in diesem Ansatz geklont, bevor sie angepasst werden. Damit bleiben komplexe Funktionalitäten gültig.

- *Erstellung neuer Mediatoren*: In bestimmten Fällen müssen neue Service-Mediatoren bzw. Mediatorfunktionalitäten entwickelt werden, wie beispielsweise im Falle der Proxy-Funktionalitäten. Der entwickelte Ansatz unterstützt den Benutzer indem benötigte WSDL und MPL-Schnittstellen sowie Programm-Skeletons in Java generiert werden. Auch hier könnten zukünftig erweiterte Ansätze, die beispielsweise Schema-Matching nutzen, in die software-unterstützte Prozedur integriert werden. Bei einfachen Matchings könnten auch interaktive Verfahren gewinnbringend eingesetzt werden. Hierzu wären entsprechende Benutzerschnittstellen notwendig.

5.5 Architektur des IRIS-Frameworks

Die vorgestellte Prozedur zur Service-Interoperabilität wird durch das im IRIS-Projekt[4] realisierte Framework unterstützt (Radetzki et al., 2003, 2002a). Dieses Framework ist durchgängig in Java realisiert. Die Kernkomponenten dieser Softwarearchitektur sind in Abbildung 5.4 illustriert. Prinzipiell lassen sich fünf übergeordnete Softwaremodule unterscheiden:

- Das *Management-Modul* beinhaltet die Komponenten, die die oben beschriebene software-unterstützte Prozedur abbilden.

[4] *Interoperability and Reusability of Internet Services* (IRIS)

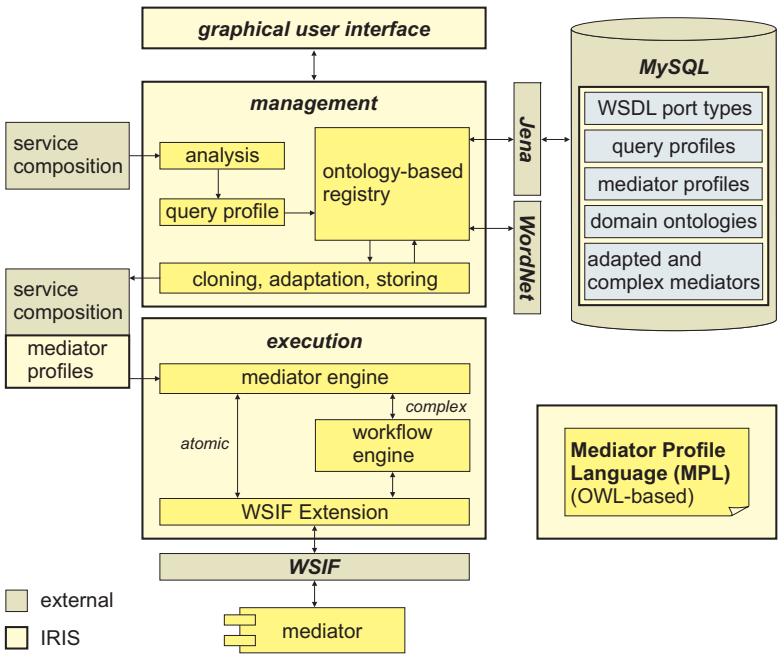

Abbildung 5.4: Architektur des IRIS-Frameworks.

- Das *Execution-Modul* besitzt Softwarekomponenten zur Ausführung atomarer und komplexer Funktionalitäten realisierter Service-Mediatoren. Dies beinhaltet unter anderem eine Workflow-Engine, die die Kompositionsgraphen der komponierten Funktionalitäten abarbeitet.

- Das *Datenbank-Modul*, zu dem auch der Mediatorpool gehört, dient der Ablage der verschiedenen Informationen.

- Die *grafische Benutzerschnittstelle* (GUI) ermöglicht die Administrierung und Kontrolle der anderen Module. Sie wird in Kapitel 6 im Rahmen eines Fallbeispiels genauer beschrieben.

- Die in OWL beschriebene *Mediatorprofilsprache* MPL erlaubt die Spezifikation der Service-Mediatoren.

Der Entwurf und die Beschreibung eines Web Service basierten Workflows, d.h. die Abbildung der Geschäftsprozesse durch BPEL, wird nicht durch das IRIS-Framework unterstützt und liegt außerhalb des Fokus dieser Plattform. Hierzu existieren verschiedene Middleware-Plattformen, die diese Aufgabe übernehmen, wie beispielsweise Oracle's BPEL Process Manager (vgl. Kapitel 2.4).

Die Elemente des Management-Moduls und ihre Funktionsweisen wurden bereits ausgiebig in diesem Kapitel diskutiert, während die Mediatorprofilsprache Gegenstand des Kapitels 4.5 war. Die grafische Benutzerschnittstelle wird im Rahmen der Case Study in Kapitel 6 näher erläutert.

Die Ausführung einer Mediatorfunktionalität geschieht durch das Execution-Modul. Die *Mediator Engine* liest das entsprechende Profil ein und eruiert, ob es sich bei der auszuführenden Funktionalität um eine atomare oder komplexe bzw. konfigurierte Funktionalität handelt. Abhängig von dieser Entscheidung wird das WSDL-Dokument des Service-Mediators direkt an die WSIF Extension geleitet, oder es wird die Workflow Engine mit der komplexen bzw. konfigurierten, in MPL beschrieben Funktionalität aktiviert. Die *Workflow Engine* arbeitet die Kompositionsgraphen der komplexen Funktionalität ab, bis sie ihrerseits auf weitere auszuführende Funktionalitäten trifft. Sobald dies der Fall ist, werden die entsprechenden Informationen zurück an die Mediator Engine geleitet und die Abarbeitung wird fortgesetzt, bis die komplette Komposition abgearbeitet ist. Mit Hilfe der WSIF Extension setzt die Workflow Engine zudem die Eigenschaftsfelder konfigurierter Mediatorfunktionalitäten.

Die *WSIF Extension* erweitert das *Web Services Invocation Framework* (WSIF)[5] um die folgenden Fähigkeiten: Zum einen wird das Setzen und Abfragen von Properties explizit unterstützt, zum anderen verbessert die Erweiterung das vorhandene Java-Binding um die Fähigkeit, mehrere Outputattribute spezifizieren zu können. Dieses wird gewöhnlich in Java nicht direkt unterstützt. WSIF ist eine offene Architektur, die es erlaubt, zukünftig auch für andere Sprachen und Komponentenmodelle Bindungen zu spezifiziert.

Die relevanten Daten, die innerhalb der IRIS-Plattform benötigt werden, werden zum einen in einem relationalen Datenbankmanagementsystem (RDBMS) abgelegt, zum anderen liegen sie aber auch zusätzlich direkt als WSDL- bzw. OWL-Dokumente vor. Der Vorteil des RDBMS liegt in der theoretischen Fundierung und Praxistauglichkeit, sowie in dem Vorhandensein einer standardisierten Anfragesprache (SQL). Zudem sind verschiedene Indexstrukturen Kernbestandteil moderner RDBMS. Zugriff auf die relationale Datenbank geschieht durch die *Java Database Connectivity* (JDBC), während die Verarbeitung OWL-basierter Dokumente durch das offene *Jena-Framework*[6] geschieht. Jena ist ein Java Framework zur Realisierung von Semantic Web Anwendungen. Jena ermöglicht dem IRIS-Framework einen Zugriff und ein Speichern der Domänenontologien sowie der Mediatorprofile. Linguistische Analysen werden mit Hilfe der *WordNet 2.0*[7] Datenbank durchgeführt.

[5] `ws.apache.org/wsif/`
[6] `www.hpl.hp.com/semweb/jena.htm`
[7] `wordnet.princeton.edu/`

5.6 Herausforderungen und verwandte Ansätze

Herausforderungen der Anfragegenerierung

Eine große Herausforderung stellt die Abbildung der vorhandenen WSDL-Dokumente auf die Konzepte einer Domänenontologie dar. Konzept-Matching hängt zu aller erst stark von der gegebenen Qualität der vom Requester und Provider angebotenen Informationen ab (Lutz et al., 2003). Selbst mit linguistischen Methoden wird die Gleichheit der verwendeten Terme benötigt. Dies ist ein beträchtliches Problem, wenn Abkürzungen oder verschiedene Schreibweisen verwendet werden oder Schreibfehler in den Termen vorhanden sind. Hier könnten beispielsweise die Hamming-Distanz oder die Levenshtein-Distanz eingesetzt werden, um Missmatches aufgrund von Schreibfehlern zu überwinden. Dong et al. (2004) beschreiben ein Clusterverfahren, das die Ähnlichkeit von Termen auf Basis von Assoziationsregeln bestimmt. So erstellte Cluster könnten mit Konzepten der Domänenontologie in Beziehung gesetzt werden, so dass ein Cluster einem Konzept entspricht. Auf dieser Grundlage könnte die Distanz eines Terms zu einem Cluster bestimmt und damit der Term einem Konzept zugeordnet werden, ohne dass hierbei die Gleichheit der Terme erzwingt wird.

Die Analyse und Abbildung der Elemente der WSDL-Dokumente auf die Ontologie geschieht gegenwärtig zur Entwurfszeit. Damit lassen sich *generische Typen* nur schwer identifizieren. Ein Algorithmus, der zur Laufzeit die Ausführung des Workflows überwacht, könnte die Instanzinformationen analysieren und so eventuell exaktere Mappings beschreiben.

Die Anwendungsdömane und die Anwendungsklasse lassen sich nur bedingt aus den WSDL-Dokumenten der Services herleiten. Wird ein Service in dem Workflow über eine UDDI-Beschreibung (Oas, 2003) identifiziert oder ist der Service nicht durch WSDL sondern durch OWL-S (Martin et al., 2004) spezifiziert, kann die Domäneninformation hier herausgelesen werden. Die Anwendungsklasse eines Service-Mediators beschreibt die Klasse der Methodik. Welche Klasse in einem Workflow benötigt wird, hängt dabei vom Bedarf des Nutzers ab und dem Ziel, welches der Workflow erreichen soll. Die Beschreibung der Ziele, d.h. warum in einem Workflow zwei Services verknüpft werden und welche konkrete Methodik zwischen den Services benötigt wird, ist ein offenes Problem. Daher müssen die Anwendungsklassen momentan manuell angegeben werden.

Herausforderungen des Discovery

Durch die Bestimmung möglicher neuer Kompositionen von Mediatoren beim Matchmaking müssen wir uns zukünftig der Herausforderung stellen, zu entscheiden, welche Komposition hinsichtlich der Anforderungen des Workflows „korrekter“ ist, wenn verschiedene Kompositionen möglich sind. Beispielsweise, wenn zwei Funktionalitäten die gleichen Input- und Outputdaten verarbeiten, jedoch semantisch unterschiedliche Berechnungsfunktionen

verwenden. Nach Beendigung der software-unterstützten Prozedur werden alle relevanten Informationen im Bezug zu dem Workflow gespeichert und stehen somit für folgende Workflows zur Verfügung. Daher könnte ein zukünftiger Weg sein, aus den vorhandenen Informationen auf Kompositionen neuer Workflows zu schließen.

Alle Matchmaking-Verfahren lassen sich über Gewichte den konkreten Anforderungen des Benutzers anpassen. Die Standardeinstellungen der Gewichte sind zur Zeit rein willkürlich. Zukünftig sollten diese Einstellungen experimentell bestimmt werden, um „optimale“ Retrievalergebnisse zu erzielen. Diese Bestimmung ist zur Zeit nicht möglich, da keine entsprechenden oder nur unzureichende Testsets und Benchmarks für Web Services bzw. Service-Mediatoren vorhanden sind. Auch bei der Erstellung entsprechender Testsets und Benchmarks ist zukünftig Arbeit zu leisten.

Zu diesem Zeitpunkt ist das Discovery und das Matchmaking von Services bzw. von Agenten im Allgemeinen mit hoher *Precision* und *Recall* ein offenes Forschungsfeld im Bereich des Information Retrieval und der Agententechnologie. Die Anforderung auch Kompositionen zu berücksichtigen, stellt eine zusätzliche Herausforderung dar.

Im Bereich der Web Services Technologie ist UDDI der derzeitige Retrieval- und Registry-Ansatz. UDDI stellt einen Verzeichnisdienst für Serviceinformationen bereit (Keidl et al., 2003). Hier können Metadaten, Taxonomien, und Schnittstellen angelegt und angefragt werden. Die Schnittstellen der Services werden lediglich syntaktisch durch WSDL beschrieben. Sowohl WSDL als auch UDDI mangelt es an expliziter Semantik (Paolucci et al., 2002). Daher ist eine Identifizierung von Services oder Service-Mediatoren im Bezug auf semantische Anforderungen nicht möglich.

Hier versuchen derzeitige Ansätze aus dem Bereich des Semantic Web Lösungen zu finden, die auf spezielle, manuell annotierte Web Services aufsetzen. Vor allem Arbeiten um OWL-S sind hier zu nennen (Martin et al., 2004; Paolucci et al., 2002). Ist die manuelle Auszeichnung über eine Domänenontologie abgeschlossen und sind die Services in OWL-S beschrieben, kann diese Ontologie für das Discovery (Paolucci et al., 2002) und die Komposition von Services eingesetzt werden (McIlraith et al., 2001; Narayanan und McIlraith, 2002). Im Gegensatz zu den OWL-S Ansätzen setzt die hier vorgestellte Discovery Einheit auf rein in WSDL beschriebene Services auf und stellt ein automatisches Mapping vor (vgl. Anfragegenerierung). Ferner bieten bestimmte Matchmaking-Verfahren der Discovery-Einheit Filtereigenschaften an. Im Gegensatz zu (Paolucci et al., 2002) ermöglicht der hier entwickelte Algorithmus zum Konzept-Matchmaking eine kooperative Anpassung über verschiedene Gewichte und die „Identifizierung“ von benötigten, aber nicht vorhandenen Modulen in einer Komposition (Proxy-Funktionalitäten).

Das kürzlich vorgestellte *Woogle*-System basiert ebenfalls auf rein in WSDL beschriebenen Web Services und nutzt ein Clusterverfahren zur Ähnlichkeitssuche von Serviceoperationen (Dong et al., 2004). Die Schlüsselidee des Clusterverfahrens basiert auf der Heuristik, dass Parameternamen, die häufig zusammen auftreten (*co-occurrence*), dazu neigen das gleiche semantische Konzept auszudrücken und somit dem gleichen Cluster angehören.

In diesem Ansatz werden Parameternamen mit einer entsprechenden Vorverarbeitung in Terme zerlegt, über denen anschließend Assoziationsregeln definiert werden. Diese Regeln werden bei dem Clustering ausgenutzt, indem in greedy-Manier *bottom-up* aus einelementigen Clustern größere Cluster zusammengemischt werden.

Das vorgestellte Woogle-Verfahren kann gut mit den hier entworfenen Methoden des Konzept-Matchmakings kombiniert werden. Die semantischen Konzepte bei Woogle, die durch die verschiedenen Cluster definiert werden, sind nicht hierarchisch angeordnet. Eine mögliche Erweiterung wäre die Cluster durch Konzepte einer Dömänenontologie zu annotieren. Auf diese Weise könnte man von Beziehungen zwischen den Konzepten, vor allem der Subsumptionsrelation, beim Matchmaking profitieren.

Einige der vorgestellten Arbeiten gehen im wesentlichen auf Grundlagen zurück, die im Bereich der Agententechnologie bzw. im Bereich der Komponententechnologie erzielt wurden (Klusch und Sycara, 2001; Zaremski und Wing, 1997). Ein zentrales Anliegen innerhalb der Multi-Agentenplattformen ist die Suche nach Agenten, die zur Erreichung eines Aufgabenzieles eines anderen Agenten eine bestimmte Funktionalität bereitstellen (dynamische Suche und Anwendung der Agenten). In diesem Kontext wurden verschiedene Plattformen und Sprachen vorgeschlagen. Prominente Arbeiten sind beispielsweise *LARKS* (Sycara et al., 1999, 2002) und *KRAFT* (Preece et al., 2000). Im Gegensatz dazu werden bei der komponentenbasierten Entwicklung konkrete Komponenten gesucht, die in eine Softwarearchitektur integriert werden können. Hier steht der Aspekt der Wiederverwendung im Zentrum. Die Anforderungen an die Matchmaking-Verfahren sind aber ähnlich.

Zaremski und Wing (1997) beschreiben in ihrem Ansatz eine Reihe von Matching-Varianten für Komponenten. Softwarekomponenten werden hierbei als Module verstanden, die eine Menge von Funktionen beinhalten. Jede Komponente wird durch ihre Signatur und durch eine Spezifikation des Programmverhaltens ausgezeichnet. Letztere werden durch Prä- und Postkonditionen der Schnittstelle spezifiziert.

LARKS ist eine Beschreibungssprache um die Fähigkeiten eines Agenten ausdrücken zu können (Sycara et al., 1999, 2002). Basierend auf LARKS wurde ein Matchmaking-Prozess definiert, der verschiedene Filter nutzt. Diese Filter können durch den Benutzer kombiniert werden, um geeignete Agenten zu identifizieren. Neben dem reinen Signatur-Matching wird auch hier ein *Constraint Matching* beschrieben, wobei Vor- und Nachbedingungen über einer gemeinsam genutzten Ontologie spezifiziert werden. Diese Ansatz ist mit KRAFT verwandt (Preece et al., 2000).

KRAFT ist eine agentenbasierte Mediatorplattform zur Integration heterogener, verteilter Datenquellen. In KRAFT werden alle wissensverarbeitenden Komponenten (z.B. Datenbanken oder Integrationseinheiten) sowie Benutzeraktionen durch Softwareagenten realisiert. Semantische Diskrepanzen zwischen verschiedenen Datenquellen werden durch eine gemeinsam genutzte Ontologie überwunden. Dazu wird für jede lokale Datenquelle eine eigenständige lokale Ontologie definiert, die anschließend mit der gemeinsam genutzten Ontologie in Verbindung gebracht wird. Die dabei auftretenden ontologischen Mismatches

werden weitestgehend durch Ontology-Mappings überwunden. Die Fähigkeiten der Agenten werden durch Bedingungen (*constraints*) in Abhängigkeit der gemeinsam genutzten Ontologie definiert. Mit dieser Beschreibung werden die Agenten dann bei dem *Facilitator*, einem *Yellow-Pages Service*, registriert. Der Facilitator ermitteln unter Angabe verschiedener *Constraints* möglichst gute Agenten, die die Bedürfnisse des Anfragenden am weitesten erfüllen.

Keiner der vorgestellten Verfahren unterstützt die Komposition von Komponenten. Ferner sind Beschreibungen von Vor- und Nachbedingungen gegenwärtig nicht im Web Services Standard, d.h. in WSDL, vorhanden und lassen sich nicht ohne weiteres herleiten. Auf der anderen Seite können Mediatorfunktionalitäten Berechnungen realisieren, die aufgrund ihrer Komplexität nur ungenügend, wenn überhaupt, durch logische Vor- und Nachbedingungen repräsentiert und effizient bearbeitet werden können. Da in dieser Arbeit ebenfalls eine gemeinsam genutzte Ontologie eingesetzt wird, wäre es dennoch prinzipiell möglich ein Constraint-basiertes Retrievalverfahren als weitere Discovery-Einheit in IRIS zu integrieren. Aus den oben geschilderten Gründen wurde dieser Weg jedoch nicht beschritten.

Herausforderungen bei der Anpassung

Die Anpassung identifizierter Service-Mediatoren hat verschiedene Facetten. Zentral sind die Komposition und die Konfigurierung. Während der Komposition kann es zudem geschehen, dass bestimmte Funktionalitäten benötigt werden, die zur Zeit der Komposition nicht im Mediatorpool enthalten sind (Proxy-Funktionalitäten). Diese gilt es entsprechend zu realisieren. Hierzu könnten zukünftig beispielsweise Schema-Matching Ansätze in den Prozess integriert werden. Rahm und Bernstein (2001) liefern einen Überblick über gegenwärtige Methoden im Bereich des Schema-Matchings, die auch mit Ontologien verknüpft werden können (Bowers und Ludäscher, 2004). Neuere Verfahren versuchen zudem auf Basis früher durchgeführter Match Operationen bessere Resultate beim Schema-Matching zu erzielen (Do und Rahm, 2002; Madhavan et al., 2003). Diese Verfahren werden auch unter dem Stichwort *reuse-oriented strategies* zusammengefasst (Do und Rahm, 2002). Ergänzende Informationen zum Thema Schema-Matching finden sich im Kapitel 3.3 zu den vergleichenden Interoperabilitätsansätzen hinsichtlich der a posteriori Verfahren. Bei einfachen Matchings könnten auch interaktive Verfahren gewinnbringend eingesetzt werden. Hierzu wären entsprechende Benutzerschnittstellen zu realisieren. Die Konfigurierung der atomaren und komplexen Mediatorfunktionalitäten muss derzeit manuell durchgeführt werden. Zukünftig wäre es wünschenswert, wenn auch hier eine (semi-)automatische Anpassung vom System durchgeführt würde. Dieses könnte eventuell dadurch erreicht werden, dass die möglichen Werte der Eigenschaftsfelder auf die vorhandenen Beziehungen der Konzepte der verwendeten Ontologie abgebildet werden. Bei der Komposition durch das Konzept-Matchmaking könnten dann diese Werte direkt verwendet werden. Dieses würde jedoch mehr Aufwand bei der Definition der Mediatorprofile bedeuten.

5.7 Zusammenfassung

Die mediatorbasierte Service-Interoperabilität eines Workflows wird durch den geschickten Einsatz von Service-Mediatoren erzielt. Hierzu müssen vorher die konkreten Workflows analysiert und entsprechende Service-Mediatoren identifiziert werden. Die in dieser Arbeit entwickelte semiautomatische, software-unterstützte Prozedur adressiert genau diese Fragestellung. Genauer gesagt, es wird ein Verfahren vorgestellt, das den Übergang eines nicht-interoperablen Workflows W hin zu einem mediator-interoperablen Workflow W' ermöglicht. Diese Prozedur gliedert sich in vier Phasen, die durch das IRIS-Framework prototypisch umgesetzt werden:

1. Anfragegenerierung,
2. Discovery,
3. Adaption und
4. Bekanntmachung.

Aufgrund der WSDL-Beschreibungen der Services, die lediglich syntaktische sowie unstrukturierte Informationen bereitstellen, gestaltet sich gerade die Anfragegenerierung und die anschließende Suche als extrem schwierig. Der entwickelte Ansatz nutzt ontologische Mittel und linguistische Ansätze, um aus den WSDL-Dokumenten entsprechende Anforderungen zu bestimmen und konkrete Service-Mediatoren aufzufinden. Dieses bedingt zum einen eine entsprechende Domänenontologie und zum anderen eine initiale Menge essentiell benötigter Service-Mediatoren.

Die Algorithmen zum Signatur- und Konzept-Matchmaking sind die zentralen Bausteine des Discovery-Prozesses. Sie ermöglichen neben der Identifizierung eines einzelnen Service-Mediators auch die Entdeckung neuer Kompositionen auf Basis der Datentypen und der Untertyprelation bzw. der ontologischen Konzepte und der Subsumptionsrelation. Es zeigt sich aber, dass die Signatursuche, die Portnamen inklusive möglicher Synonyme einsetzt, zu restriktiv ist und mögliche Service-Mediatoren als Trefferkandidaten nicht erkennt. Sie ist daher nur gut einsetzbar für klar strukturierte Bereiche und *inhouse*-Anwendungen, bei denen Services und Service-Mediatoren aus ein und derselben Hand sind. Die semantische Suche auf Basis der Domänenontologie ist flexibler, da die Beziehungen der Konzepte beim Retrieval eingesetzt werden können. Auf diese Weise können auch Service-Mediatoren beschrieben werden, die zur Zeit des Retrievals noch nicht realisiert sind (Proxy-Funktionalitäten). Die Verfahren zum Konzept-Matchmaking benötigen auf der anderen Seite aber eine entsprechend gute Abbildung der Terme der WSDL-Dokumente auf die Domänenontologie.

Kapitel 6

Fallstudie

Die Effektivität und Robustheit der QDAS-Prozedur wird in einer ersten Fallstudie untersucht. Im Kern der Untersuchung stehen die Phasen der Anfragegenerierung und des Discovery, da diese ausschlaggebend für die Wiederverwendbarkeit der Service-Mediatoren sind. Es konnte gezeigt werden, dass für einen konkreten Workflow angepasste Service-Mediatoren diese auch bei sukzessiv modifizierten Workflows mit hoher Precision und Recall wiedererkannt werden. Ferner wird das Konzept-Matchmaking auf die semantische Suche nach Web Services Operationen übertragen. Hier zeigen die Versuche eine klare Überlegenheit des Konzept-Matchmakings gegenüber Standardverfahren wie dem Vector Space Model im Zusammenhang mit TF-IDF.

In diesem Kapitel wird die QDAS-Prozedur zur Erzielung der Service-Interoperabilität in der Praxis verdeutlicht, die den Arbeitsaufwand des Benutzers bei der Erstellung *interoperabler* Workflows verringert. Es wird zudem auf den Aspekt der Entwicklung neuer Service-Mediatoren durch den Anwender eingegangen. Zur Illustrierung wird in Abschnitt 6.1 ein einfaches *in silico* Experiment vorgestellt, anhand dessen der Einsatz der Service-Mediatoren sowie der entwickelten Verfahren praxisbezogen erörtert werden.

In Abschnitt 6.2 wird auf den Aspekt der initialen Ausprägung des Mediatorpools sowie der Erstellung der Domänenontologie eingegangen. Das konkrete Vorgehen zur Erstellung eines neuen Service-Mediators ist Gegenstand des Abschnitts 6.3. In diesem Kontext wird auch das entwickelte Anwendungswerkzeug des IRIS-Frameworks vorgestellt.

Die konkrete Durchführung der QDAS-Prozedur, die auf Basis des Anwendungsbeispiels stattfindet, wird in Abschnitt 6.4 beschrieben. Dieses Beispiel wird in darauffolgenden Versuchen variiert, um die Robustheit und Effektivität des Verfahrens zu analysieren (Abschnitt 6.5). In Abschnitt 6.6 wird in einem letzten Experiment das entwickelte Verfahren auf das Problem der semantischen *Web Services Discovery* übertragen. Das Kapitel endet mit einer kurzen Zusammenfassung.

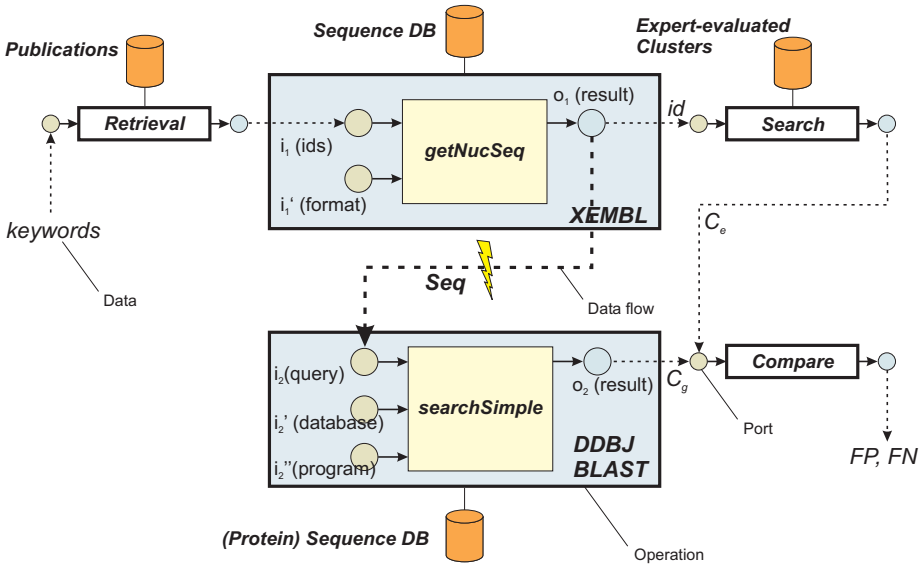

Abbildung 6.1: Beispielhafter Workflow eines vereinfachten *in silico* Experiments.

6.1 Anwendungsfall – Ein *in silico* Experiment

Einen zentralen Stellenwert in den Biowissenschaften und den angrenzenden informatikbezogenen Wissenschaften nimmt die Identifizierung von Genen und Proteinen sowie deren Funktionen ein. Betrachten wir in diesem Kontext als Anwendungsszenario ein einfaches *in silico* Experiment, welches, basierend auf einer Menge von Schlüsselwörtern, mögliche Proteine bzw. Gene identifiziert, die eventuell mit der durch die Schlüsselwörter angegebenen Funktion in Zusammenhang stehen (*Kandidatenproteine* bzw. *Kandidatengene*).

Dieses Experiment sei durch einen Workflow über einer Menge von Services beschrieben und diene als Input für die QDAS-Prozedur zur Erzielung der Service-Interoperabilität. Es wird davon ausgegangen, dass jeder Dienst als WSDL-basierter Web Service zur Verfügung steht und alle Dienste autonom und unabhängig voneinander entwickelt wurden. Aufgrund der zweiten Annahme sind die verwendeten Datentypen und Schnittstellen syntaktisch und semantisch unterschiedlich, wodurch die Service-Interoperabilität des gesamten Workflows *a priori* nicht gegeben ist.

Abbildung 6.1 zeigt die Operationen der teilnehmenden Dienste und den Datenfluss des Workflows. Der Workflow beginnt mit einer Literaturrecherche, in der Publikationen identifiziert werden, die angegebene Schlüsselwörter beinhalten. Dieser Service kann beispielsweise über Werkzeuge wie PubMed Central[1], Medline[2] oder Entrez[3] realisiert werden.

[1] `www.pubmedcentral.gov/index.html`
[2] `medline.cos.com/`
[3] `www.ncbi.nlm.nih.gov/entrez/query.fcgi`

Ausgehend von den in den Publikationen benannten Genen oder Proteinen sollen nun die kompletten annotierten Sequenzinformationen den nachfolgenden Services des Workflows bereitgestellt werden. Beispielsweise liefern GenBank, EMBL oder DDBJ diese Informationen, jedoch in unterschiedlichen Formaten. Der entsprechende Dienst benötigt für die Bereitstellung der annotierten Sequenzinformationen einen Identifikator (z.B. die Akzession-Nummer) des Gens oder Proteins, der jedoch erst aus den Publikationen heraus identifiziert werden muss.

Mit den annotierten Sequenzinformationen soll nun die Identifikation neuer Kandidatenproteine bzw. -gene beginnen, indem Cluster verschiedener Datenbanken und Algorithmen verglichen werden. Exemplarisch werden zwei verschiedene „Cluster-Provider“ herausgenommen. Zum einen wird von einer Datenbank ausgegangen, die Cluster von Proteinen bereitstellt, wobei Biologen das Clustering auf Basis ihres Hintergrundwissens erstellten. Diese expertenevaluierten Cluster C_e werden beispielsweise durch die PIR Datenbank[4] zur Verfügung gestellt. Zum anderen sei ein weiteres Clustering von Proteinen algorithmisch hergeleitet (Cluster C_g). Beispielsweise könnten die Cluster C_g durch systematische Iteration einer BLAST-Suche generiert werden. Hier haben Krause und Vingron (1998) ein entsprechendes mengentheoretisches Verfahren mit dem Namen *Systers* vorgeschlagen[5]. Dieses Clustering betrachtet alle verfügbaren Proteine einer Proteinsequenzdatenbank. Dabei werden eventuell Cluster generiert, die nicht die Genauigkeit der Cluster C_e haben, da die expertenevaluierten Cluster C_e als exakter angenommen werden. Die „Cluster-Provider“ benötigen zur Bereitstellung der Cluster unterschiedliche Informationen. Der Provider für die Cluster C_e benötigt einen entsprechenden Identifikator, während die iterative BLAST-Suche über der konkreten Sequenzinformation arbeitet.

Basierend auf der Analyse der beiden unterschiedlichen Cluster von Proteinen bzw. Genen sollen nun die möglichen neuen Kandidatenproteine bzw. -gene für die angegebene Funktion identifiziert werden. Auch hier kann man davon ausgehen, dass die beiden Cluster in unterschiedlichen Datenformaten vorhanden sind.

Ein letzter Service (*Compare*) biete nun die Möglichkeit auf zwei beliebigen Datenmengen, die in einem bestimmten Format vorliegen, Mengenoperationen durchzuführen. Beispielsweise Schnitt, Vereinigung, Differenz, etc. Dieser Service soll nun genutzt werden, um einen Vergleich der beiden Cluster C_e und C_g durchzuführen. Hierzu müssen die Clusterinformationen der Cluster C_e und C_g auf das entsprechende Format des Services transformiert werden.

Der Vergleich liefert aus Sicht von C_g *falsch positive* (FP) und *falsch negative* (FN) Proteine bzw. Gene, die für weitere Labor- oder *in silico* Experimente herangezogen werden können. Ein Protein wird als *falsch positiv* bezeichnet, wenn es in C_g aber nicht in C_e vorhanden ist, und als *falsch negativ*, wenn es in C_e jedoch nicht in C_g enthalten ist. Wird ein Protein als FP klassifiziert, kann es tatsächlich dem falschen Cluster zugeordnet sein

[4] `pir.georgetown.edu/home.shtml`
[5] Mittlerweile auch als eigenständige Datenbank unter `systers.molgen.mpg.de` verfügbar.

oder es steht mit der diskutierten Funktion in Beziehung (*Kandidat*), wobei dieses bis dato nicht bekannt war (daher ist es nicht in C_e vorhanden). Dieser Kandidat muss im Laborversuch überprüft werden. Wird im Gegensatz dazu ein Protein als *FN* eingestuft, kann diese Information genutzt werden, um das Verfahren zur Generierung des Proteinclusters C_g zu optimieren. Beispielsweise indem Gewichte oder Schranken des algorithmischen Verfahrens modifiziert werden.

6.2 Initiale Ausprägung und Domänenontologie

Der geschilderte Anwendungsfall verdeutlicht, dass die beschriebenen Services der Bedingung für Service-Interoperabilität nicht genügen. Um den Übergang des nicht-interoperablen Workflows des Anwendungsszenarios in einen mediatorbasierten interoperablen Workflow durch Anwendung der QDAS-Prozedur zu ermöglichen, benötigt man

(a) eine initiale Menge von Service-Mediatoren und

(b) eine datenorientierte Domänenontologie mit der die Service-Mediatoren annotiert werden können.

Die initiale Ausprägung des Mediatorpools hinsichtlich einer gegebenen Klasse von Fragestellung beinhaltet die Service-Mediatoren, die für diese Klasse essentiell benötigt werden. Doch wie bestimmt man die Elemente dieser ersten Ausprägung? Eine automatische Herleitung der initialen Ausprägung ist sehr schwierig, wenn überhaupt lösbar. Hierfür gibt es verschiedene Gründe:

- Es gibt kein bekanntes Verfahren, dass eine gegebene Software in eine „optimale“ Partitionierung von Komponenten zerlegt. Dies liegt daran, dass nicht klar ist, wann eine Partitionierung optimal ist. Welche Granularität müssen die Komponenten haben, wie hoch sollte der Grad ihrer Autarkie (*self-containedness*) sein?
- Komplexe Funktionalitäten eines Service-Mediators entstehen durch Verkettung anderer Funktionalitäten an bestimmten *Datenknoten*[6]. Wieviele Unterfunktionalitäten sollte eine komplexe Funktionalität haben und was sind „optimale“ Datenknoten, die von vielen Funktionalitäten genutzt werden können?
- Welche Properties sollte ein Service-Mediator besitzen, damit er möglichst intuitiv genutzt und gut wiederverwendet werden kann?

Die Problematik der Herleitung der initialen Ausprägung ist eng verwandt mit der Problematik der Erstellung einer Domänenontologie, die zur Annotierung der Service-Mediatoren und Services genutzt werden kann. Auch hier gibt es keine allgemeinen Richtlinien, sondern nur *Best-Practice-Ansätze* (Uschold und Gruninger, 1996; Noy und McGuinness, 2001).

[6]Dies sind die Punkte, an denen sich Output des Datenproduzenten mit dem Input des Datenkonsumenten treffen (vgl. Definition 4.8)

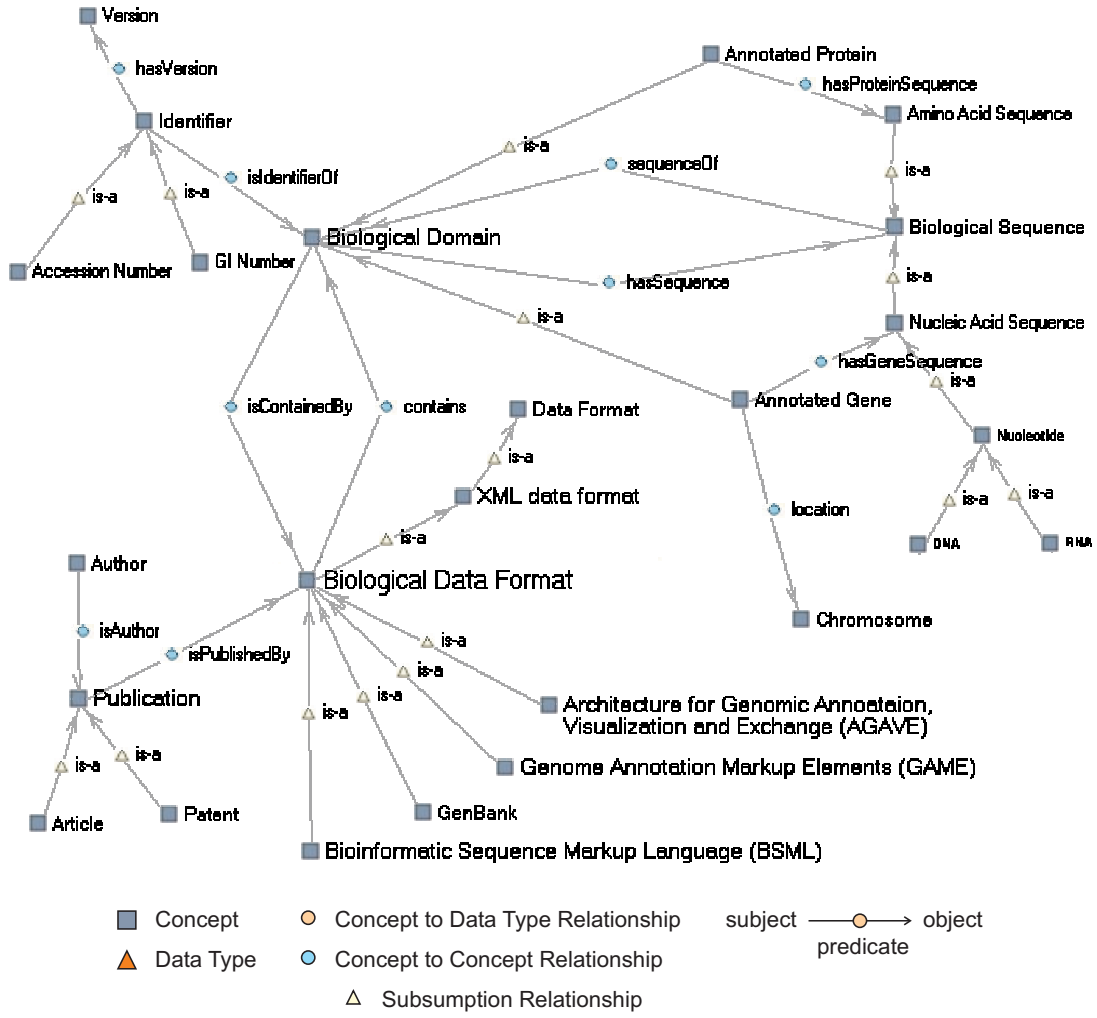

Abbildung 6.2: Eine prototypische Domänenontologie im Bereich der Bioinformatik.

6.2.1 Erstellung einer Domänenontologie

Es wird eine datenorientierte Domänenontologie benötigt, um die Ports und Properties der Mediatorfunktionalitäten der Service-Mediatoren zu annotieren, die anschließend beim Mediatorpool registriert werden sollen. Der Aspekt der „Datenorientierung“ einer Ontologie verdeutlicht, dass die Entitäten (Konzepte) – die in der Ontologie beschrieben und gegenseitig in Beziehung gesetzt werden – sich aus den verschiedenen Datenformaten und Datentypen des betrachteten Anwendungsgebietes ergeben. Beispielsweise können derartige Entitäten standardisierte Datenformate, bestimmte Metadaten und einzelne Elemente der Datenformate sein.

Gewöhnlich werden die Konzepte und Beziehungen durch eine manuelle Analyse der vorhandenen Services eines Anwendungsgebietes hergeleitet. Daher werden zukünftig Verfahren benötigt, die derartige Ontologien zumindest semiautomatisch erstellen können. Hier müssen Text Mining Ansätze und Clusterverfahren entsprechend entwickelt werden. Ein Clustering der Input- und Outputformate der verschiedenen Services könnte hier einen ersten Ansatz liefern. Hierzu könnte beispielsweise das von Dong et al. (2004) vorgestellte Verfahren eingesetzt werden. Vergleicht man zudem die Cluster der Inputformate mit den Clustern der Outputformate, lassen sich eventuell kleinere Strukturen der einen Clustermenge erkennen, die Teil größerer Strukturen der anderen Clustermenge sind. Dieser Vergleich kann zu *part-of* Beziehungen führen.

Ist eine derartige Domänenontologie auf Basis einer eingehenden Analyse entstanden, so sollte diese weitestgehend standardisiert werden, wie es beispielsweise bei der GeneOntology (GO)[7] erreicht wurde. Dieses kann die Zahl der zusätzlich benötigten *inter-ontology mappings* zwischen verschiedenen Domänenontologien reduzieren (Wache et al., 2001).

Im Gebiet der Bioinformatik könnte eine datenorientierte Domänenontologie Konzepte wie *BSML*, *AGAVE* aber auch Konzepte wie *Autor*, *Gen* oder *Sequenz* enthalten und diese in Beziehung setzen[8]. *is-a* und *part-of* Beziehungen ergeben sich beispielsweise folgendermaßen: BSML *ist ein* biologisches Datenformat, welches seinerseits ein XML Datenformat *ist*. AGAVE *beinhaltet* annotierte Gene, die eine korrespondierende Sequenz (hier DNA) *beinhalten*. Die DNA *ist eine* spezialisierte Nukleinsäuresequenz, welche ihrerseits eine Sequenz *ist*, genauso wie die Aminosäuresequenz. Aus dieser Beschreibung lassen sich erste Beziehungen für eine prototypische Domänenontologie identifizieren. Die Konzepte und ihre Beziehungen der in dieser Arbeit entwickelten und verwendeten Domänenontologie werden in Abbildung 6.2 visualisiert.

6.2.2 Erstellung einer initialen Ausprägung

Eine mögliche Herangehensweise zur Erstellung der initialen Ausprägung, welche teilweise durch die entwickelte Software unterstützt wird, ist an die Domänenontologie und deren Herleitung angelehnt. Ausgangspunkt der Analyse ist die Fragestellung, welche Datenformate in dem betrachteten Anwendungsgebiet häufig genutzt werden bzw. häufig auftreten?

Bei der Analyse sollten die Konzepte der Domänenontologie, denen die Datenformate der Services angehören, mit den Nutzungshäufigkeiten der einzelnen Services verknüpft werden. Konzepte von Services, die häufig verwendet werden, sollten einen höheren Einfluss haben als jene, die nur selten verwendet werden. *part-of* Beziehungen innerhalb der

[7] `www.geneontology.org/`

[8] Die im Rahmen dieser Arbeit entworfene prototypische Domänenontologie beinhaltet diese Konzepte und ähnelt damit in bestimmten Bereichen der im Projekt BioMOBY entwickelten Ontologie (siehe `biomoby.org/RESOURCES/MOBY-S/Objects`). In BioMOBY sind jedoch keine Konzepte vorhanden, die komplette Datenformate beschreiben und mit anderen Konzepten in Beziehung setzen.

Domänenontologie können zur Realisierung von Parsern herangezogen werden, die durch entsprechende Schema-Matchings der entsprechenden Datenformate teilweise auch generiert werden können.

Sind bereits mehrere Workflows vorhanden, können auch diese einer rechnerunterstützten Analyse unterzogen werden. Es kann untersucht werden, welche Services miteinander verknüpft werden und ob bestimmte Services häufig verbunden werden. Zudem können aus den Verbindungen Anfrageprofile generiert werden, die anschließend auf Basis ihrer Konzepte ebenfalls gruppiert werden. Auch dieses liefert Hinweise auf mögliche essentiell benötigten Service-Mediatoren.

Auch die verschiedenen Anwendungsklassen bilden eine Grundlage zur Identifikation relevanter Service-Mediatoren. Prinzipiell kann man auf Basis der Ontologie und auf Basis einer Anwendungsklasse überlegen, welche Funktionalitäten gebraucht werden:

- Essentielle Konvertierer zwischen relevanten Datenformaten (z.B. GenBank XML, AGAVE, BSML).
- Parser zwischen Subelementen und den verwendeten Datenformaten (z.B. ist Sequenz ein häufiger Input und kommt in den meisten biologischen Formaten vor).
- Komplexe Vergleichsoperationen, die häufig verwendet und benötigt werden (z.B. isSubsequenceOf, isProteinOf, intersect, contains).
- Spezielle oder allgemein einsetzbare Einheitenkonvertierer (z.B. für Längenmaße, Gewichtsmaße, Umrechnung von Nukleotidsequence in drei mögliche Aminosäuresequenzen).

Betrachten wir den geschilderten Anwendungsfall aus Abschnitt 6.1. Die Erzielung der Service-Interoperabilität des Workflows erfordert Service-Mediatoren, die die folgenden Funktionalitäten bereitstellen:

- Es wird eine Mediatorfunktionalität benötigt, die Namen oder Abkürzungen von Genen bzw. Proteinen auf die entsprechenden Identifikatoren abbildet.
- Werden unterschiedliche Identifikatoren eingesetzt, wird zudem eine Funktionalität gebraucht, die Identifikatoren unterschiedlicher Formate aufeinander abbilden kann.
- Es bedarf eines Parser, der auf die Sequenzinformation projizieren kann, die in der annotierten Information enthalten ist.
- Liegen die annotierten Informationen in unterschiedlichen Formaten vor, können Konvertierer benötigt werden.
- Die verschiedenen Clusterdarstellungen müssen durch eine Funktionalität auf das allgemeine Format des *Compare*-Service abgebildet werden.

Sind diese Service-Mediatoren vorhanden, kann durch ihren Einsatz ein mediator-interoperabler Workflow beschrieben werden. Im Folgenden wird das Vorgehen bei der Erstellung eines Service-Mediators beschrieben.

6.3 Vorgehen bei der Erstellung eines Service-Mediators

Sind die essentiell benötigten Service-Mediatoren identifiziert bzw. werden durch einen Workflow weitere benötigt, dann muss zum einen ein entsprechendes Profil erstellt und zum anderen der entsprechende Mediator realisiert werden. Hierzu existieren verschiedene Vorgehensweisen.

6.3.1 Erstellung eines Mediatorprofils

Die Erstellung des Mediatorprofils kann von Grund auf durch den Benutzer mit Hilfe eines entsprechenden Werkzeuges realisiert werden oder durch eine vorherige Anfrage auf Basis von zu überbrückenden Services (beispielsweise im Falle der Proxy-Funktionalitäten).

From Scratch

In der IRIS-Umgebung wurde ein Anwenderwerkzeug entwickelt, welches den Benutzer bei der Erstellung und Wartung eines Mediatorprofils unterstützt. Somit muss sich der Nutzer nicht mit der OWL-basierten Mediatorprofilsprache (MPL) direkt auskennen, sondern kann intuitiv ein neues Profil erstellen. Abbildung 6.3 illustriert dieses Anwendungswerkzeug. Über mehrere Fenster(-Schritte) hinweg wird der Benutzer zu den verschiedenen, einzugebenden Elementen des Profils geführt (in Abb. 6.3 durch die nummerierten Pfeile dargestellt). Nach Fertigstellung des Profils ermöglicht das Werkzeug dem Benutzer die Einbindung des Profils in den Mediatorpool.

Herleitung

Über die Herleitung eines Anfrageprofils auf Basis datenproduzierender und datenkonsumierender Operationen eines Workflows kann ein benötigtes Mediatorprofil auch generiert werden. Das Anfrageprofil kann anschließend editiert und als Grundlage eines neuen Service-Mediators genutzt werden. Ferner können auch während des Retrievalprozesses neue Mediatorprofile durch sog. Proxy-Funktionalitäten entstehen, wie es bereits in Kapitel 5 beschrieben wurde. Die grafische Benutzerschnittstelle zur Behandlung der Retrievalergebnisse inklusive eines *Preview*-Fensters ist in Abbildung 6.4 gezeigt. Aus dem

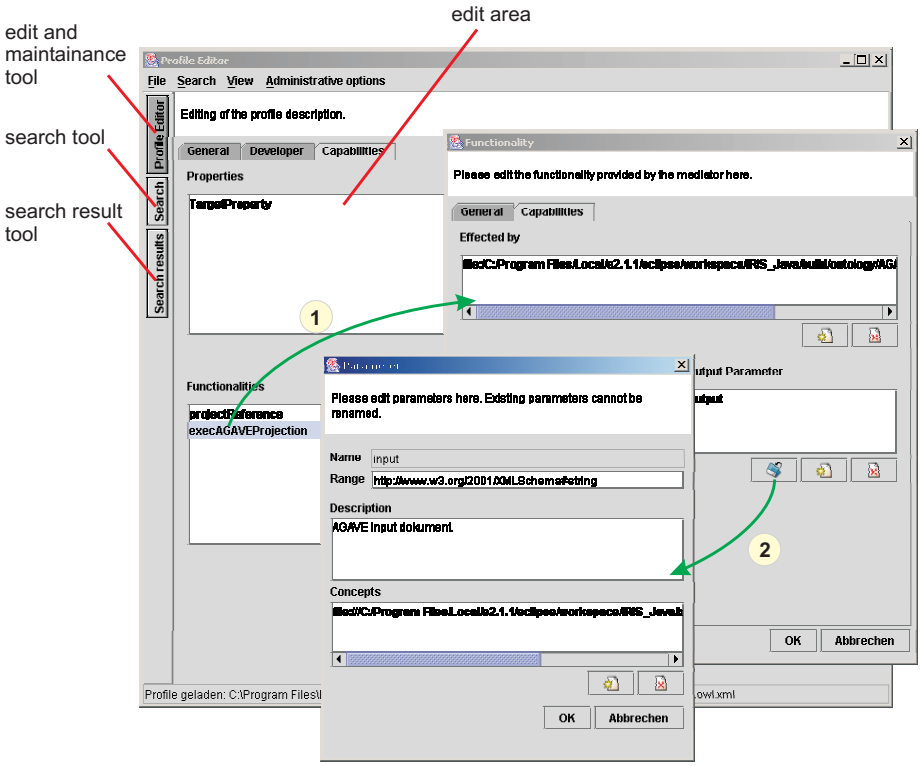

Abbildung 6.3: GUI zur Erstellung und Editierung von Mediatorprofilen.

Preview-Fenster heraus kann auch das generierte Profil editiert und abgespeichert werden. In diesem Fall gelangt man zurück zum Editier- und Wartungsbereich des Werkzeuges, wie er in Abbildung 6.3 dargestellt ist.

6.3.2 Realisierung eines konkreten Service-Mediators

Nachdem das Mediatorprofil eines neuen Service-Mediators vorhanden ist, muss dieser in einer vom System unterstützten Programmiersprache entwickelt oder durch Komposition existierender Service-Mediatoren erstellt werden.

Sprachgeneratoren

Derzeit sind im Rahmen der IRIS-Plattform Generatoren sowohl für Java-Skeletons als auch für die benötigten WSDL-Beschreibungen vorhanden. Das Listing in Abbildung 6.5 stellt einen Beispielauszug eines generierten WSDL-Dokument dar. Zu erkennen ist, dass neben den Funktionalitäten auch Properties durch Operationen im Port-Typ ausgedrückt werden.

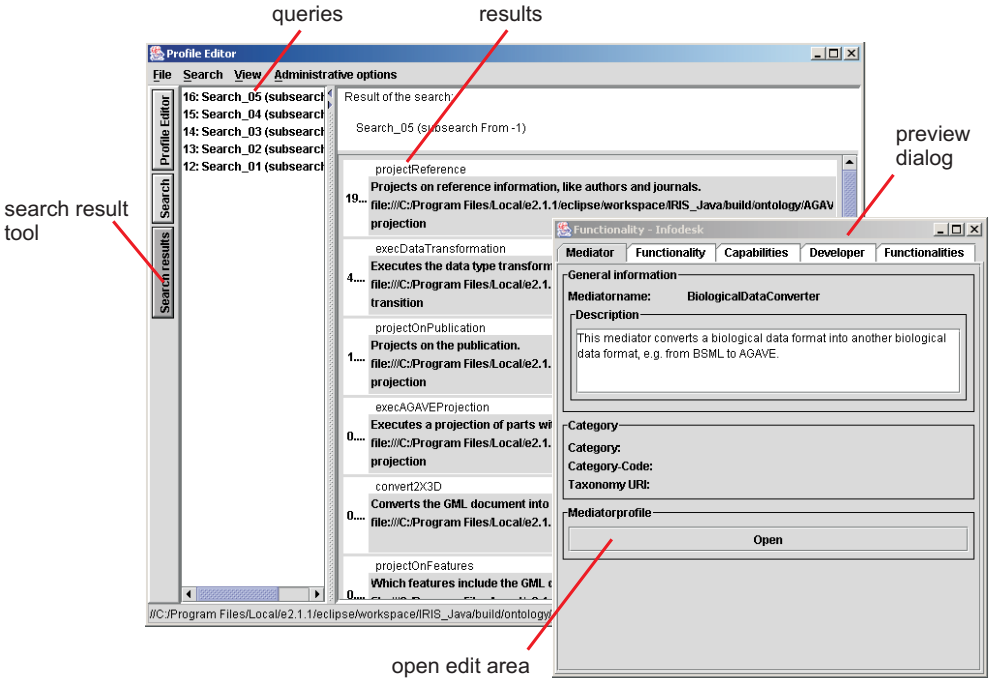

Abbildung 6.4: GUI der Anfrageergebnisse inklusive des *Preview*-Fensters.

Hier wird standardmäßig ein *Set* vor den entsprechenden Property-Namen eingefügt, d.h. die Konvention ist `Set<Property-Name>Property`. Diese Konvention wurde eingeführt, da WSDL standardmäßig keine Properties explizit unterstützt.

Neben dem WSDL-Dokument werden ferner eine Interface-Klasse und eine Skeleton-Klasse in Java generiert. Das Interface beinhaltet die Methoden, die von dem konkreten Service-Mediator zu realisieren sind. Die Skeleton-Klasse implementiert dieses Interface. In dieser Klasse müssen an den ausgezeichneten Stellen konkrete Realisierungselemente und Algorithmen eingefügt werden. Die entsprechenden Punkte sind, wie in Abbildung 6.6 zu erkennen, durch das Schlüsselwort `TODO` gekennzeichnet. Ein durch die IRIS-Software generiertes Java-Interface findet sich in Abbildung 6.7 wieder.

Neben den derzeitigen Generatoren für die Programmiersprache Java können beliebige Erweiterungen für andere Sprachen integriert werden. Hierzu muss lediglich ein Interface der IRIS-Plattform für den neu anzubindenden Generator implementiert werden.

Komposition von Service-Mediatoren

Eine andere Möglichkeit, Service-Mediatoren zu erstellen, ist die Komposition. Hierzu wird auf Basis bereits realisierter Mediatorfunktionalitäten eine neue komplexe Funktionalität

```
<portType name="GeneProjectionPortType">

 <!-- Properties -->
 <operation name="SetSequenceProperty">
  <input name="SetSequencePropertyRequest"
   message="tns:SetSequencePRequestM" />
  <documentation>Specification of the
    target sequence
  </documentation>
 </operation>
```

```
 <!-- Functionalities -->
 <operation name="getSequence">
  <input name="getSequenceRequest"
   message="tns:getSRequestM" />
  <output name="getSequenceResponse"
   message="tns:getSResponseM" />
  <documentation>Get the amino acid or
   nucleotide sequence from the
   annotated gene.</documentation>
 </operation>
</portType>
```

Abbildung 6.5: Generierter Port-Typ einer beispielhaften Proxy-Funktionalität.

```
public class GeneProjectionImpl implements
             GeneProjectionPortType {

    /*
     * -------------------------
     * Properties Implementation
     * -------------------------
     */
    /**
     * Specification of the target sequence
     */
    public void setSequenceProperty
        (java.lang.String targetSequence)
        throws java.rmi.RemoteException {
        // TODO Auto-generated method stub
        // for property SequenceProperty
    }
```

```
    /*
     * ----------------------------
     * Functionalities Implementation
     * ----------------------------
     */
    /**
     * Get the amino acid or nucleotide
     * sequence from the annotated gene.
     */
    public java.lang.String getSequence
        (java.lang.String annotetedGene)
        throws java.rmi.RemoteException {
        // TODO Auto-generated method stub
        //for functionality getSequence
        return null;
    }
}
```

Abbildung 6.6: Generiertes Java-Skeleton für eine beispielhafte Proxy-Funktionalität mit entsprechend gekennzeichneten, zu realisierenden Bereichen.

erzeugt, die einem Service-Mediator zugeordnet werden kann. Die Komposition kann, wie im vorherigen Kapitel beschrieben, teilweise automatisch geschehen, oder interaktiv durch den Nutzer. Im letzteren Fall wählt er einzelne Funktionalitäten direkt aus dem Mediatorpool aus oder sucht gezielt über die verschiedenen Retrievalmethoden nach geeigneten Mediatoren. Anschließend kann er diese verknüpfen.

Auch hierfür stellt die IRIS-Plattform ein rudimentäres Werkzeug zur Verfügung. Abbildung 6.8 zeigt die prototypische Realisierung des Werkzeuges. Dargestellt werden die Funktionalitäten inklusive ihrer Ports. Über die Ports lässt sich nun der Datenflussgraph beschreiben. Auf ähnliche Weise kann auch der Kontrollflussgraph modelliert werden. Komplexe Funktionalitäten werden, wie in Kapitel 5.5 beschrieben, durch die im IRIS-Framework vorhandene Workflow-Engine ausgeführt.

```
public interface GeneProjectionPortType
                 extends java.rmi.Remote {
    /*
     * ----------
     * Properties
     * ----------
     */
    /**
     * Specification of the target sequence
     */
    public void setSequenceProperty
        (java.lang.String targetSequence)
        throws java.rmi.RemoteException;

    /*
     * ---------------
     * Functionalities
     * ---------------
     */
    /**
     * Get the amino acid or nucleotide
     * sequence from the annotated gene.
     */
    public java.lang.String getSequence
        (java.lang.String annotetedGene)
        throws java.rmi.RemoteException;
}
```

Abbildung 6.7: Generiertes Java-Interface einer beispielhaften Proxy-Funktionalität.

6.4 Die software-unterstützte Prozedur in der Praxis

Im Folgenden betrachten wir die in Kapitel 5 vorgestellte QDAS-Prozedur in der Praxis. Ausgehend von dem in Abschnitt 6.1 beschriebenen Anwendungsfall, werden in diesem Abschnitt exemplarisch zwei konkrete Services zur Demonstration der Prozedur genutzt. Für die Sequenzdatenbank wird der XEMBL Service[9] und für die BLAST-Methode der DDBJ BLAST Service[10] herangezogen. Diese sind in Abbildung 6.1 hervorgehoben dargestellt. Tabelle 6.1 beschreibt die Schnittstellen des XEMBL Service und des DDBJ BLAST Service des Anwendungsfalles. Die kompletten WSDL Port-Typen finden sich in Anhang B.3.

Es wird folgende Notation für Operationen und Nachrichten eingesetzt: Eine *Operation* O eines Service wird durch den Vektor $O = (n_O, i_O, o_O, desc_O)$ spezifiziert, wobei

n_O ist der Operationsname,
i_O ist die *Inputnachricht*,
o_O ist die *Outputnachricht* und
$desc_O$ ist eine inhaltliche Beschreibung der Operation.

Nachrichten m werden allgemein dargestellt als Paar $m = (\mathcal{P}_m, desc_m)$, wobei

$\mathcal{P}_m$ ist eine endliche Menge von erweiterten Ports (siehe Definition 3.2),
$desc_m$ ist eine inhaltliche Beschreibung der Nachricht.

Namen und Beschreibungen werden durch Worte über einem beliebigen Alphabet gebildet.

Aus der Tabelle 6.1 wird ersichtlich, dass die beiden verknüpften Operationen nicht der Bedingung 3.3 auf Seite 40 genügen, und daher nicht-interoperabel sind. Ziel der QDAS-Prozedur ist daher die Identifizierung und Anpassung einer Mediatorfunktionalität f mit

[9] www.ebi.ac.uk/xembl/
[10] xml.nig.ac.jp/wsdl/

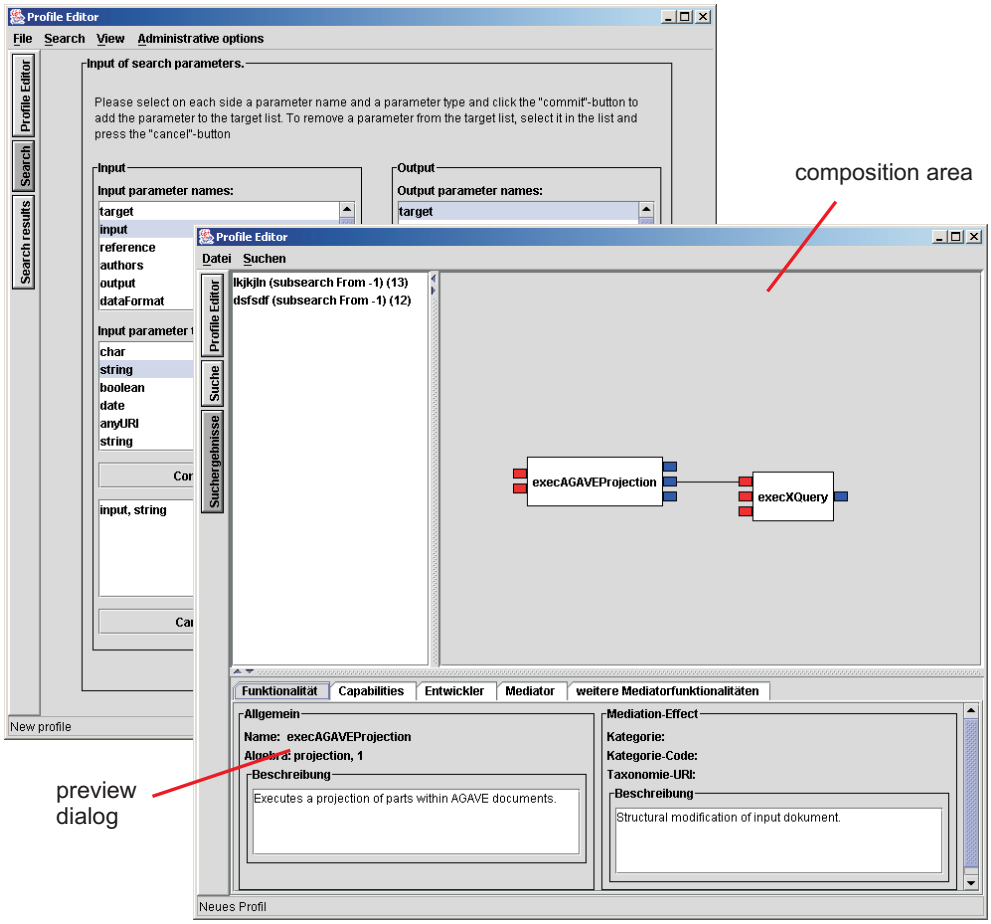

Abbildung 6.8: GUI zur Darstellung und zum Entwurf komplexer Funktionalitäten.

$O_1 \rightsquigarrow_f O_2$. Diese muss anschließend in den Workflow eingebracht werden, um einen mediator-interoperabelen Workflow zu erhalten.

Es wird weiterhin von einer initialen Ausprägung mit zwei Service-Mediatoren ausgegangen, die sich mittels des in Abschnitt 6.3 beschriebenen Verfahrens realisieren lassen. Ihre Mediatorfunktionalitäten sind in Tabelle 6.2 beschrieben.

- Der erste Service-Mediator besitzt eine Konvertierfunktionalität f_1, die biologische Datenformate aufeinander abbildet.

- Der zweite Service-Mediator beinhaltet eine Parserfunktionalität f_2, die aus den Daten im AGAVE Format auf die annotierte Gensequenz oder das entsprechende Chromosom projiziert.

Service	Schnittstelle
XEMBL	$O_1 = (\text{'getNucSeq'}, i_1, o_1, \perp)$ $i_1 = (\{(\text{'format'}, \text{'string'}, \emptyset, \text{'...Legit values: Bsml or sciobj...'}),$ $(\text{'ids'}, \text{'string'}, \emptyset, \text{'...Sequence accession numbers...'})\}, \perp)$ $o_1 = (\{(\text{'result'}, \text{'string'}, \emptyset, \text{'An XML...Bsml or AGAVE format.'})\}, \perp)$
DDBJ BLAST	$O_2 = (\text{'searchSimple'}, i_2, o_2, \text{'Execute BLAST specified...'})$ $i_2 = (\{(\text{'program'}, \text{'string'}, \emptyset, \text{'specify blastn, blastp,...'}),$ $(\text{'database'}, \text{'string'}, \emptyset, \text{'specify database, e.g. DDBJ,...'}),$ $(\text{'query'}, \text{'string'}, \emptyset, \text{'query sequence'})\}, \perp)$ $o_2 = (\{(\text{'result'}, \text{'string'}, \emptyset, \perp)\}, \perp)$

Tabelle 6.1: Schnittstelle des XEMBL Services und des DDBJ BLAST Services. Zu beachten ist, dass die inhaltlichen Beschreibungen des DDBJ BLAST Service in das WSDL-Dokument aus dem Dokument 'blast.txt' eingefügt wurden. Dieses Dokument ist derzeit unter `http://xml.ddbj.nig.ac.jp/wsdl/` separat verfügbar. Mit $\perp$ werden leere Beschreibungen bezeichnet.

Service-Mediator	Schnittstelle der Mediatorfunktionalität
Konvertierer	$f_1 : (i_{f_1}, \mathcal{P}^{\wp}_{f_1}) \rightarrow (o_{f_1})$ $i_{f_1} = (\{(\text{'convert'}, \text{'string'}, \{\text{'biological data format'}\}, \text{'...'})\}, \text{'...'})$ $o_{f_1} = (\{(\text{'bioTarget'}, \text{'string'}, \{\text{'biological data format'}\}, \text{'...'})\}, \text{'...'})$ $\mathcal{P}^{\wp}_{f_1} = \{(\text{'targetFormat'}, 0..3, \{\text{'agave','bsml','genbank','game'}\}, \text{'...'})\}$ $\mathcal{A}_{f_1} = (\text{'convert'}, \text{'...'}, \text{'biology'}, \text{'converter'})$
Parser	$f_2 : (i_{f_2}, \mathcal{P}^{\wp}_{f_2}) \rightarrow (o_{f_2})$ $i_{f_2} = (\{(\text{'agaveInput'}, \text{'string'}, \{\text{'agave'}\}, \text{'...'})\}, \text{'...'})$ $o_{f_2} = (\{(\text{'xmlProjection'}, \text{'string'}, \{\text{'annotated gene','chromosome'}\},$ $\text{'...'})\}, \text{'...'})$ $\mathcal{P}^{\wp}_{f_2} = \{(\text{'projectOn'}, \text{'string'}, \{\text{'annotated gene','chromosome'}\}, \text{'...'})\}$ $\mathcal{A}_{f_2} = (\text{'project'}, \text{'...'}, \text{'biology'}, \text{'parser'})$

Tabelle 6.2: Schnittstellenspezifikation der Mediatorfunktionalitäten zweier Service-Mediatoren. Zur besseren Übersicht sind die inhaltlichen Beschreibungen weggelassen.

Die in Abschnitt 6.2 beschriebene Domänenontologie wird zur *erweiterten Ontologie*, indem bei der Registrierung zwischen den durch eine Mediatorfunktionalität abgebildeten Konzepten eine neue Relation eingefügt wird, die auf die entsprechende Mediatorfunktionalität verweist (vgl. Kapitel 5.3.5). Diese, in Abbildung 6.9 durch *med.* gekennzeichneten Kanten, werden beim Konzept-Matchmaking ausgenutzt.

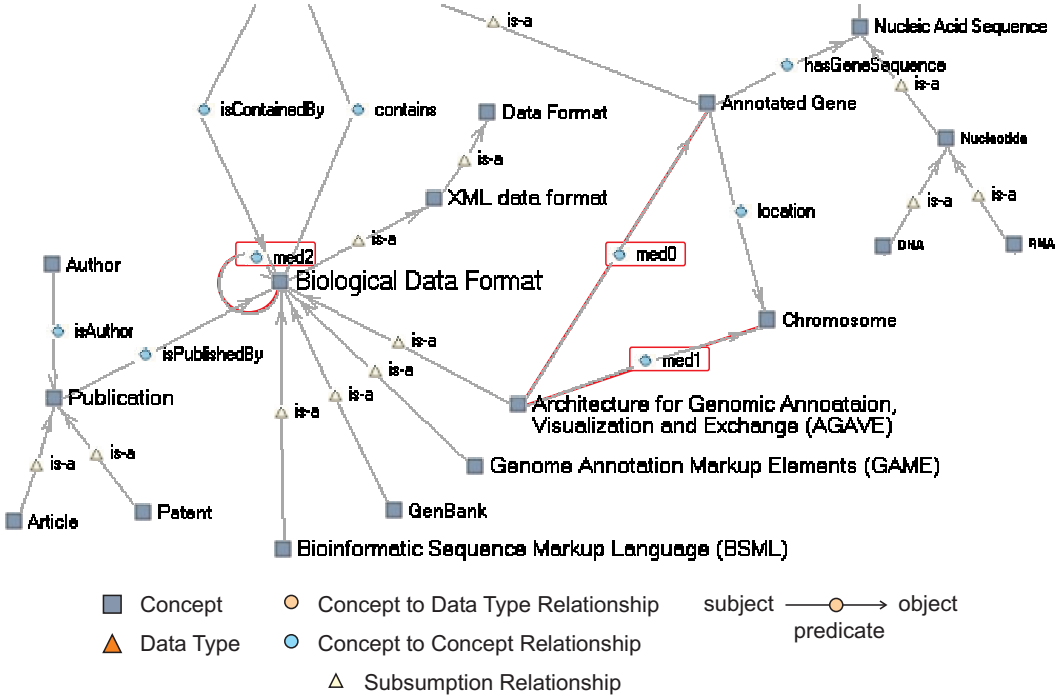

Abbildung 6.9: Erweiterte Domänenontologie zur Indexierung: hervorgehobene, mit *med.* markierte Kanten weisen auf in der Domänenontologie indexierte Service-Mediatoren hin.

Herleitung des Anfrageprofils

Die QDAS-Prozedur beginnt mit der Herleitung des Anfrageprofils basierend auf den Schnittstelleninformationen der verwendeten Services. Die Problematik dieser Herleitung lässt sich gut anhand der beschriebenen Services veranschaulichen. Beispielsweise beinhaltet der in Tabelle 6.1 beschriebene XEMBL Service die Operation '`getNucSeq`'. Diese besitzt einen Port mit dem Namen '`result`' des Typs '`string`', jedoch keine annotierten Konzepte. Dieser Port beinhaltet weiterhin eine inhaltliche Beschreibung. Der Name '`result`' besitzt keine (semantische) Bedeutung, die zur Verarbeitung durch den Computer herangezogen werden kann (*Was heißt Resultat und wovon das Resultat?*). Auch der Datentyp ist irreführend, da hier ein generischer Typ verwendet wird. Beschrieben wird ein '`string`', kodiert wird jedoch ein komplexes XML Dokument im AGAVE oder im BSML Format. Ferner ist der Operationsname aus drei Worten zusammengesetzt, wobei zwei davon Abkürzungen sind.

Wendet man nun das entwickelte Verfahren auf die Aggregation des XEMBL Service (Datenproduzent) mit dem DDBJ BLAST Service (Datenkonsument) an, ergibt dies das in Tabelle 6.3 vorgestellte und im Anhang B.4 beschriebe Anfrageprofil q. Das Profil spezifiziert

Anfrage	Spezifikation in MPL
Anfrageprofil	$q : (i_q, \mathcal{P}_q^{\wp}) \rightarrow (o_q)$
	$i_q = (\{(\text{'result'}, \text{'string'}, \{\text{'agave'}, \text{'bsml'}, \text{'data format'}, \text{'XML'}\},$
	'An XML formatted...Bsml or AGAVE format.')$\}, \perp)$
	$o_q = (\{(\text{'program'}, \text{'string'}, \emptyset, \text{'specify blastn, blastp,...'}),$
	$= (\{(\text{'database'}, \text{'string'}, \emptyset, \text{'specify database, e.g. DDBJ,...'}),$
	$= (\{(\text{'query'}, \text{'string'}, \{\text{'sequence'}\}, \text{'query sequence'})\}, \perp)$
	$\mathcal{A}_q = (\perp, \text{'...'}, \perp, \perp)$

Tabelle 6.3: Generiertes Anfrageprofil aus XEMBL nach DDBJ BLAST. Mit $\perp$ werden leere Beschreibungen bezeichnet.

die Anforderungen an konkrete Service-Mediatoren, die zwischen diesen beiden Services vermitteln sollen ($O_1 \rightsquigarrow_q O_2$). Neben den syntaktischen Informationen werden durch den Algorithmus auch semantische Konzepte der Domänenontologie aus Abschnitt 6.2 annotiert, wie '`agave`' oder '`sequence`'. Das Konzept '`data format`' wurde zusätzlich identifiziert, da '`data format`' ein Synonym für '`format`' ist und dieser Term in der Beschreibung des XEMBL-Services genutzt wird.

Konzept-Matchmaking

Auf Basis des generierten Anfrageprofils q findet nun der Retrievalprozess statt. In dem konkreten Beispiel wird lediglich das Konzept-Matchmaking betrachtet, d.h. es wird das Verfahren aus Kapitel 5.3.5 auf das Anfrageprofil q und die beiden registrierten Service-Mediatoren aus Tabelle 6.2 angewendet.

Der beschriebene Algorithmus identifiziert den Konvertierer f_1, da der XEMBL Service *BSML* oder *AGAVE* Daten produzieren kann, die beide dem Konzept '`biological data format`' angehören. Über die *is-a* Beziehung des Konzepts '`biological data format`' zum Konzept '`AGAVE`' wird ferner der Parser f_2 erkannt. Da diese *is-a* Beziehung jedoch „absteigend“ ist, ist die korrekte Anwendung von f_2 an dieser Stelle nicht sicher gestellt. Daher wird für diesen Fall der Wert h mit $h = 1$ belegt (vgl. Gleichung 5.6). Abschließend führt der Algorithmus eine Proxy-Funktionalität f_3 ein, die sich auf Basis der Beziehung zwischen den Konzepten '`sequence`' und '`annotated gene`' ergibt. Das resultierende Operationsnetzwerk für die Anfrage q wird in Abbildung 6.10 visualisiert.

Anpassung der komplexen Funktionalität

Der obige Discovery-Prozess liefert eine mögliche Komposition von Service-Mediatoren, die in dieser Phase der Prozedur abschließend angepasst werden muss. Hinsichtlich des Kontrollflusses umfasst dies die Einfügung einer Fallunterscheidung in den Kontrollflussgraphen

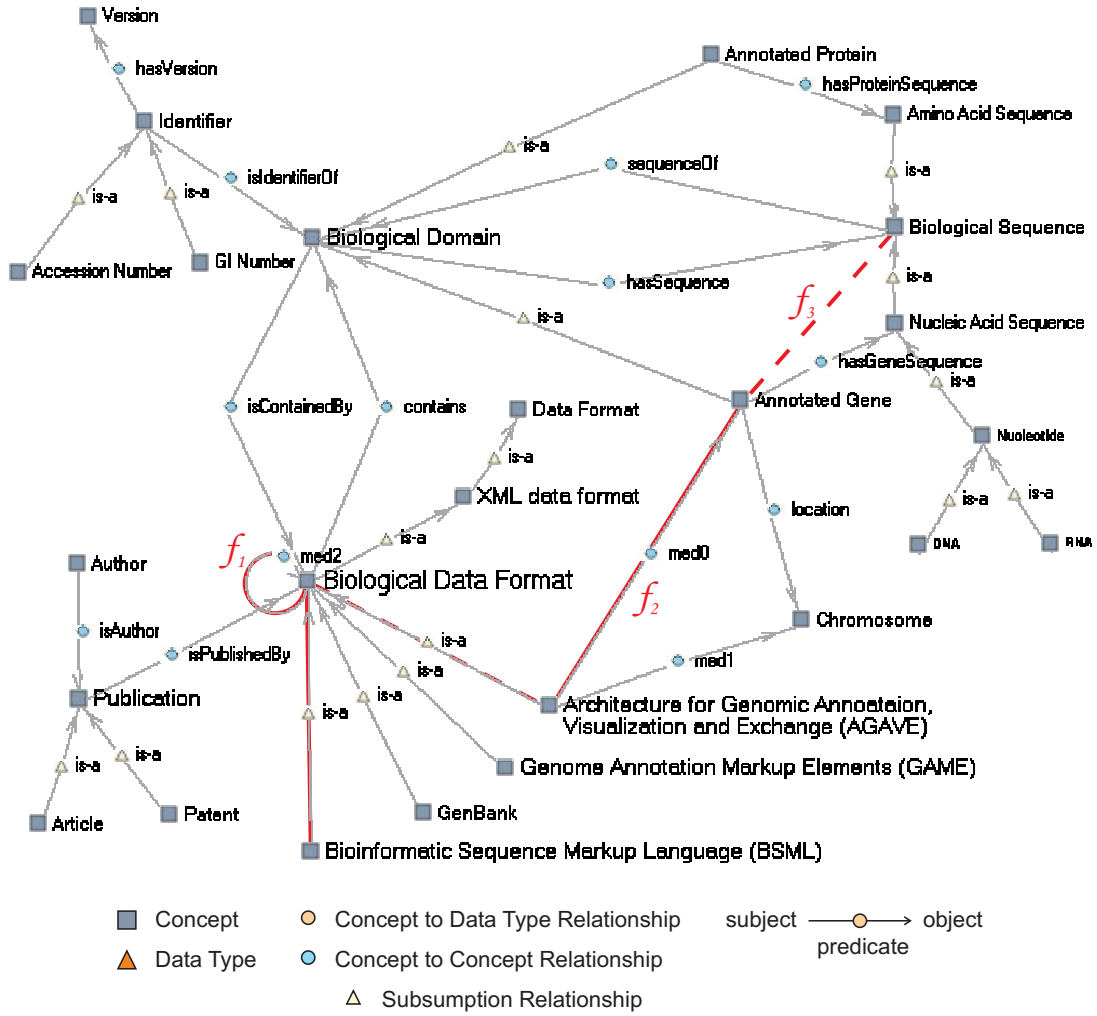

Abbildung 6.10: Ontologiebasiertes Retrieval der Service-Mediatoren: gestrichelte Kanten weisen auf problematische Übergänge oder Proxy-Funktionalitäten hin.

mit dem entsprechenden Komparator *Comp*, der überprüft, ob die eingehenden Daten vom Typ BSML oder AGAVE sind. Es müssen ferner folgende Aktivierungsübergänge dem beim Discovery generierten Kontrollflussgraphen hinzugefügt werden:

- (*Comp*, *test*, *convert*): Evaluiert der Komparator die neu eingefügte Variable *test* zu `true`, wird die Funktionalität f_1 (*convert*) ausgeführt.
- (*Comp*, $\neg$*test*, $\hat{e}$): Sind die eingehenden Daten bereits vom Typ AGAVE, evaluiert *Comp* die Variable *test* zu `false`. Daher wird nachfolgend keine Konvertierung durchgeführt. Stattdessen wird die Aktivierung an die leere Aktivität $\hat{e}$ weitergegeben.

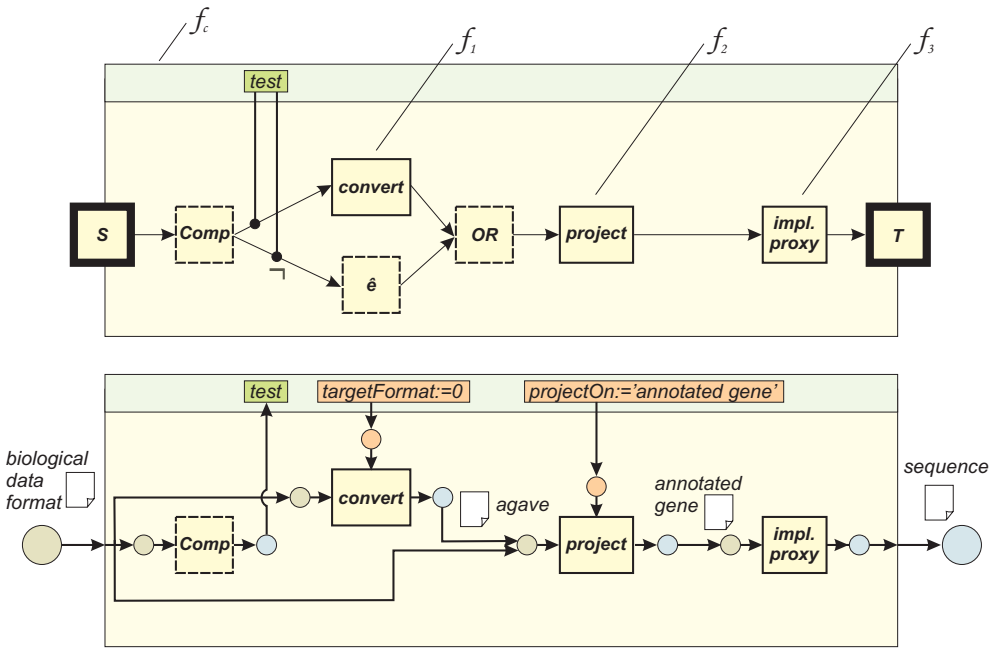

Abbildung 6.11: Beispiel der komplexen, angepassten Funktionalität f_c: Kontrollflussgraph (oben), Datenflussgraph und Eigenschaftsgraph (unten).

- $(convert, \top, \text{OR})$ und $(\hat{e}, \top, \text{OR})$: Abschließend werden die bedingten Aktivierungsübergänge durch die OR-Synchronisationsaktivität zusammengeführt.

Existiert ein derartiger Komparator, können die Discovery-Mechanismen ausgenutzt werden, um diesen zu identifizieren. Abbildung 6.11 (oben) illustriert den resultierenden Kontrollflussgraphen.

Neben der Modifikation des Kontrollflussgraphen muss ferner die Proxy-Funktionalität f_3 realisiert werden (*impl. proxy*). Hierzu können die in Abschnitt 6.3 vorgestellten Verfahren eingesetzt werden. Beispielsweise lässt sich Mediatorprofil und WSDL-Dokument für f_3 automatisch generieren, da diese durch den Discovery-Prozess bereits bekannt sind.

Abbildung 6.11 (unten) skizziert sowohl den Datenfluss- als auch den Eigenschaftsgraphen. Über letzteren wird abschließend die Konfigurierung der komplexen Funktionalität vorgenommen. Hier müssen die Eigenschaftsfelder *targetFormat* und *projectOn* der Funktionalitäten f_1 und f_2 entsprechend den Anforderungen auf `0` bzw. '`annotated gene`' gesetzt werden.

Die durch die Phase der Adaption neu entstandenen atomaren und komplexen Funktionalitäten werden dem System durch die Phase der *Bekanntmachung* zur Verfügung gestellt. Diese Phase ist ebenfalls Bestandteil der QDAS-Prozedur. Während der Bekanntmachung werden die neuen Mediatorprofile im Mediatorpool registriert. Durch die Bekanntmachung

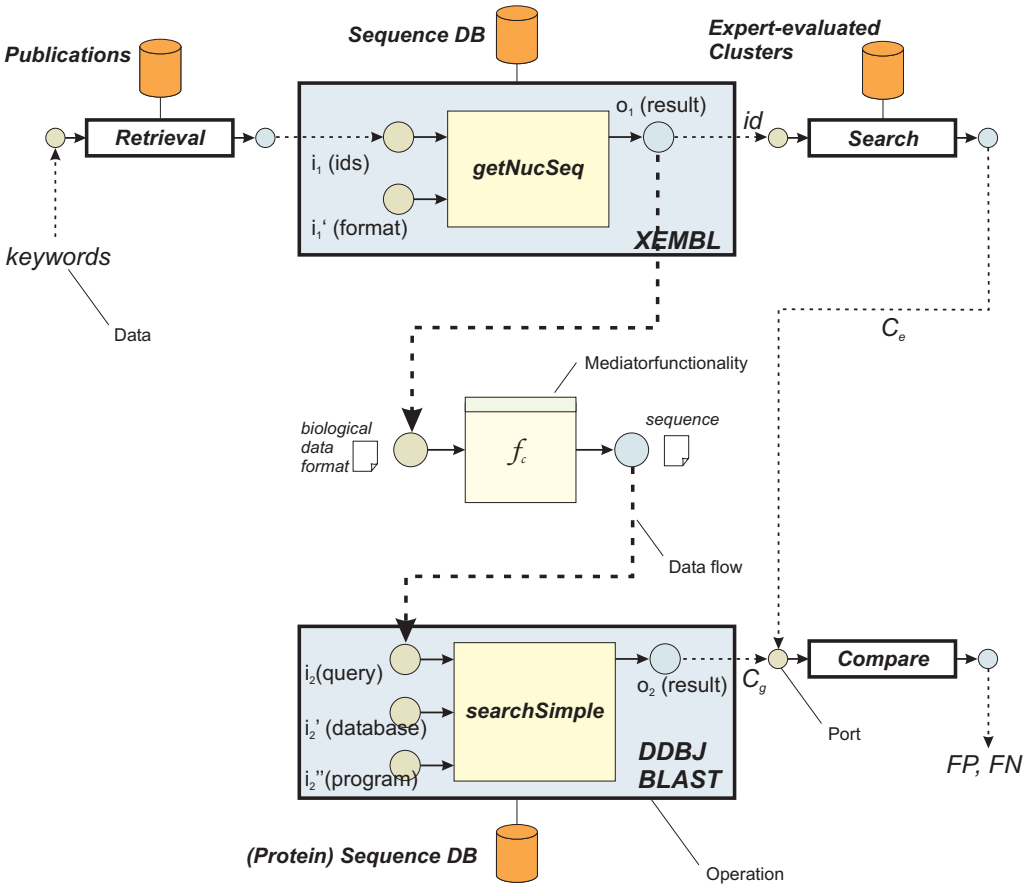

Abbildung 6.12: Erweiterung des *in-silico* Experiments zu einem mediator-interoperablen Workflow.

stehen diese Service-Mediatoren zukünftigen Workflows direkt zur Verfügung. In dem hier vorgestellten Beispiel wären dies der Komparator *Comp* (falls noch nicht vorhanden), die realisierte Proxy-Funktionalität f_3, sowie die komplexe Funktionalität f_c.

Das Resultat der QDAS-Prozedur ist in diesem Beispiel die neu entstandene Funktionalität f_c, die nun in den Workflow aus Abschnitt 6.1 eingebettet wird ($O_1 \rightsquigarrow_{f_c} O_2$). Die Einbettung ist in Abbildung 6.12 verdeutlicht. Dieses Prinzip wird nun sukzessiv auf die anderen Datenverbindungen des Workflows angewendet, bis alle komponierten Services mediator-interoperabel sind und der gesamte Workflow mediator-interoperabel ist. Hierzu ist lediglich eine entsprechend angepasste initiale Ausprägung nötig, wie sie in Abschnitt 6.2 beschrieben wurde.

6.5 Effektivität der software-unterstützten Prozedur: Sukzessive Modifikation eines Workflows

Ziel der QDAS-Prozedur ist es, ausgehend von einer geeigneten Ausprägung des Mediatorpools, diejenigen Service-Mediatoren zu identifizieren, die einen gegebenen Workflow zu einem mediator-interoperabelen Workflow erweitern können. Dieses Prinzip wurde im vorigen Abschnitt exemplarisch betrachtet. Bei jeder Anwendung der Prozedur können neue atomare bzw. komplexe Mediatorfunktionalitäten entstehen, falls diese nicht im Mediatorpool vorhanden sind. Die Anzahl der Anpassungen bzw. Neuentwicklungen sollte jedoch bei *ähnlichen* Workflows zurückgehen. Workflows sind sich ähnlich, wenn sie identische Teildatenflüsse über Services beinhalten, wobei die Service-Schnittstellen ähnliche Konzepte bereitstellen bzw. verarbeiten. Die Effektivität und Robustheit der QDAS-Prozedur hängt nun stark davon ab, ob es gelingt, die neu erstellten bzw. gezielt angepassten Mediatorfunktionalitäten mit hoher *Precision* und hohem *Recall* für *ähnliche* oder *modifizierte* Workflows wiederzuentdecken.

Um das Verhalten der QDAS-Prozedur diesbezüglich analysieren zu können, wird das *in silico* Experiment aus Abschnitt 6.1 sukzessiv modifiziert. Genauer gesagt, es wird die genutzte BLAST-Operation durch einen anderen Service-Provider ersetzt, der ebenfalls eine BLAST-Operation zur Verfügung stellt, jedoch über einer anderen Sequenzdatenbank. Die praktische Vorstellung dieser Modifikation ist, dass entweder der genutzte DDBJ BLAST Service nicht mehr erreichbar ist, oder dass dieser Dienst durch einen anderen BLAST-Service über einer anderen Datenbank ersetzt werden soll. Beispielsweise könnten Nutzer, die gewöhnlich über einer Arabidopsis-Datenbank arbeiten, sich entschließen, eine Reis-Datenbank für weitere Experimente zu verwenden. Der untere Pfeil der Abbildung 6.13 illustriert diesen ersten Versuch, der in Abschnitt 6.5.1 behandelt wird.

In einem zweiten Versuch wird zudem sukzessiv der Datenanbieter durch einen neuen Service ersetzt, d.h. es wird nun auch der XEMBL Service ausgetauscht. Dieser Aspekt wird in Abschnitt 6.5.2 beleuchtet und ist in Abbildung 6.13 durch den obigen Pfeil dargestellt. Hinter diesem Austausch verbirgt sich beispielsweise die praktische Vorstellung, dass der XEMBL Service nicht mehr gewartet wird, und daher durch einen neuen Dienst ersetzt wird. Der Aufbau der entsprechenden Testmengen findet sich in Anhang B.5.

Die durch die beschriebenen Modifikationen neu entstehenden Workflows werden als *ähnlich* angenommen, da gewöhnlich die ausgetauschten Operationen ähnliche Schnittstellen aufweisen. Somit müssten sich die bereits vorhandenen Mediatorfunktionalitäten größtenteils wiederverwenden lassen. Die *Ausführungssemantik* der verschiedenen Workflows kann jedoch aufgrund der Nutzung unterschiedlicher Datenquellen verschieden sein.

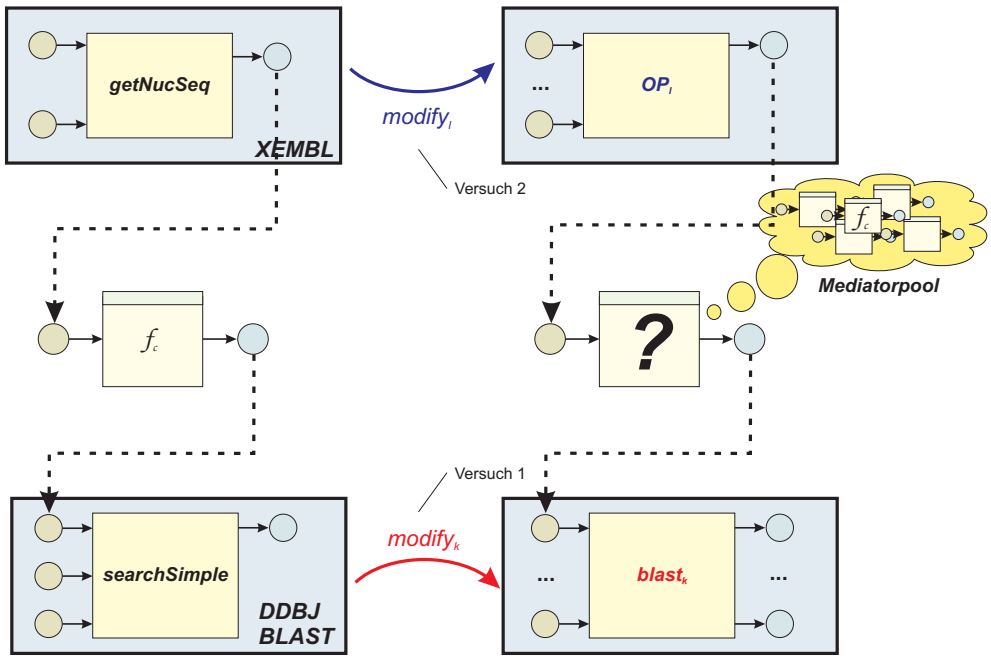

Abbildung 6.13: Modifikation des *in-silico* Experiments.

6.5.1 Versuch 1: *XEMBL nach BLAST*$_k$

In diesem Versuch wird sukzessiv die vorhandene Operation des XEMBL Services durch eine BLAST-Operation eines anderen Anbieters (BLAST$_k$, $1 \leqslant k \leqslant 34$) ersetzt. Untersucht werden die ersten beiden Phasen der QDAS-Prozedur. Außerdem wird das Retrievalverhalten der verschiedenen, entwickelten Matchmaking-Verfahren aus Kapitel 5.3 verglichen. Dabei werden mögliche Kompositionen jedoch nicht berücksichtigt, da diese nicht von allen Verfahren geliefert werden können.

Zu untersuchende Fragestellungen

1. Die verschiedenen Retrievalmechanismen sollten auf Basis der hergeleiteten Anfrageprofile möglichst die gleichen Service-Mediatoren identifizieren, die auch bei der Verknüpfung des XEMBL Services mit dem DDBJ BLAST Service gefunden wurden (vgl. Abschnitt 6.4).

2. Bestimmte BLAST Services sind nur für Nukleotidsequenzen gültig, andere nur für Proteinsequenzen und einige für beide Sequenztypen. Das Konzept-Matchmaking sollte auch hier möglichst die Unterscheidung treffen können.

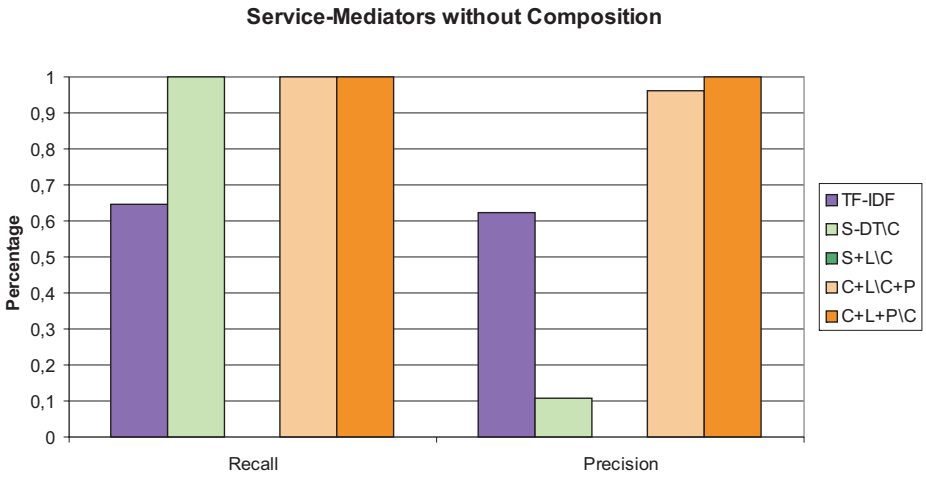

Abbildung 6.14: Vergleich der verschiedenen Matchmaking-Verfahren (Precision und Recall ohne Komposition).

Erzielte Ergebnisse

Angewendet wurden

- `TF-IDF`: das Keyword-Matchmaking, welches sich auf das Vector Space Model in Kombination mit der TF-IDF stützt,
- `S-DT\C`: das Signatur-Matchmaking, das nur auf den Datentypen arbeitet und die Attributnamen ignoriert,
- `S+L\C`: das Signatur-Matchmaking, das sowohl die Attributnamen als auch die Datentypen berücksichtigt und zusätzlich linguistische Methoden einsetzt ('`+L`'),
- `C+L\C+P`: das Konzept-Matchmaking, das auch nicht miteinander verbundene Ports betrachtet, d.h. alle Konzepte der Input- und Outputports zusammenfasst, und
- `C+L+P\C`: das Konzept-Matchmaking, das nur direkt miteinander verbundene Ports betrachtet.

Wie bereits erwähnt, werden mögliche Kompositionen von Service-Mediatoren ignoriert, da nicht alle Verfahren diese bereitstellen ('`\C`').

Abbildung 6.14 liefert die Precision- und Recall-Werte der verschiedenen Verfahren. Die `TF-IDF` liefert wie zu erwarten eine mittlere Precision. Interessanter ist, dass der Recall-Wert ebenfalls im mittleren Bereich auftritt und nicht volle 100% erreicht. Auch die

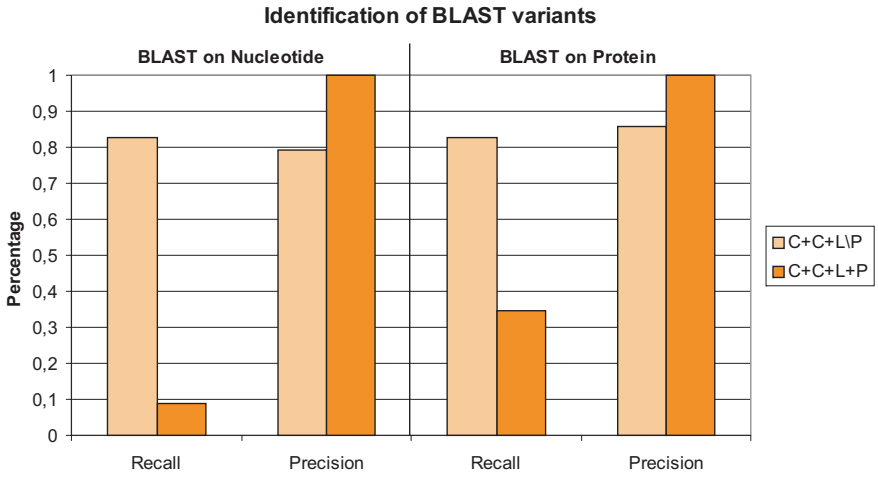

Abbildung 6.15: Recall und Precision des Konzept-Matchmaking unter Berücksichtigung der Komposition hinsichtlich Protein- und Nukleotidsequenzen.

schwächere Form der Signatursuche `S-DT\C` liefert im wesentlichen erwartete Ergebnisse, da sie im Großen und Ganzen alle Mediator identifiziert. Dadurch erhält sie einen hohen Recall bei sehr schlechter Precision. Die Versuche zeigen aber auch, dass die strengere Form der Signatursuche `S+L\C`, die Attributnamen und zudem linguistische Verfahren einsetzt, trotzdem zu restriktiv bleibt, um Service-Mediatoren zu identifizieren. Dieses Matchmaking-Verfahren ist somit nur für standardisierte Services und Service-Mediatoren oder für Organisationen mit klar definierten Schnittstellen geeignet. Die besten Resultate liefern die Konzept-Matchmaking-Verfahren `C+L\C+P` und `C+L+P\C`. Sowohl ihr Recall als auch ihre Precision liegen bei (fast) 100%.

Da die Konzept-Matchmaking-Algorithmen die einzigen sind, die zwischen Konzepten wie Nukleotid und Protein unterscheiden können, wurde die zweite Fragestellung lediglich mit diesen beiden Matchmaking-Varianten getestet. Abbildung 6.15 beschreibt die erzielten Resultate.

Angewendet wurden hier

- `C+C+L\P`: das Konzept-Matchmaking, das auch nicht miteinander verbundene Ports betrachtet, d.h. alle Konzepte der Input- und Outputports zusammenfasst, und
- `C+C+L+P`: das Konzept-Matchmaking, das nur direkt miteinander verbundene Ports betrachtet.

Beide Verfahren wurden angewendet sowohl unter Berücksichtigung ihrer Kompositionsfähigkeit (’`+C`’) als auch unter Verwendung linguistischer Methoden (’`+L`’).

Während die restriktivere Variante C+C+L+P, die nur die konkreten Port-Informationen berücksichtigt, eine sehr gute Precision erzielt, bleibt ihr Recall jedoch unterdurchschnittlich. Im Gegensatz dazu erreicht die schwächere Form C+C+L\P einen überdurchschnittlichen Recall bei gleichzeitiger überdurchschnittlicher Precision. Beide Werte liegen bei ca. 80%. Daher wäre nach diesen Resultaten das Verfahren C+C+L\P dem Verfahren C+C+L+P vorzuziehen, es sei denn, man benötigt eine annähernd 100%-ige Precision und kann die schlechteren Recall-Werte ignorieren.

6.5.2 Versuch 2: OP_l nach $BLAST_k$

Der erste Versuch wird dahingehend erweitert, dass auch die datenproduzierende Operation des XEMBL Services durch eine neue datenproduzierende Operation ersetzt wird (OP_l, $1 \leqslant l \leqslant 7$). Dieses wird mit dem Austausch der BLAST-Operationen kombiniert. Ferner wird für diesen Versuch davon ausgegangen, dass die in Abschnitt 6.4 beschriebene Anpassung durchgeführt wurde und sowohl die Proxy-Funktionalität als auch die neu entstandene, komplexe Funktionalität dem Mediatorpool bekannt gemacht wurden. Diese stehen somit der QDAS-Prozedur für die modifizierten Workflows zur Verfügung. Auch hier werden die verschiedenen Precision- und Recall-Werte gemessen, die über die 238 Testläufe hinweg gemittelt wurden. Da nur die Konzept-Matchmaking- und die Signatur-Matchmaking-Verfahren die Komposition unterstützen, jedoch das Signatur-Matchmaking nur für bestimmte Anwendungsgebiete mit wohldefinierten Schnittstellen geeignet ist, werden in diesem Versuch lediglich verschiedene Konzept-Matchmaking-Algorithmen angewendet.

Zu untersuchende Fragestellungen

Eine wesentliche Anforderung an die QDAS-Prozedur ist, dass einmal entwickelte und angepasste Service-Mediatoren gut wiederverwendet werden können. Daher sollten die angepassten, teilweise komplexen Service-Mediatoren durch den Such-Prozess identifiziert werden. Das heißt das Zusammenspiel von Anfragegenerierung und Discovery muss möglichst optimale Ergebnisse liefern (hohe Precision und hoher Recall).

Erzielte Ergebnisse

Angewendet wurden

- TF-IDF: das Keyword-Matchmaking, das das Vector Space Model in Kombination mit der TF-IDF nutzt,
- C+C\P+L: das Konzept-Matchmaking, das auch nicht miteinander verbundene Ports betrachtet, d.h. alle Konzepte der Input- und Outputports zusammenfasst und keine linguistischen Methoden einsetzt,

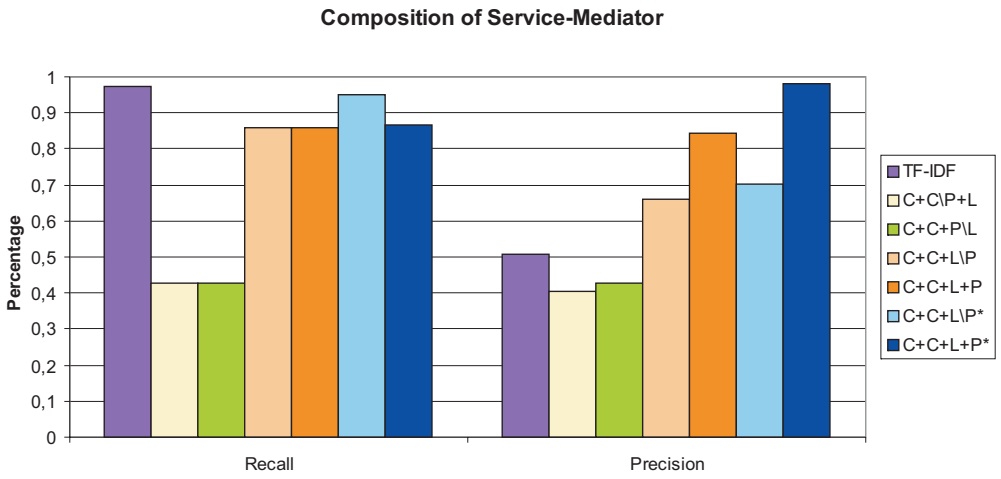

Abbildung 6.16: Identifizierung des komplexen Service-Mediators. Messung der Precision und des Recalls.

- `C+C+P\L`: die Konzept-Matchmaking-Einheit, die nur direkt miteinander verbundene Ports betrachtet und keine linguistischen Methoden einsetzt,
- `C+C+L\P`: das Konzept-Matchmaking, das auch nicht miteinander verbundene Ports betrachtet, d.h. alle Konzepte der Input- und Outputports zusammenfasst und zudem linguistischen Methoden einsetzt,
- `C+C+L+P`: das Konzept-Matchmaking-Einheit, das nur direkt miteinander verbundene Ports betrachtet und zudem linguistischen Methoden einsetzt,
- `C+C+L\P*`: eine Erweiterung des Konzept-Matchmaking `C+C+L\P` (s.u.) und
- `C+C+L+P*`: eine Erweiterung des Konzept-Matchmaking `C+C+L+P` (s.u.).

Alle Konzept-Matchmaking-Algorithmen wurden angewendet unter Berücksichtigung ihrer Kompositionsfähigkeit (’`+C`’). Das `TF-IDF` Verfahren liefert die Eichmarke, anhand derer die verschiedenen Verfahren bewertet werden. Abbildung 6.16 fasst das Resultat des Versuches zusammen.

Auch in diesem Versuch liefert das `TF-IDF` Verfahren im wesentlichen die erwarteten Ergebnisse: ein Recall von ca. 97% bei einer mittleren Precision von ca. 50%. Vergleicht man die Matchmaking-Varianten, die keine linguistischen Methoden einsetzen (`C+C\P+L` und `C+C+P\L`) mit denen, die diese einsetzen, zeigt sich sowohl bei der Precision als auch beim Recall ein signifikanter Unterschied. Ohne linguisitsche Ansätze bleiben die Verfahren unterdurchschnittlich und liegen sogar hinter dem `TF-IDF` Verfahren.

Die bereits betrachteten Verfahren C+C+L\P und C+C+L+P erreichen eine gute Precision (66% bzw. 84%), die signifikant besser ist, als die des TF-IDF Verfahrens. Leider erreichen sie aber nicht die gewünschten Recall-Werte. Hier liegen beide Verfahren bei ca. 86%, d.h. ca. 10% unter dem Wert der TF-IDF. Letzteres liegt vor allem daran, dass die Anfragegenerierung in bestimmten Konstellationen bei bestimmten datenproduzierenden bzw. datenkonsumierenden Diensten nur sehr ungenügende Anfrageprofile generiert.

> **Beispiel 6.1.** Sei S_1 und S_2 ein Datenproduzent bzw. ein Datenkonsument. Biete ferner S_2 eine inhaltliche Dokumentation im WSDL-Dokument an, während S_1 keine Dokumentation im WSDL-Dokument bereitstellt. Dann sucht das TF-IDF Verfahren lediglich mit Hilfe der Informationen aus S_2 den Mediatorpool ab. Die Konzept-Matchmaking-Verfahren arbeiten gleichzeitig mit den Input- und Outputkonzepten des Anfrageprofils, die sich aus den WSDL-Dokumenten herleiten lassen. Unter obiger Annahme werden jedoch durch die Anfragegenerierung keine Inputkonzepte aus S_1 extrahiert. Somit können die Matchmaking-Verfahren keine Service-Mediatoren entdecken.

Erweiterung des Konzept-Matchmaking

Aufgrund der oben geschilderten Problematik wird eine einfache Erweiterung in die Verfahren C+C+L\P und C+C+L+P eingeführt. Die Varianten C+C+L\P* und C+C+L+P* verlaufen für Anfrageprofile, deren Input- und Outputports Konzepte enthalten, identisch zu den Verfahren C+C+L\P bzw. C+C+L+P. Wurden jedoch durch die Anfragegenerierung entweder keine Konzepte für die Inputports oder keine Konzepte für die Outputports ins Anfrageprofil geschrieben, arbeiten die Erweiterungen lediglich mit den Konzepten der Outputports respektive der Inputports. In diesen Fällen werden alle Mediatorfunktionalitäten durch das Matchmaking selektiert, die entsprechende Konzepte der Outputports bzw. der Inputports inklusive der Subsumptionsrelation *matchen*.

Diese Erweiterungen liefern sehr gute Ergebnisse, die auch gegenüber dem TF-IDF Verfahren Vorteile aufweisen. Während die restriktivere Einheit C+C+L+P* eine sehr gute Precision von ca. 98% erreicht, liegt ihr Recall-Wert nur unwesentlich über dem ursprünglichen Verfahren C+C+L+P. Die schwächere Form C+C+L\P* hingegen erreicht ein nicht signifikant unterschiedliches Recall verglichen mit dem TF-IDF (95% vs. 97%), bei gleichzeitiger Erhöhung der Precision auf knapp über 70% (im Gegensatz dazu die Precision der TF-IDF von ca. 50%).

6.5.3 Schlussfolgerung

Die Effektivität und Robustheit der QDAS-Prozedur wird dadurch bestimmt, ob einmal entwickelte oder gezielt angepasste Funktionalitäten sich in verschiedenen Workflows einsetzen lassen. Hierzu ist es notwendig die geeigneten Funktionalitäten aus der Menge aller

Funktionalitäten automatisch auswählen zu können. Dieses Verhalten wurde anhand modifizierter Workflows des beschriebenen Anwendungsfalls untersucht.

Die Ergebnisse der Versuche zeigen unterschiedliche Ergebnisse der verschiedenen Matchmaking-Varianten. Wie zu erwarten, liefert das Keyword-Matching Resultate im mittleren Bereich. Vor allem das Konzept-Matchmaking erzielt sehr gute Ergebnisse, während das Signatur-Matchmaking unterdurchschnittlich bleibt. Beide Verfahren ließen sich eventuell verbessern, wenn neben den linguistischen Verfahren zusätzlich Ansätze aus dem Text Mining in die Anfragegenerierung und das anschließende Discovery eingebracht würden.

Das Ziel, robust auf Änderungen eines Workflows einzugehen, in denen Services durch ähnliche Services ausgetauscht wurden, wurde für annähernd 100% der Fälle mit gleichzeitig sehr hoher Genauigkeit erreicht.

6.6 *Semantic Web Services Discovery*

Die beschriebenen Versuche legen die Frage nahe, inwieweit sich das Konzept-Matchmaking auch auf die semantische Suche nach Web Services übertragen lässt. Das heißt, lässt sich mit dem entwickelten Verfahren eine Ähnlichkeitssuche über Operationen regulärer Web Services durchführen, die lediglich durch WSDL beschrieben sind? Um dieser Frage nachzugehen wurde eine Testmenge von 40 Services mit insgesamt 186 Operationen aufgebaut. In dieser Testmenge finden sich auch die bereits beschriebenen 34 BLAST_k Operationen wieder.

Zu untersuchende Fragestellungen

Gegeben sei eine Service-Operation, dann sind alle die Service-Operationen gesucht, die eine semantisch äquivalente Funktionalität realisieren. Zur Untersuchung dieser Fragestellung wurde als Äquivalenzklasse die Alignment-Operation BLAST ausgewählt.

Erzielte Ergebnisse

Getestet wurden

- `TF-IDF`: das Keyword-Matchmaking, das das Vector Space Model in Kombination mit der TF-IDF nutzt,
- `C\C+L*`: die Erweiterung des Konzept-Matchmaking, die weder linguistische Methoden noch Kompositionen einsetzt, und
- `C+L\C*`: die Erweiterung des Konzept-Matchmaking, die linguistische Methoden aber keine Kompositionen verwendet.

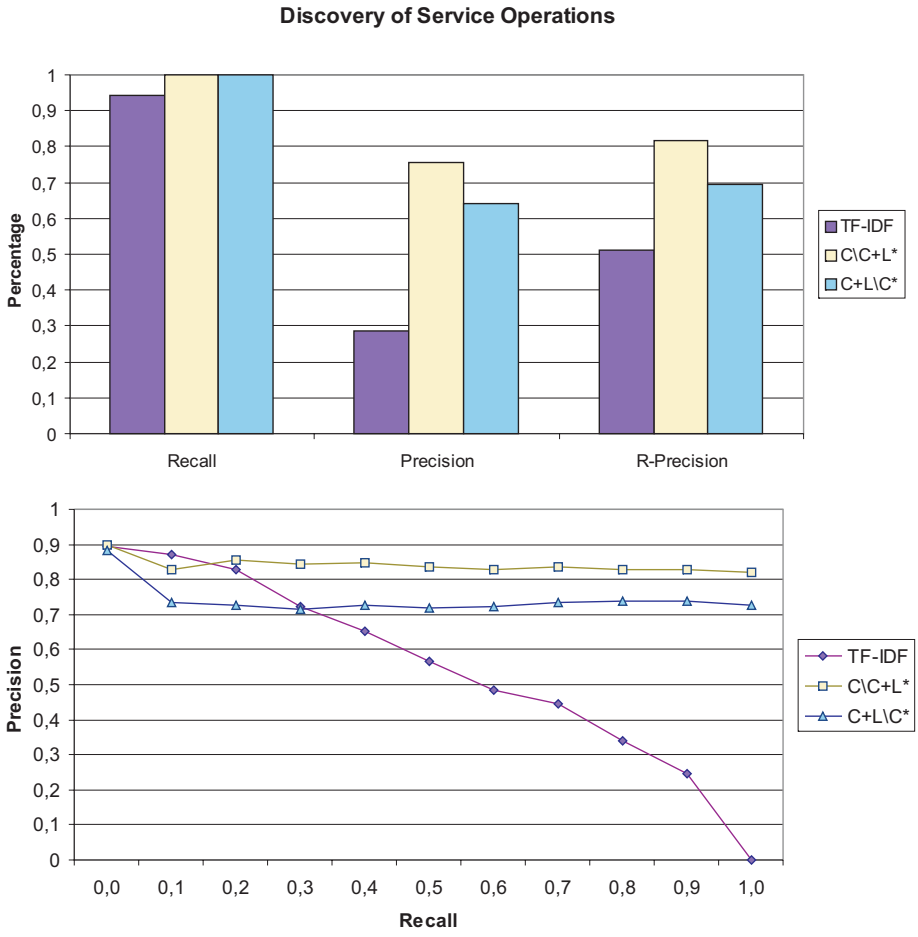

Abbildung 6.17: *Semantic Web Services Discovery*: (oben) Precision, Recall und R-Precision, (unten) Precision-Recall-Kurve

Wie bereits im zweiten Versuch wird das `TF-IDF` Verfahren als Eichmarke verwendet. Abbildung 6.17 beschreibt die erzielten Ergebnisse. Die `TF-IDF` besitzt ein Recall von ca. 94% bei einer Precision von unter 30%. Die R-Precision liegt hier bei knapp über 50%. Im Gegensatz dazu liefern die Konzept-Matchmaking-Einheiten `C\C+L*` und `C+L\C*` signifikant bessere Ergebnisse. Zum einen erreichen beide Varianten ein Recall von 100%. Zum anderen liegen die Precision-Werte bei `C\C+L*` bei ca. 75% bzw. bei `C+L\C*` bei ca. 64% und die R-Precision bei ca. 81% bzw. ca. 70%. Auch die Precision-Recall-Kurve (Abb. 6.17 (unten)) unterstützt diese Aussage und zeigt einen klaren Vorteil der Konzept-Matchmaking-Algorithmen gegenüber dem Standard `TF-IDF` Verfahren auf.

Abschließend sei jedoch erwähnt, dass, obwohl die restriktivere C\C+L* Variante die schwächere C+L\C* Variante in diesem Beispiel übertrumpft, die C+L\C* im Allgemeinen der C\C+L* zu bevorzugen ist. In diesem Beispiel identifiziert das Konzept-Matchmaking bereits ohne linguistische Methoden die nötigen Konzepte für das Discovery. Dies ist jedoch nicht der Normalfall, wie der zweite Versuch gezeigt hat (vgl. Abschnitt 6.5.2).

6.7 Zusammenfassung

Das Kapitel beschreibt einen einfachen Anwendungsfall, anhand dessen die QDAS-Prozedur in der Praxis dargelegt wird. Ferner wird auf die Aspekte der Erstellung der initialen Ausprägung und der Domänenontologie eingegangen. Die Arbeit eines Entwicklers eines neuen Service-Mediators wird vorgestellt und es werden verschiedene Vorgehensmöglichkeiten bei der Erstellung des Mediatorprofils und bei der Implementierung separat diskutiert. In diesem Kontext wird auch das entwickelte Anwenderwerkzeug der IRIS-Plattform vorgestellt.

Weiterhin wird anhand eines Benchmarks, welches das Problem der Service-Interoperabilität adressiert, die QDAS-Prozedur hinsichtlich Effektivität und Robustheit bewertet. Im Zentrum der Analyse steht die Frage, was der Benutzer bei Veränderung eines bereits mediatorinteroperablen Workflows unternehmen muss, um anschließend wieder einen mediatorinteroperablen Workflow zu erhalten? Diese Frage wird anhand zweier Versuche mit sukzessiv modifizierten Workflows eruiert. Das Ergebnis dieser Versuche zeigt, dass die entwickelten Verfahren der QDAS-Prozedur die bereits angepassten Service-Mediatoren mit hohem Recall und hoher Precision wiederentdecken. Die Anfragegenerierung und das Konzept-Matchmaking werden in einem letzten Versuch auf das Problem der Suche nach Web Services Operationen übertragen. Auch hierfür wird ein entsprechender Benchmark aufgestellt.

Die beschriebenen Versuche unterstützen ferner die These, dass das Retrieval über ontologischen Konzepten eine signifikante Verbesserung der Precision und des Recalls gegenüber Standardverfahren, wie beispielsweise dem Vector Space Model mit der TF-IDF, erzielt. Weiterhin zeigt sich, dass linguistiche Methoden ein erster Ansatz sind, den Recall bei der Suche zu erhöhen.

Kapitel 7

Diskussion

Man cannot discover new oceans
unless he has the courage to lose sight of the shore.
— André Gide (1869-1951), *Französischer Schriftsteller und Humanist*

7.1 Zusammenfassung

Gegenwärtig wird in der Softwaretechnologie die *Serviceorientierung* als neues Realisierungsparadigma propagiert. Ziel ist die bedarfsbezogene Aggregation lose gekoppelter *Services*. Diese Aggregationen werden gewöhnlich durch *Workflows* beschrieben. Ein zentrales Problem von hohem wirtschaftlichem Interesse ist die Überbrückung der Heterogenität der komponierten Dienste, d.h. die Erzielung der *Service-Interoperabilität*. Die Service-Interoperabilität kann technische, syntaktische, semantische oder politische Aspekte betreffen. Im Zentrum dieser Arbeit steht die syntaktische und semantische Service-Interoperabilität. Ohne eine besondere Unterstützung des Benutzers erfordert die Erzielung der Service-Interoperabilität einerseits einen hohen Wissensstand des Benutzer hinsichtlich der verwendeten Services, andererseits eine genaue Kenntnis des Benutzers über Werkzeuge zur Bildung von Softwarebrücken. Durch die rein manuelle Erstellung von Softwarebrücken bleibt das Erzielen der Service-Interoperabilität ein zeitaufwändiges und häufig fehleranfälliges Unterfangen. Gegenwärtige Ansätze zur Unterstützung des Benutzers stützen sich auf die Standardisierung der Schnittstellen und Datenformate oder auf die automatische Herleitung benötigter Softwarebrücken. Diese Verfahren stoßen jedoch an folgende Grenzen:

- Standardisierungsprozesse sind zeitaufwändig. Sie laufen aktuellen Entwicklungen und Anforderungen eines Anwendungsgebietes hinterher und haben gewöhnlich keine Möglichkeit, schnell auf veränderte Voraussetzungen zu reagieren. Zudem können die Bedürfnisse hoch-spezialisierter Applikationen nur teilweise abgedeckt werden.

- In Workflows werden teilweise komplexe Softwarebrücken zur Überwindung der Heterogenität benötigt, die von rein automatischen Ansätzen nicht hergeleitet werden können. Vorhandene, rein automatische Mediationsansätze unterstützen zudem keine Wiederverwendung und Anpassung einmal generierter Softwarebrücken in anderen Workflows.

Bei der Realisierung einer möglichst optimalen Unterstützung des Benutzers müssen verschiedene Herausforderungen adressiert werden:

1. Entwicklung eines offenen Verfahrens zur Erzielung der Service-Interoperbilität, welches Ergebnisse vorhandener Ansätze ausnutzt und einbezieht.

2. Entwicklung einer adäquaten Beschreibungssprache für Softwarebrücken, um diese flexibel in verschiedenen Workflows wiederverwenden und anpassen zu können.

3. Entwicklung von Verfahren, die benötigte Softwarebrücken identifizieren und gegebenenfalls auch neue, kombinierte Softwarebrücken entdecken.

Zur Lösung dieser Fragestellungen leistet die vorliegende Dissertation einen essentiellen Beitrag.

Service-Mediatoren und software-unterstützte Prozedur (QDAS-Prozedur)

In dieser Arbeit wird ein Konzept für anpassbare und wiederverwendbare *Service-Mediatoren* entwickelt, die über eine offene und erweiterbare, *software-unterstützte Prozedur* (semi-)automatisch identifiziert und problembezogen in einen Workflow eingebettet werden können. Die komplette Prozedur gliedert sich in Anfragegenerierung (Q*uery Generation*), Entdeckung (D*iscovery*), Anpassung (A*daptation*) und Bekanntmachung (S*toring*) der Service-Mediatoren, kurz QDAS. Derartige Service-Mediatoren überbrücken die Heterogenität der beteiligten Services und bilden so die Basis zur Erzielung der Service-Interoperabilität. Die entwickelten Konzepte werden durch die IRIS-Plattform prototypisch realisiert. Die offene Architektur und Entwicklung der QDAS-Prozedur erlaubt die Vorteile gängiger Ansätze, wie Ergebnisse des Schema-Matching, Text Mining oder Information Retrieval, zu integrieren und vorhandene Standards bei der Realisierung neuer Service-Mediatoren zu nutzen.

Mediatorprofilsprache (MPL)

Für die verschiedenen Phasen der QDAS-Prozedur ist eine entsprechende Beschreibungssprache essentiell, mit der die Fähigkeiten inklusive vorhandener Anpassungsmöglichkeiten eines Service-Mediators sowohl syntaktisch als auch semantisch beschrieben werden können.

Die in dieser Arbeit auf Basis von OWL entwickelte *Mediator Profil Language* (MPL) erfüllt diese Anforderungen und bildet den Grundstock für das Komponentenmodell der Service-Mediatoren. MPL unterteilt sich in einen *Mediatorprofilbereich*, in dem die Schnittstelle eines Service-Mediators definiert wird, und in einen *Prozessbereich*, in dem komplexe Service-Mediatoren über Kompositionsgraphen spezifiziert werden können. Außerdem ermöglicht MPL die *Konfigurierung* einfacher und komplexer Service-Mediatoren. Die semantische Auszeichnung des Mediatorprofils erfolgt über Konzepte einer Domänenontologie. Das Besondere an MPL ist das erstmalige stringente Zusammenbringen dieser Annotation mit den verschiedenen Anpassungs- und Konfigurationsfähigkeiten.

Anfragegenerierung und Discovery

Die Einbettung und Wiederverwendung einmal entwickelter Service-Mediatoren in verschiedene Workflows erfordert eine gezielte Suche und Identifikation, sowie eine adäquate Anpassungsmöglichkeit benötigter Service-Mediatoren. Vor allem der Suchprozess ist ein essentieller aber auch besonders schwieriger Bestandteil der QDAS-Prozedur, insbesondere, wenn erst mehrere, geeignet verknüpfte Service-Mediatoren zusammen die Service-Interoperabilität erreichen und diese Kombinationen schon bei der Suche erkannt werden müssen.

Die in dieser Arbeit entwickelten Suchalgorithmen identifizieren Service-Mediatoren und neue Kompositionen von Service-Mediatoren (komplexe Service-Mediatoren), die innerhalb eines Workflows benötigt werden. Dabei beziehen sich die Algorithmen sowohl auf syntaktische als auch semantische Merkmale (*Discovery*). Die konkreten Anforderungen an benötigte Service-Mediatoren werden in Form MPL-basierter Anfrageprofile beschrieben. Trotz der den Servicebeschreibung innewohnende Unschärfe wird ein Verfahren realisiert, welches die Servicebeschreibungen auf MPL abbildet und dabei gleichzeitig eine automatische Annotation durch die Konzepte einer Domänenontologie vornimmt (*Anfragegenerierung*).

Praktische Experimente zeigen, dass benötigte, teilweise angepasste Service-Mediatoren mit hoher *Precision* und *Recall* entdeckt werden. Vor allem ontologiebasierte Retrievalverfahren zeigen signifikant bessere Ergebnisse, wenn man sie mit Standard-IR-Techniken vergleicht.

7.2 Zukünftige Forschungsrichtungen

Die QDAS-Prozedur und das entwickelte IRIS-Framework bilden eine offene und erweiterbare Plattform zur Überbrückung der Interoperabilität in servicebasierten Workflows. Die Ansätze wurden auf Basis der Web Services Technologie entwickelt und im Rahmen biologischer Fragestellungen untersucht. Sie sind aber weder auf diese Technologie noch auf diese Anwendungsdomäne beschränkt, sondern lassen sich auch auf Technologien wie Grid Services oder CORBA Services und andere Anwendungsdomänen übertragbar. Hierzu wären weitere Domänenontologien und Service-Mediatoren nötig. Hinsichtlich der Erstellung der initialen Ausprägung und der Domänenontologie besteht weiterer Forschungsbedarf, um den Mediator- und Ontologiedesigner besser zu unterstützen. Über Text Mining Ansätze in Kombination mit linguistischen Analysen lassen sich beispielsweise aus den Workflow- und Servicebeschreibungen Wortgruppen (Konzepte) herleiten und in Beziehung setzen. Diese Verfahren sollten in eine computer-unterstützten Entwicklungsumgebung eingebettet werden.

Die offene und erweiterbare Plattform erlaubt zudem, weitere Algorithmen und Methoden in das System zu integrieren. Zukünftige Forschungsrichtungen erstrecken sich hier über die Anfrageherleitung und das anschließende Retrieval, sowie die Komposition von Services und Service-Mediatoren, die Erweiterung von Beschreibungssprachen und Typsystemen und die automatische Anpassung und Konfigurierung. Einige Herausforderungen und Forschungsansätze wurden bereits in den verschiedenen Kapiteln diskutiert und werden hier nochmals zusammengefasst vorgestellt.

Anfragegenerierung und Retrieval

Die Abbildung der inhaltlichen Beschreibungen auf Konzepte einer Ontologie ist ein schwieriges Unterfangen. Die vorliegende Arbeit stellt einen Basisalgorithmus für dieses Problem vor, der sich im Grundgedanken auf einen linguistischen Ansatz stützt. Zukünftige Erweiterungen, die den Rahmen diese Arbeit gesprengt hätten, sollten neue Fuzzy-Ansätze aus dem Bereich Data Mining und Text Mining einsetzen. Beispielsweise könnten über Assoziationsregeln Wortgruppen berechnet und anschließend einzelnen Konzepten einer Ontologie zugeordnet werden. Auf diese Weise wäre es auch ohne Termgleichheit und linguistische Verfahren möglich, die Zugehörigkeit eines Services zu einem ontologischen Konzept zu identifizieren. Hier liefert das kürzlich von Dong et al. (2004) beschriebene, auf Assoziationsregeln aufsetzende Clusterverfahren einen ersten Ausgangspunkt. Die Integration dieses Verfahrens in die IRIS-Plattform und die anschließende Evaluierung ist Gegenstand einer laufenden Diplomarbeit (Mohebbian, 2006).

Das Discovery nutzt gegenwärtig die Informationen, die durch die Services bereitgestellt werden. Obwohl die realisierten Methoden bereits jetzt sehr gute Ergebnisse liefern, wäre es sinnvoll, den Kontext, d.h. den konkreten Workflow, in dem die Services eingesetzt werden, während des Retrievals zusätzlich zu berücksichtigen. Sind beim Retrieval verschiede-

ne Kompositionen möglich, könnte eventuell das Hintergrundwissen über die vorhandenen Workflows die Komposition auswählen, die wahrscheinlich die Anforderungen am besten erfüllt. Zukünftige Retrievalverfahren sollten daher *kontext-sensitiv* (*context-aware*) ablaufen und Informationen früherer Workflows mit in das Discovery aufnehmen.

Beschreibungssprachen und Typssyteme

Gegenwärtige Typsysteme klassischer Programmiersprachen erlauben lediglich die Beschreibung der syntaktischen Aspekte eines Datentyps. Zukünftig wäre es hilfreich, wenn die durch ein Typsystem beschriebenen Daten auch semantische, maschinen-verarbeitbare Informationen bereitstellen. Dies würde u. a. die bessere Nutzung und Wiederverwendung von Softwarebibliotheken und Softwarekomponenten ermöglichen. Ontologiebeschreibungssprachen wie OWL liefern hier eine gute Grundlage. Der Einsatz ontologiebasierter Typsysteme kann außerdem dazu genutzt werden, Software bereits bei der Kompilierung auf semantische Fehler zu überprüfen, wie es derzeit lediglich für die Syntax möglich ist.

Anpassung und Konfigurierung

Die Verbesserung der (semi-)automatischen Konfigurierung komponentenbasierter Software eröffnet ein weiteres Forschungsfeld. Neben der Komposition, die in dieser Arbeit über Signatur- bzw. Ontologieinformationen erfolgt, gilt es, auch zukünftig die Konfigurierung weitestgehend automatisch herzuleiten und durchzuführen. Dies ist vor allem bei komplexen Komponenten, d.h. Komponenten, die aus Unterkomponenten bestehen, extrem schwierig. Es ist nicht ohne weiteres zu erkennen, welche Konfigurationselemente eine komplexe Komponente von ihren Unterkomponenten erbt und wie diese gegebenenfalls zu belegen sind. Kurz gesagt, die für Service-Mediatoren definierten Eigenschaftsgraphen sind nicht bekannt und lassen sich derzeit nur unzureichend aus den generierten Kompositionen herleiten. In einem ersten Ansatz könnte eventuell auf vorhandenes Wissen zurückgegriffen werden, wenn bereits komponierte Einheiten vorhanden sind. Vielleicht ließe sich zudem über Ähnlichkeitssuche einzelner Komponenten auch auf Konfigurierungen anderer Komponenten schließen.

Anhang A

Web Services Plattformen und verwandte Technologien

Schon Mitte 2002 hat *Cap Gemini Ernst & Young* eine Studie[1] aufgestellt, die besagte, dass die Web Services Technologie eine zentrale Rolle im Bereich des verteilten Rechnens einnimmt bzw. einnehmen wird. Laut dieser Studie hatten bereits rund ein Drittel der in der Studie befragten Unternehmen eine Strategie zum Einsatz dieser neuen Technologie und fast genau so viele betrachteten Web Services als bedeutsames oder sehr bedeutsames Thema, Tendenz steigend. Viele Unternehmen erhoffen sich durch den Einsatz von Web Services schnell und flexibel an sich permanent wandelnde Anforderungen anpassen zu können.

Laut Schätzung der *Gartner Group*[2] werden Web Services bis Ende 2004 das Hauptentwicklungswerkzeug für neue Applikationen darstellen. Ihrer Ansicht nach wird der Einsatz von Web Services die Kosten reduzieren und die Effizienz von IT-Entwicklungsprojekten um 30 Prozent steigern. Ob diese Prognosen so tatsächlich eintreffen bzw. eingetroffen sind bleibt fraglich, Fakt ist jedoch, dass Web Services aus der Softwaretechnik zur Realisierung serviceorientierter Architekturen im Allgemeinen und zur *Enterprise Application Integration* im Speziellen nicht mehr wegzudenken sind.

Derzeit findet der Einsatz am häufigsten in Bereichen des eCommerce, B2B, Marketing, Vertrieb und Customer Relationship Management statt. Web Services setzen sich aber nicht nur in kommerziellen Bereichen durch (Brandner et al., 2004), sondern sind ebenfalls Gegenstand intensiver Forschung und gewinnen auch in naturwissenschaftlichen Anwendungsgebieten, wie den Geo- oder Biowissenschaften, zunehmend an Bedeutung. Letzteres

[1] Die Studie wurde Mitte 2002 von *Cap Gemini Ernst & Young* mit 170 Teilnehmern aus bundesdeutschen Unternehmen durchgeführt, wovon 108 der Befragten angaben, mit dem Begriff *Web Services* gut oder recht gut vertraut zu sein. Eine detaillierte Übersicht findet man unter `www.capgemini.de/servlet/PB/menu/1004619/index.html`

[2] `www3.gartner.com/DisplayDocument?id=350986` und `www3.gartner.com/DisplayDocument?id=344028`

wird vor allem durch die Weiterentwicklung der Grid Plattformen und die Annäherung dieser Technik an die Web Services weiter vorangetrieben (Cremers et al., 2005).

Ein weiterer Grund für die rasche Verbreitung und die große Aufmerksamkeit der Web Services Technologie ist darin begründet, dass führende IT-Hersteller, wie Microsoft, HP, IBM, SUN oder Oracle, den Basistechnologie-Stack der Web Services unterstützen und somit auf technischer Ebene Interoperabilität erreicht wurde.

Im folgenden wird die Web Services Architektur (WSA) dem bis dato gängigen Kommunikationsstandard CORBA gegenübergestellt. Im Anschluss daran werden zwei konkrete, aber konkurrierende Realisierung dieser Architektur, die J2EE und .NET Plattform, betrachtet und aus Sicht der Softwaretechnik kurz verglichen.

Neben den Web Services etablieren sich die so genannten Grid Services und Peer Services. Deren Annäherung bzw. deren Zusammenhänge zur WSA wird in den letzten beiden Abschnitten dieses Kapitels Rechnung getragen.

A.1 CORBA, J2EE und .NET

Die WSA ist eine spezielle Instanz der SOA, die ähnlich wie CORBA oder Java RMI verteiltes Rechnen adressiert (Kossmann und Leymann, 2004). Zudem lassen sich Web Services als verteilte, virtuelle Komponenten verstehen, die ihrerseits jedoch mehr *self-contained* sind als typische Komponenten, d.h. sie eröffnen nicht explizite Kontextabhängigkeiten zu anderen Diensten (Szyperski et al., 2002). Sie lassen sich prinzipiell durch verschiedene Technologien realisieren, da sie unabhängig sind von jeglichen Komponentenmodellen, Programmiersprachen, Betriebssystemen und Plattformen. Zwei der bekanntesten Realsierungsplattformen sind Microsoft's .NET und Sun's J2EE.

Common Object Request Broker Architecture

Abbildung A.1 beschreibt den Zusammenhang zwischen der *Common Object Request Broker Architecture* (CORBA) und der Web Services Technologie hinsichtlich des Aspekts des verteilten Rechnens. WSDL entspricht bzw. erweitert die *Interface Definition Language* (IDL) von CORBA in der Art, dass die Sprach- und Plattformunabhängigkeit der IDL gewahrt bleibt, und zusätzlich eine Unabhängigkeit des genutzten Nachrichtenformats und Übertragungsprotokolls erreicht wird. Denn WSDL ist nicht auf die Übertragung von SOAP-Nachrichten über HTTP fixiert. Es können jederzeit neue Bindungen für andere Protokolle eingesetzt werden. Ferner erlaubt WSDL neue Ports zu definieren, ohne dabei vorhandene Ports zu verändern.

Nachrichten werden in CORBA gewöhnlich durch das *Internet Inter-ORB Protocol* (IIOP) beschrieben. Das IIOP ist eine Spezialisierung des *General Inter-ORB Protocol* (GIOP)

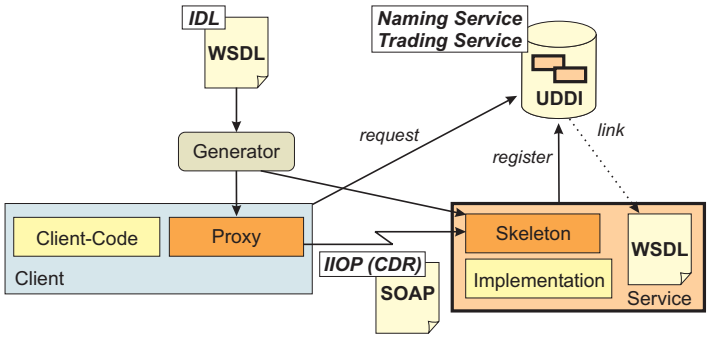

Abbildung A.1: Grundkonzepte des RPCs bei Web Services und CORBA.

über TCP/IP. Beide Standards wurden entwickelt, um die Interoperabilität von unterschiedlichen ORB-Implementierung zu gewährleisten. Innerhalb des GIOP wurde die *Common Data Representation* (CDR) und das *GIOP-Nachrichtenformat* definiert. Das CDR beschreibt, wie die Werte der IDL-Typen übertragen werden, während das Nachrichtenformat den Aufbau einer Nachricht spezifiziert. Diese Einheiten des GIOP entsprechen im wesentlichen den SOAP-Nachrichten der Web Services Technologie.

Auffinden lassen sich CORBA Objekte durch die in der *Object Management Architecture* (OMA) spezifizierten *CORBAservices*. Hier zu nennen ist der so genannte *Naming Service* sowie der *Trader Service*. Derzeit spezifiziert die OMA vierzehn weitere CORBAservices, die sich ebenfalls mit Aspekten wie Events, Notification, Security oder Transaktionsmanagement befassen. Der Naming Service ist vergleichbar mit den White Pages des UDDI, während der Trader Service zu den Yellow Pages des UDDI korrespondiert (Newport, 2001; Vasudevan, 2001). Auch diese Dienste nutzen Namensgleichheit oder Schlüsselworte zur Suche.

Ein ähnlicher Zusammenhang lässt sich auch zwischen Java RMI und Web Services herstellen. Bei Java RMI wird durch den *rmic-Compiler* aus dem Java Interface entsprechende Client Stubs und Server Skeletons generiert. Die Daten lassen sich mit dem Standard Serialisierungsmechanismus von Java verpacken und über das Netz verschieben. Heute sollte hierbei ebenfalls das IIOP genutzt werden (Java RMI/IIOP). RMI bietet ferner einen Registry Service im Sinne eines Naming Service (*Java Naming and Directory Interface* – JNDI). Durch die Umsetzung von RMI/IIOP wurde eine Brücke von Java RMI nach CORBA geschlagen.

Im direkten Vergleich zu CORBA oder Java RMI sind es die einfache Anwendung, die verbesserte Unterstützung von Programmiersprachen und die große Anzahl vorhandener Entwicklungswerkzeuge, die Web Services einen entscheidenen Vorteil verschaffen (Brandner et al., 2004). Durch die XML-basierten Nachrichtenprotokolle und die Nutzung der Internetprotokolle wie HTTP lassen sich prinzipiell jegliche Programmiersprachen anbinden, die textuelle Daten verarbeiten und auf HTTP-Requests reagieren können. Es werden

keine zusätzlichen Sprachbindungen, wie es bei CORBA der Fall ist, benötigt, da die XML-Dokumente nur entsprechend verarbeitet werden müssen.

Durch den Einsatz von XML und die breite Nutzung und Verbreitung der Internettechnologie haben Web Services ebenfalls einen Vorteil gegenüber CORBA oder Java RMI. Viele Firmen besitzen bereits Web Portale und können ihre bis dato rein für den menschlichen Benutzer spezifizierten Dienste jetzt recht schnell und ohne großen Aufwand auch für die maschinelle Nutzung bereitstellen. Dies gilt auch für Investitionen in andere Komponenten, wie beispielsweise COM (*Component Object Model*) oder JavaBeans, die sehr leicht als Web Services publiziert werden können (Glass, 2000).

Aufgrund der Tatsache, dass SOAP über verschiedene Protokolle, wie HTTP(S) oder SMTP, übertragen werden kann, ist es zudem möglich, durch die meisten Firewalls hindurch Nachrichten zu übermitteln. Dies ist bei anderen Protokollen, wie beispielsweise CORBA's IIOP normalerweise nicht möglich (Keidl et al., 2003).

CORBA und Java RMI haben jedoch den Vorteil, dass sie effizienter als die Codierung über SOAP sind, was im wesentlichen daran liegt, das SOAP-Dokumente sehr gross (ca. Faktor vier im Gegensatz zu binär-kodierten Daten) und nicht binär kodiert werden (höhere Verarbeitungszeit) (Reinefeld und Schintke, 2004). Jedoch gibt es Ansätze dieses Problem durch geschickte Kompression und reduzierte Übertragung zu beheben. Beispielsweise verwenden Werner et al. (2004) ein Verfahren, dass auf Differential Encoding beruht, bei dem nur der Teil der SOAP-Nachricht übertragen wird, der sich bei jedem Aufruf ändert. Dieser Teil wird zusätzlich geschickt komprimiert.

Java 2 Enterprise Edition

Wie bereits erwähnt unterliegen Web Services keinem konkreten Komponentenmodell noch einer konkreten Plattform. Eine Realisierungsplattform, die sich zur Erstellung von Geschäftsanwendungen eignet und ebenfalls Web Services unterstützt, ist die von Sun spezifizierte *Java 2 Enterprise Edition* (J2EE) in der Version 1.4. Abbildung A.2 beschreibt die APIs, die der kürzlich erschienenen J2EE Plattform Version 1.4 angehören, und gruppiert diese den jeweiligen J2EE Containern entsprechend (Armstrong et al., 2004).

Diese neue Version der J2EE steht im Zeichen der Interoperabilität, d.h. die Anbindung an die Web Services Technologie ist nun offizieller Bestandteil dieser Spezifikation (Ibba, 2004). Es ist schon erstaunlich, dass diese Technologie erst kürzlich Einzug in die J2EE Plattform gefunden hat – die Version 1.4 wurde am 24. November 2003 veröffentlicht. Bis dato konnte ein Web Service in J2EE nur durch eine JavaServer Page (JSP) oder ein Servlet realisiert werden. Ein Konsument konnte den Service aktivieren, indem er eine XML-Nachricht an die entsprechende URL schickte, die die JSP Seite oder das Servlet repräsentiert (Manes, 2001). Diese Möglichkeit besteht, da JSPs und Servlets auf HTTP Requests reagieren können. Das Sun's ONE (*open net environment*) erweiterte die damalige J2EE mit Java

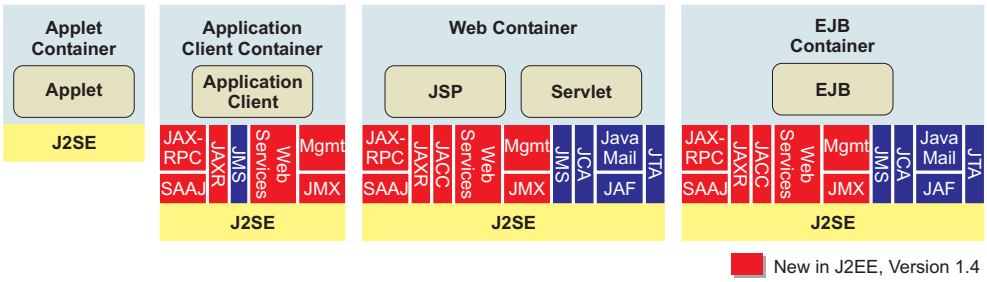

Abbildung A.2: APIs der J2EE Plattform Version 1.4.

APIs zum Bearbeiten und Nutzen von XML und XML-basierten Protokollen. Diese APIs wurden nun ebenfalls mit in die J2EE 1.4 aufgenommen (vgl. Abb. A.2).

Da die Vorstellung der kompletten J2EE Plattform den Rahmen sprengen würde – hier sei auf ausgewählte Literatur verwiesen (Armstrong et al., 2004; Kao, 2001) – sollen nur die wesentlichen Technologien und ihre Neuerungen hinsichtlich Web Services besprochen werden. Allgemein setzt die J2EE Plattform auf der Programmiersprache Java auf, genauer gesagt auf der *Java 2 Standard Edition* (J2SE). Aus diesem Grund ist J2EE zwar auf Java fixiert, jedoch frei in der Wahl des Anbieters sowie des verwendeten Betriebssystems.

Herzstück dieser Plattform stellen die so genannten *Enterprise JavaBeans* (EJBs) dar. Hierbei handelt es sich um serverseitige Komponenten, die Geschäftslogik in modulare Einheiten verpaken (Adatia, 2001; Roman et al., 2002). Diese Komponenten können anschließend alleine oder mit anderen EJBs kombiniert werden, um ein Geschäftsproblem zu lösen. Innerhalb der EJBs lassen sich drei Klassen identifizieren: Session Beans, Entity Beans und Message-Driven Beans.

Session Beans stellen ausführbare Geschäftslogik bereit, deren Lebensdauer durch die einer (Client) Session vorgegeben ist. Im Gegensatz dazu beschreiben *Entity Beans* persistente Objekte. Ihre Lebensdauer endet nicht mit Beendigung einer Session. Weiterhin lassen sich Session Beans in zustandlose (*stateless*) und zustandsbehaftete (*stateful*) Beans einteilen. Hinsichtlich der Entity Beans ist es seit Version 1.4 möglich read-only Entity Beans definieren zu können. Da diese Beans keine Veränderungen auf der Datenbank vornehmen können, besteht hier neues Optimierungspotential für Serveranwendungen. Mit der Notwendigkeit auch asynchrone Nachrichten innerhalb der J2EE Plattform durch EJBs verarbeiten zu können, wurden schließlich noch *Message-Driven Beans* (MDB) definiert. Diese stellen im wesentlichen ein Pendant der Session Beans dar.

> **Beispiel A.1.** Ein Modul, das die Kreditwürdigkeit eines Kunden überprüft würde als stateless Session Bean realisiert, während der Account des Kunden auf eine Entity Bean abgebildet würde. Der Warenkorb eines Kunden beim eShoping würde durch eine stateful Session Bean realisiert, damit der Kunde mehrere Produkte in den Warenkorb legen kann und dieser nicht nach jeder Aktion wieder leer ist.

EJBs erlauben in der neuen Version die direkte Anbindung an Web Services, d.h. eine stateless Session Bean kann direkt als Port eines Services dienen, ohne den zusätzlichen Umweg über eine JSP gehen zu müssen (vgl. (Manes, 2001)). Message-Driven Beans (MDB) die bis jetzt an den *Java Message Service* (JMS) gebunden waren, können nun auch andere asynchrone Nachrichtendienste verarbeiten. Hierzu wurden Schnittstellen bereitgestellt, um Nachrichten des *Java API for XML Messaging* (JAXM) empfangen zu können. JAXM ist ein auf SOAP basierendes Nachrichtenframework. Somit können durch MDB asynchrone Web Services realisiert werden. Leider ist das JAXM kein Bestandteil der J2EE und muss zusätzlich installiert werden.

Auch bei den Technologien der Präsentationsschicht, d.h. den *JavaServer Pages* (JSP) und den *Servlets*, hat sich einiges verändert. Wesentliche Neuerung ist die Definition und Integration der Version 1.1 der *Java-Server Standard Tag Library* (JSTL). So genannte *Tag Libraries* erlauben die Definition von eigenen Tags, die an eine konkrete Implementierung, z.B. EJB oder JavaBean, gekoppelt werden können und beim Lesen der JSP entsprechend ausgeführt werden. Die JSTL 1.1 stellt eine standardisierte Tag-Library dar, die bis dahin vorhandene proprietäre Bibliotheken ablöst. Das Pendant zu den WebForms bei .NET, die so genannten *JavaServer Faces* (JSF) haben noch nicht den Einzug in die J2EE gefunden. Dieses könnte daran liegen, dass sie bis zu diesem Zeitpunkt noch nicht endgültig standardisiert sind und es zur Zeit andere Framework gibt, die die Lücke füllen, wie beispielsweise das Struts-Framework.

Neu aufgenommen in die J2EE – und dies ist für die Web Services Technologie zentral – sind die *Java API for XML Processing* (JAXP) als Grundlage zum Verarbeiten von XML-Dokumenten, die *Java API for XML-based Remote Procedure Call* (JAX-RPC), die *SOAP with Attachments API* (SAAJ) und die *Java API for XML Registries* (JAXR). Ein *WS-Compiler* übernimmt die Rolle des WSDL2Java und Java2WSDL, wie sie beispielsweise in Axis[3] realisiert sind. Damit lassen sich flexibel Skeletons und Stubs für die statische Kommunikation über SOAP generieren.

Eine Kernkomponente der XML-verarbeitenden APIs ist das JAX-RPC. Diese Technologie versteckt die Komplexität des XML-Messaging komplett vor dem Entwickler und bietet die gleichen Möglichkeiten, wie sie auch Java RMI ermöglicht. Hierzu spezifiziert JAX-RPC ein Type-Mapping zwischen WSDL und Java, wodurch das Marshalling und Unmarshalling von Java-Objekten zu XML-Elementen und umgekehrt erfolgen kann. Um ferner SOAP-basierte Nachrichten verarbeiten zu können, nutzt JAX-RPC im Hintergrund SAAJ.

JAXR erlaubt eine Kommunikation mit UDDI, ohne auf die Ebene der XML-Dokumente absteigen zu müssen, d.h. eigene SOAP-Nachrichten zu erstellen oder eine API (SAAJ), die dies ermöglicht, zu nutzen. JAXR ist dabei prinzipiell nicht auf UDDI festgelegt, sondern kann auch andere Registrierungen ansprechen, beispielsweise die ebXML-Registry. Dies wird dadurch erreicht, dass JAXR eine abstrakte API festlegt, die von konkreten Providern implementiert werden kann (Wang, 2003).

[3] `ws.apache.org/axis2/`

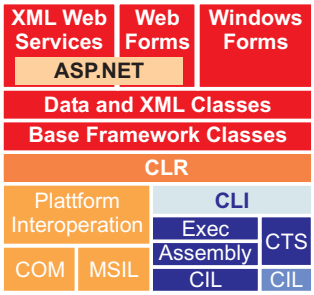

Abbildung A.3: Überblick über das .NET Framework.

Microsoft .NET

Microsoft's .NET ist das Resultat einer kontinuierlichen Weiterentwicklung vorhandener Komponenten- und Servertechnologie hin zu einer Plattform, die im Kern Web Services unterstützt. Weiterhin dient dieses Framework als Deployment-Plattform (Server und Clients) sowie als Entwicklungsumgebung (Szyperski et al., 2002). Verschiedene Serverprodukte und der Windows.NET Server wurden über mehrere Schritte hinweg in eine Plattform transformiert, die Web Services und die Verarbeitung von XML unterstützen. Aufgrund der Komplexität des Frameworks kann in dieser Arbeit nur ein kurzer Einblick ermöglicht werden. Abbildung A.3 illustriert einen groben Überblick über das .NET Framework.

Das .NET Framework besteht aus der *Common Language Runtime* (CLR), einer großen Anzahl von Frameworks (Interfaces und Klassen), die in Assemblies zusammengefasst sind, sowie einer Anzahl von Werkzeugen.

Die CLR ist eine Implementierung der *Common Language Interface* (CLI) Spezifikation. CLR geht über die CLI Spezifikation hinaus, da sie die Integration des COM sowie Zugriffsdienste auf die Windows Plattform, d.h. Win32 oder andere DLL-basierte APIs, ermöglicht. Aufgrund dieser Integration kann die CLR-Ausführungseinheit optimale Performanzresultate liefern (Szyperski et al., 2002).

Die CLI-Spezifikation etabliert eine sprachneutrale Plattform, ähnlich wie CORBA. Im Unterschied zu CORBA definiert sie weiterhin eine *Intermediate Language* (IL) und ein Deployment-Dateiformat (*Assemblies*). Ferner beinhaltet CLI Spezifikationen für Ausführungsdienste (Loader, JIT-Compiler, Garbage-Collector, Memory-Manager), die *Common Type System* (CTS) und die *Common Language Specification* (CLS) (Szyperski et al., 2002). Man kann sagen, dass die CTS eine Obermenge über viele in verschiedenen Programmiersprachen verwendeten Typsysteme bildet. Da jedoch zwei beliebige Sprachen häufig nicht die exakt gleichen Typsysteme nutzen, ist es sinnvoll eine Untermenge der CTS bereitzustellen, die von sehr vielen Sprachen unterstützt wird. Auf diese Weise lässt sich die Interoperabilität dieser Sprachen erreichen. Dieses liefert die CLS. CLS ist eine Untermenge des CTS, die so konzipiert wurde, dass sie möglichst viele Sprachen vollständig

unterstützen. Einen Überblick über die CTS-Typhierarchie findet sich in (Szyperski et al., 2002).

CLR stellt eine neue Komponenteninfrastruktur bereit. Wie die Java Virtual Machine (JVM) definiert die CLR eine virtuelle Anweisungsmenge, die prozessorunabhängig ist, aber im Gegensatz zur JVM auch Komponenten ermöglicht, die Zugriff auf eine spezielle Plattform benötigen. Die CLR ist zuständig für Runtime-Dienste, wie Sprachintegration, Speicherverwaltung, Prozess- und Thread-Management. Derzeit werden von Microsoft vier Sprachen angeboten, die auf dem sprachneutralen CLR basieren: C#, JScript, Managed C++ und Visual Basic.NET.

C# ist eine objektorientierte Programmiersprache, die an Java und C++ angelehnt ist. Sie bietet direkte Unterstützung zu fast allen CLR-Möglichkeiten. Im Gegensatz zu Java bietet C# eine Syntax um Properties auszudrücken. Eine *Property* ist ein abstraktes Feld, das eine get-Methode, set-Methode, oder beides besitzen kann. Properties können in C# nicht nur auf Klassen definiert werden, sondern auch auf Interfaces und Structs. Durch Weglassen einer getter- oder setter-Methode werden Properties write- oder read-only.

Aufbauend auf der CLR bietet das .NET Framework eine große Anzahl weiterer Frameworks. Hierzu zählen die *Base Framework Classes*, aber auch das Framework für Internet, XML Standards und Web Services. Die Basisklassen bieten Standardfunktionalitäten wie Input/Output, String-Manipulation, Security- oder Thread-Management. Die XML-Klassen erlauben semistruktierte Daten in Form von XML zu manipulieren, innerhalb dieser Daten zu suchen und Transformation vorzunehmen. Dahingegen ermöglichen die Daten-Klassen den Zugriff und die Manipulation von relationalen Daten, die durch ADO.NET verwaltet werden. Das ADO.NET Framework (*active data objects* – ADO) unterstützt hierzu OLE-DB (*Object Linking and Embedding* for *Databases*) und ODBC (*OLE Database Connectivity*) um relationale Datenquellen ansprechen zu können.

Die CLI-Deployment-Einheiten werden *Assemblies* genannt. Sie stellen die neuen .NET Softwarekomponenten dar. Assemblies sind die Einheiten, die in .NET deployed, versioniert und verwaltet werden. Ein Assembly ist in seiner Gesamtheit einem JAR-File der J2EE sehr ähnlich. Innerhalb der Assemblies wird der Code durch die *Common Intermediate Language* (CIL) ausgedrückt. Diese lässt sich mit dem Java-Bytecode vergleichen. Die *Microsoft Intermediate Language* (MSIL) ist eine CIL-fähige Obermenge, die zudem Instruktionen erlaubt, um spezielle Fähigkeiten der CLR auszudrücken, die über die CLI-Spezifikation hinausgehen (wie COM-Anbindung oder Plattform Interoperation).

Einen zentralen Kern der .NET Plattform stellt die integrierte Entwicklungsumgebung Visual Studio .NET dar. Visual Studio .NET setzt auf der CLR auf und unterstützt somit alle obengenannten Sprachen. Prinzipiell benötigt ein .NET Entwickler für den kompletten Entwicklungsprozess nur dieses eine Werkzeug. Weiterhin ist es möglich spezielle Editoren und Debugger in Visual Studio .NET zu integrieren.

Das Pendant zu den Technologien der Präsentationsschicht in J2EE sind in .NET die *Windows Forms* und *Web Forms*. Die Windows Forms erlauben eine einfache und schnelle

Konstruktion von desktop-basierten Windows-Applikationen. Um Controls zu erstellen und mit dem Benutzer zu interagieren, benutzen die Windows Forms die Windows-Plattform. Hierdurch sind die resultierenden Applikationen nicht plattformunabhängig.

Web Forms ist ein Framework, das dazu genutzt werden kann, um Dialoge innerhalb von ASP.NET (*Active Server Pages* (ASP)) zu erstellen, die in Form vom dynamischen HTML Seiten (DHTML) dem Nutzer dargeboten werden. Web Forms benötigen einen Browser um Forms anzuzeigen und mit dem Benutzer zu interagieren.

Eine zentrale Komponente des .NET Frameworks ist hierbei das ASP.NET, das dem Entwickler eine einheitliche Entwicklungsplattform zur Erstellung von unternehmensweiten Webanwendungen und die dazu erforderlichen Dienste bereitstellt. ASP.NET ist nicht nur der Nachfolger der ASP, sondern setzt direkt auf der Windows Server Plattform auf, wodurch eine sehr hohe Performanz erreicht wird, gleichzeitig aber auch Plattformabhängigkeit erzwungen wird. Die ASP.NET Technologie lässt sich mit Java ServerPages und Java Servlets vergleichen. ASP.NET kann weiterhin eingesetzt werden, um einfache Web Services zu erstellen. Um jedoch komplexere Web Services zu realisieren sind verschiedene weitere Werkzeuge im .NET Framework integriert. Hierbei muss dann nicht auf ASP.NET zurückgegriffen werden. Beispielsweise generiert das *wsdl.exe* Werkzeug ausgehend von WSDL-Dokumenten Code (Proxies) für den Web Service als auch den entsprechenden Client.

J2EE versus .NET – Ein kurzes Fazit

Viele Anbieter unterstützen heute die Entwicklung und das Deployment von Web Services. Zwei der wohl bekanntesten Entwicklungsumgebungen sind Sun's J2EE beispielsweise in der Sun's ONE Plattform realisiert und Microsoft's .NET. Beide Plattformen unterstützen die Entwicklung von Geschäftseinheiten und das Deployment als Web Service und beide Plattformen haben ihre Vorzüge und Nachteile.

Im Gegensatz zur oben beschriebenen J2EE Plattform hat Microsoft's .NET den entscheidenen Vorteil, dass Entwicklung und Spezifikation aus einer Hand stammen und sich auf ein Produkt konzentrieren, welches kontinuierlich überarbeitet und weiterentwickelt wird. Über die *Visual Basic Controls* (VBX), das OLE, das COM, die ODBC, ActiveX und die ASP mündet dies in der heutigen .NET Spezifikation. Dieses kann nicht nur Performanzvorteile bringen (.NET ist verflochten mit der Windows Oberfläche), sondern äußert sich auch in einer einheitlichen Entwicklungsumgebung (Visual Studio .NET). Die integrierte Nutzung des Visual Studios .NET zeichnet einen Kernvorteil der .NET Plattform aus (Langner, 2003). Zudem ist die .NET Plattform sprachunabhängig, wodurch Entwickler sich auf ihre vorhandenen Kenntnisse zurückziehen können, um Business-Objekte zu realisieren.

Die andere Seite der Medaille ist, dass man auf auf ein Produkt fixiert und somit im wesentlichen auch von einem Anbieter abhängig ist. Hier liegen die Stärken von J2EE,

bei der es sich nur um eine Spezifikation handelt, die von verschiedenen Anbietern realisiert wird, beispielsweise BEA WebLogic, IBM WebSphere oder SUN One. Weiterhin ist man auch bezüglich der Wahl der Entwicklungsumgebung nicht direkt an einen Hersteller gebunden. Es hat sich jedoch häufig gezeigt, dass es sinnvoll ist die IDE zu verwenden, dessen Laufzeitumgebung eingesetzt wird, da sonst die Entwicklung und das Time-To-Market bis ein Business-Objekt erstellt ist, erheblich länger als bei der .NET Realisierung sein kann (Langner, 2003). J2EE ist im Kern von der Programmiersprache Java abhängig, umgekehrt jedoch plattformunabhängig. Hier können getätigte Investitionen in eine Plattform weiter verwendet werden.

Für einen ausgiebigen Vergleich dieser beiden Technologien sei beispielsweise auf (Langner, 2003; van Eyle, 2001; Vawter und Roman, 2001) verwiesen. Während (van Eyle, 2001; Vawter und Roman, 2001) diese beiden Plattformen hinsichtlich der Web Services Technologie konzeptionell vergleichen, wird der Vergleich von (Langner, 2003) anhand eines konkreten Projektes, welches in beiden Plattformen realisiert wurde, durchgeführt.

Web Services werden derzeit nicht nur von verschiedenen Herstellern durch Produkte und Spezifikationen unterstützt, sondern halten auch in verschiedenen anderen Bereichen der SOA sowie der Middleware Einzug. Allen voran ist das in den frühen 90er-Jahren entstandene *Grid Computing* und die damit verbundenen *Grid Services* zu nennen. Auch die Zusammenhänge mit *Peer-To-Peer-Netzwerken* und *Peer Services* sind nicht von der Hand zu weisen (Cremers et al., 2005). Diese aktuellen Trends sollen der Vollständigkeit halber in den beiden nachfolgenden Kapitel kurz diskutiert werden.

A.2 Grid Computing und Grid Services

Um den Zusammenhang zwischen den so genannten *Grid Services* mit der Web Services Technologie herstellen zu können, muss der Begriff *Grid Computing* genauer betrachtet werden. Dieses Computing-Paradigma ist nicht neu, sondern wurde bereits in den 90er-Jahren im Zusammenhang mit der Kopplung von Hochleistungsrechnern geprägt, um hier extrem speicher- oder rechenintensive Prozesse behandeln zu können (Reinefeld und Schintke, 2004). An dieser Stelle sei erwähnt, dass es derzeit keine allgemein anerkannte Definition des Begriffs „Grid Computing“ gibt. Vielmehr wird Grid Computing durch seine spezifischen Eigenschaften charakterisiert. Ein früher Versuch wurde von Ian Foster und Carl Kesselman unternommen (Foster, 2002):

> *„A* computational grid *is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.“*

Beim Grid Computing geht es somit um die Schaffung einer Infrastruktur, die eine flexible Nutzung von verteilten Ressourcen (beispielsweise Daten oder Rechenkapazität) bereitstellt. Um das Grid Computing von anderen Ansätzen des „on-demand“ Zugriffs auf

Rechenkapazität, Daten und Diensten (Foster, 2002) zu unterscheiden, gingen Foster, Kesselman und Tuecke noch einen Schritt weiter, indem sie die Konzepte des Grids in Zusammenhang mit *virtuellen Organisationen* bringen (Foster et al., 2001):

> *„[Grid computing is concerned with] coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The sharing [is] direct access to computers, software, data, and other resources [...]. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a* virtual organization *(VO).“*

Die Autoren folgern, dass bis dahin bekannte Technologien, wie CORBA oder Enterprise JavaBeans, den Anforderungen an VOs nicht gerecht werden. Da die Grid Technologie aber auf das oben beschriebene *Sharing* fokussiert, lassen sich die erwähnten Technologien durch die Grid Technologie gut ergänzen. Das eigentliche Grid Computing lässt sich dabei durch folgende Eigenschaften genauer beschreiben (Reinefeld und Schintke, 2004): Lokale Autonomie der Grid Dienste, Heterogenität der im Grid verwendeten Ressourcen (Rechner, Plattformen, Datenspeichersysteme, etc.), Skalierbarkeit des Grids, sowie Dynamik und Anpassungsfähigkeit der Grid Software auf Ausfälle einzelner Knoten und Netzwerke innerhalb des Grids.

Aufgrund der Heterogenität stellt die Interoperabilität einen zentralen Aspekt in der Grid Technologie dar, ohne die eine VO nicht realisierbar wäre. Die Autoren Foster et al. versuchen diese in ihrer Grid Architektur durch standardisierte Protokolle zu erreichen (Foster et al., 2001):

> „Interoperability *is [...] the central issue to be addressed. In a networked environment, interoperability means common protocols. Hence, our Grid architecture is first and foremost a protocol architecture, with protocols defining the basic mechanisms by which VO users and resources negotiate, establish, manage, and exploit sharing relationships.“*

Diesen Standardisierungsbemühungen hat sich heute das *Global Grid Forum* (GGF)[4] verschieben. Das GGF ist ein Forum, dessen Teilnehmer sich aus der Industrie, der Forschung und verschiedenen Anwendungsdomänen zusammensetzen. Kernziel des GGF ist die Förderung und Unterstützung bei der Implementierung und (Weiter-)Entwicklung der Grid Technologie und ihrer Anwendung durch die Herausgabe von technischen Spezifikationen, Benutzererfahrungen und Implementierungsrichtlinien. Hierzu zählen unter anderem die *Open Grid Service Architecture* (OGSA) (Foster et al., 2004a, 2002) und die *Open Grid Service Infrastructure* (OGSI) (Tuecke et al., 2003).

[4] `www.ggf.org`

Die OGSA spezifiziert innerhalb der SOA sowohl die Dienste, die Schnittstellen dieser Dienste, Zustände der Ressourcen, die zu den Diensten gehören, als auch die Semantik/Verhalten und Interaktion dieser Dienste (Foster et al., 2004a). Zusammenfassend werden diese Dienste auch als *Grid Services* bezeichnet und folgen der Direktive (Foster et al., 2002):

> *„A* Grid Services *[is] a Web service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; the conventions address naming and upgradeability.“*

Damit wird klar, Grid Services setzen auf der Web Services Technologie auf und erweitern diese den speziellen Anforderungen des Grids entsprechend. Vor allem das beschriebene WSDL ist ein Kernaspekt dieser Grid Dienste, mit denen die Diensteschnittstellen beschrieben werden. Weiterhin soll XML allgemein als *lingua franca* für die Beschreibung und Repräsentation eingesetzt werden, während SOAP als wesentliche Transportbindung dient.

Die Schnittstellen, die den Grid Services obliegen, wurden erstmals in der OGSI beschrieben. Hierzu zählen Operationen, wie Lifetime, Factory oder HandleResolver, die von den Grid Diensten unterstützt werden müssen. D.h. die OGSI beschreibt die technischen Details und liefert eine vollständige Spezifikation des Verhaltens und der Schnittstellen, die einen Grid Service definieren (Tuecke et al., 2003):

> *„[The OGSI specifies] (1) how Grid service instances are named and referenced; (2) the base, common interfaces (and associated behaviors) that all Grid services implement; and (3) the additional (optional) interfaces and behaviors associated with factories and service groups.“*

Aufgrund einiger Defizite des WSDL 1.1, hier zu nennen ist primär das Fehlen der Möglichkeit Port-Typ Erweiterungen zu definieren, ist in der OGSI der Port-Typ isoliert und in einem eigenen Namensraum entsprechend erweitert worden (Grid WSDL). Eine zentrale Erweiterung ist die Möglichkeit zustandsbehaftete Dienste (*stateful Services*) definieren zu können. Bis dato war es nicht möglich explizit die Zustände eines Web Services auszuzeichnen. Der Benutzer musste häufig folgern, welche Zustände der Service besitzt (beispielsweise durch den Namen einer Nachricht). Um dieses Problem zu lösen, wurden Mechanismen spezifiziert, die ein Anfragen (*get*), Abändern (*set*) und eine Benachrichtigung bei Veränderung (*change notification*) eines Zustands ermöglichen. Diese so genannten *ServiceData* sind mit den Properties der JavaBeans verwandt.

Das erst kürzlich vorgeschlagene *WS Resource Framework* (WSRF)[5] (Czajkowski et al., 2004b) beschreibt im wesentlichen das Refactoring der OGSI unter Einbeziehung jüngster

[5] `www-106.ibm.com/developerworks/webservices/library/ws-resource/` und `www.globus.org/wsrf/`

Entwicklungen in der Web Service Community, wie WSDL 2.0 und WS-Addressing (Czajkowski et al., 2004a). Es definiert Ansätze, um Zustände zu modellieren, zuzugreifen und zu verwalten, Dienste zu gruppieren, sowie Fehler zu beschreiben. Genauer gesagt, das WSRF spezifiziert, wie auf zustandsbehaftete Ressourcen über Web Services zugriffen werden kann. Allgemein wird der Zustand durch eine *stateful resources* modelliert. Dieses geschieht wiederum durch die Definition eines *resource property document*, in Form eines durch XML Schema beschriebenen XML-Dokumentes. Der Begriff „*WS-Resource*“ beschreibt hierbei die Kopplung eines Web Services mit einem resource property document.

Im WSRF wird nun spezifiziert, wie derartige WS-Resources durch Standardmechanismen der Web Services deklariert, erzeugt, zugegriffen und zerstört werden können, so wie auf Veränderungen des Zustands reagiert werden kann. Der Zugriff (retrieve, change, delete) auf eine stateful resource geschieht beispielsweise über die WS-Resource Property Spezifikation. Sie definiert einen standardisierten Weg, über den ein Client die mit der WS-Resource assoziierten Daten zugreifen kann (Czajkowski et al., 2004a).

Teilweise wurde festgestellt, dass derzeitig existierende Grid Middleware bei großen Systemen nicht in der gewünschten Art und Weise skalieren (Reinefeld und Schintke, 2004). Um die Skalierbarkeit des Gesamtsystems beherrschbar machen zu können, kombinieren neuere Ansätze die Resultate des P2P Computing mit denen des Grid Computing, indem einzelne Grid Services als eigenständige P2P Netze realisiert werden. In dem nun folgenden Abschnitt wird der Zusammenhang zwischen P2P Computing und dem Gebiet der Web Services kurz betrachtet.

A.3 Peer-to-Peer Computing

Genau wie der Begriff des Grid Computing im Zusammenhang mit Web Services steht, gilt dies ebenso für das Peer-to-Peer Computing, kurz P2P Computing, denn auch P2P-Systeme adressieren SOA (Schneider, 2001). Allgemein lässt sich P2P als ein Sharing von Ressourcen und Diensten durch direkten Austausch zwischen dezentral verteilten Diensten (*Peers*) verstehen (Wojciechowski und Weinhardt, 2002). Die Ressourcen können unter anderem aus Speicher- oder Rechenkapazität, Inhalten oder der Präsenz eines Anwenders bestehen, die sich an den Kanten des Internets befinden (Shirky, 2000). Leider hat sich noch keine allgemein anerkannte Definition durchgesetzt. Eine mögliche Beschreibung einer P2P-Architektur ist die von Navaneeth Krishnan (2001) im Zusammenhang mit der JXTA-Initative[6]:

> *„The* P2P architecture *is a decentralized architecture [...], where neither clients nor server status exists in a network. Every entity in the network, referred to as a* peer*, has equal status, meaning that an entity can either request a service (a client trait) or provide a service (a server trait).“*

[6] `www.jxta.org`

P2P ermöglicht es jedem Endgerät mit anderen Rechnern eines Netzwerkes in Verbindung zu treten und zu interagieren. Dabei wird im Allgemeinen von keinem zentralen Knoten ausgegangen, sondern jedes Endgerät fungiert als Client und als Server. Hierdurch wird ein symmetrisches und dezentrales Kommunikationsmodell für Applikationen, Dienste und Benutzer erreicht. Peers in einem P2P-Netzwerk haben gewöhnlich ein Bewusstsein (*Awareness*) für andere Peers. Somit wird kein zentrales Dienstrepository, in dem sich Dienste registrieren, benötigt. Weiterhin können Peers ein virtuelles Netzwerk aufspannen, ohne die Komplexität der Peerverbindungen preiszugeben.

Die P2P-Technologie ermöglicht die Collaboration von Peers zu Arbeitsgruppen oder Peer-Gruppen, unabhängig von der Lokalisierung der Gruppenmitglieder. Hierbei ist die Selbstorganisation von P2P-Netzwerken essentiell, da gewöhnlich nicht alle Mitglieder zu jeder Zeit an der Collaboration teilnehmen. Häufig werden die Peers durch PCs repräsentiert, die nur temporär an das Internet angeschlossen bzw. aufgrund von Ortstransparenz nicht immer zur gleichen Zeit verfügbar sind. Somit ist das P2P-Netzwerk starken Veränderungen ausgesetzt, was seine extrem dynamische Natur ausmacht (Alda und Cremers, 2003).

Im Gegensatz zu den Web Services besitzen P2P-Systeme kein zentrales Dienstrepository wie beispielsweise UDDI. Die Suche nach geeigneten Peers innerhalb eines P2P-Systeme wird häufig durch verteilte Suchalgorithmen realisiert. Im JXTA Framework erfolgt die Veröffentlichung von Peers beispielsweise durch so genannte *Rendezvous Peers*, die dezentral Informationen über andere Peers zwischenspeichern. Ein Rendezvous Peers kann somit Peers helfen andere Peers im Netz aufzufinden. Sie können ferner Discovery-Anfragen an zusätzliche Rendezvous Peers weiterleiten (Krishnan, 2001).

Ein zentraler Unterschied zwischen den Paradigmen der Web Services und der P2P-Netzwerke zeigt sich in der Verfügbarkeit der Inhalte und Dienste. Je stärker ein Web Service beansprucht wird, desto schlechter ist seine Verfügbarkeit. Bei P2P-Netzwerken wird die Popularität der Inhalte durch die Anzahl der Peers, die diesen Inhalt zur Verfügung stellen, ausgedrückt. Steigt die Nachfrage nach einem Inhalt, steigt auch die Redundanz und somit auch die Verfügbarkeit (Wojciechowski und Weinhardt, 2002).

Beide Technologien können neben ihren Unterschieden voneinander profitieren und geschickte Synergien eingehen. Beispielsweise lösen die Web Services das Interoperabilitätsproblem auf technischer Ebene, da sie standardisierte Protokolle anbieten, die von allen führenden Unternehmen unterstützt werden. Hierzu nennen sind vornehmlich SOAP und WSDL. Interoperabilität in P2P-Netzen ist derweil ein ungelöstes Problem, welches sich das JXTA-Projekt von Sun verschrieben hat. Jedoch bleibt abzuwarten, inwieweit andere Herstellen sich dieser „Standardisierung“ anschließen. JXTA besteht im wesentlichen aus offenen und generalisierten P2P-Protokollen, die es jedem Endgerät erlauben an einem P2P-Netzwerk teilzunehmen und als Peers untereinander zu kommunizieren und zusammenzuarbeiten. Die Kernprotokolle des JXTA-Frameworks umfassen Möglichkeiten für das Peer-Discovery, die Bildung von Peer-Gruppen, Peer-Kommunikation und -Monitoring, sowie Security. Auf dieser Basis lassen sich eigene P2P-Dienste und Applikationen realisieren.

Um von JXTA-Peers aus mit Web Services interagieren zu können, versucht das JXTA-Team zur Zeit, SOAP und WSDL in JXTA zu integrieren. Ist dieser Schritt vollzogen, könnten auf lange Sicht hinaus auch Peers über SOAP-ähnliche Protokolle kommunizieren (vgl. Grid Services in Abschnitt A.2).

P2P kann die Nutzer sowie deren Nutzereigenschaften in das Gebiet der Web Services tragen, die in Web Services nicht explizit betrachtet werden. Beispielsweise können Benutzerpräferenzen dazu herangezogen werden, bestimmte Dienste auszuwählen, die der Benutzer bevorzugt bzw. abzustimmen, ob ein Benutzer bereit ist, für einen Dienst zu bezahlen (Schneider, 2001).

Ein weiteres Anwendungsfeld, in dem P2P-Konzepte die Web Services Technologie erweitern können, ist das Prozessmanagement. Gewöhnlich werden Web Services Kompositionen, d.h. Geschäftsprozesse, die mittels Web Services realisiert werden, durch eine zentrale Komponente verwaltet und ausgeführt. Dieses Vorgehen limitiert jedoch die Anzahl bzw. Performanz der parallel ausführbaren Prozesse, da gewöhnlich die Verwaltungskomponente alle Zustände des Prozesses sichern muss, um eventuell Rollbacks oder Recoveries zu garantieren. Durch eine Verteilung des Prozessmanagements auf eine P2P-Architektur können diese Einschränkungen geschickt überwunden werden. Ein derartiger Ansatz wird beispielsweise im OSIRIS System verfolgt (Schuler et al., 2004).

Wir haben gesehen, dass die Idee der Serviceorientierung nicht nur durch die Web Services Architektur Ausdruck findet, sondern das dieses Konzept sich ebenfalls durch Softwareparadigmen wie Grid Computing und P2P Computing realisieren lässt. Dabei sind diese drei Technologien nicht nur hinsichtlich ihrer Anwendung durch Unterschiede charakterisiert, sondern zeigen auch starke Ähnlichkeiten bzw. Potential für geschickte Synergien. Der für diese Arbeit zentrale Aspekt der Heterogenität autonomer Dienste wohnt allen vorgestellten SOA-Realisierungen inne.

Anhang B

Schemadefinitionen und WSDL-Beschreibungen

B.1 Spezifikation der MPL-Kontrollstrukturen

Die Semantik der Elemente eines Kontrollflusses, wie Sequenz, Fallunterscheidung und parallele Ausführung, wird im Folgenden anhand konkreter Petri-Netze spezifiziert. Beispielsweise ist bis dato die Semantik der verschiedenen Synchronisationsaktivitäten nicht exakt formuliert. Vorher wird eine kurze Einführung in Petri-Netze gegeben. Weiterführende Aspekte finden sich in der zahlreichen Literatur, wie (Murata, 1989; Baumgarten, 1996).

B.1.1 Einführung in Petri-Netze

Petri-Netze gehen auf die Arbeiten von Carl Adam Petri (1962) zurück. Seit dieser Zeit sind viele Artikel, Bücher und Forschungsarbeiten zu diesem Thema erschienen. Einen grundlegenden Überblick über dieses Themengebiet liefern beispielsweise (Murata, 1989; Baumgarten, 1996). Im Folgenden werden einige wesentliche Definitionen eingeführt. In der Literatur wird der Begriff des *Petri-Netzes* häufig für verschiedene Netzstrukturen wie S/T-Netze oder höhere Netze synonym verwendet (Baumgarten, 1996).

Definition B.1 (Netz)**.** Ein *Netz* ist ein Tripel $N = (P, T, F)$, wobei

P ist eine endliche Menge von *Stellen* (*places*),
T ist eine endliche Menge von *Transitionen*,
$P \cap T = \emptyset$ und $P \cup T \neq \emptyset$,
$F \subseteq (P \times T) \cup (T \times P)$ ist eine Menge von Kanten (*flow relation*). ■

Definition B.2 (Petri-Netz, Petri-Netz Struktur)**.** Ein *Petri-Netz* (oder *Stellen-Transitions-System*) ist ein 5-Tupel $PN = (P, T, F, W, M_0)$, wobei

(P, T, F) ist ein Netz,
$W : F \to \mathbb{N}$ ist eine *Kantengewichtsfunktion*,
$M_0 : P \to \mathbb{N}_0$ ist eine *Anfangsmarkierung*.

Eine *Petri-Netz Struktur* $N = (P, T, F, W)$ ohne initiale Markierung wird durch N gekennzeichnet. Ein Petri-Netz mit Anfrangsmarkierung wird dann mit (N, M_0) gekennzeichnet. Die Abbildung $M : P \to \mathbb{N}_0$ (*Markierung*) weißt jeder Stelle $p \in P$ eine nicht negative Zahl zu, die so genannte Markenzahl $M(p)$. ■

In obiger Definition können die Stellen beliebig viele Markierungen (*tokens*) aufnehmen. Daher spricht man auch von einem Netz mit unbeschränkter Kapazität (*infinite capacity net*). Möchte man die Kapazitäten der Stellen auf eine maximale Anzahl von Markierungen begrenzen, überführt man das Netz in ein Netz mit beschränkter Kapazität (*finite capacity net*). Für ein derartiges Netz (N, M_0) wird jeder Stelle $p \in P$ eine *Kapazitätsfunktion* $K : P \to \mathbb{N} \cup \{\infty\}$ zugeordnet, die die maximale Kapazität der Stelle p begrenzt. Man beachte, auch hier sind Stellen mit unendlicher Kapazität möglich. In einem Netz mit beschränkter Kapazität muß für jede Markierung M gelten: $\forall p \in P : M(p) \leqslant K(p)$.

Anschaulich gesprochen ist das einem Petri-Netz zugeordnete Netz ein gerichteter, gewichteter, bipartiter Graph, der zwei Arten von Knoten besitzt: Stellen (statische Bestandteile) und Transitionen (dynamische Bestandteile), wobei Kanten sowohl von Stellen zu Transitionen als auch von Transitionen zu Stellen existieren. In der graphischen Darstellung werden die Stellen durch Kreise und die Transitionen durch Rechtecke ausgezeichnet.

Ausgehend von einem Knoten $x \in P \cup T$ definiert man folgende Mengen von Nachbarknoten: Die Menge der *Eingangs-* bzw. *Inputknoten*

$$\bullet x := \{y | (y, x) \in F\},$$

und die Menge der *Ausgangs-* bzw. *Outputknoten*

$$x\bullet := \{y | (x, y) \in F\}.$$

Die Mengen $\bullet x$ und $x\bullet$ werden auch *Vor-* bzw. *Nachbereich* von x genannt. Diese Notation läßt sich leicht auf Knotenmengen ausdehnen: sei $X \subseteq P \cup T$, dann ist $\bullet X := \bigcup_{x \in X} \bullet x$ und $X\bullet := \bigcup_{x \in X} x\bullet$.

Ein Paar $p \in P$ und $t \in T$ wird *Schlinge* (*self-loop*) genannt, wenn p sowohl Input- als auch Outputknoten von t ist, d.h. $p \in \bullet t \cap t\bullet$. Ein Petri-Netz heißt *schlingenfrei* (*pure*), falls es keine Schlingen enthält.

B.1.2 Dynamik von Petri-Netzen

Die Dynamik eines Petri-Netzes wird durch das Aktivieren oder Feuern (*firing*) einer Transition simuliert. Dabei werden Marken bei einem Vorbereich verbraucht bzw. bei einem Nachbereich erzeugt.

Definition B.3 (Aktivierung)**.** Eine Transition $t \in T$ heißt *aktiviert* (*enabled*) bzgl. einer Markierung M, in Zeichen $M[t\rangle$, wenn

$$\forall p \in \bullet t : M(p) \geqslant W(p,t).$$

Für Netze mit beschränkter Kapazität wird weiter gefordert, dass

$$\forall p \in t\bullet : M(p) \leqslant K(p) - W(t,p).$$

■

Eine Transition t *schaltet* von M nach M', in Zeichen $M[t\rangle M'$, wenn t unter M aktiviert wird und M' aus M durch Entnahme bzw. Ablage von Marken entsteht. Dabei gilt:

$$M'(p) = \begin{cases} M(p) - W(p,t), & \text{falls } p \in \bullet t \backslash t\bullet, \\ M(p) + W(t,p), & \text{falls } p \in t\bullet \setminus \bullet t, \\ M(p) - W(p,t) + W(t,p), & \text{falls } p \in t\bullet \cap \bullet t \text{ (Schlinge)}, \\ M(p) & \text{sonst.} \end{cases}$$

M' heißt dann *(unmittelbare) Folgemarkierung von M unter t* und wird auch als Mt geschrieben (Baumgarten, 1996).

Beispiel 2.1. Das in Abbildung B.1 illustrierte Beispiel zeigt die chemische Reaktion $2H_2 + O_2 \rightarrow 2H_2O$. Die Transition t konsumiert die zwei Tokens der Stelle H_2 und ein Token der Stelle O_2 (Abb. B.1 (links)). Nach dem Feuern werden zwei Tokens in H_2O erzeugt (Abb. B.1 (rechts)). Anschließend ist die Transaktion t nicht länger aktiviert (Murata, 1989).

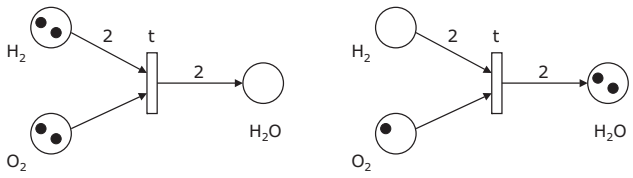

Abbildung B.1: Das Beispiel illustriert den chemischen Prozess $2H_2 + O_2 \rightarrow 2H_20$.

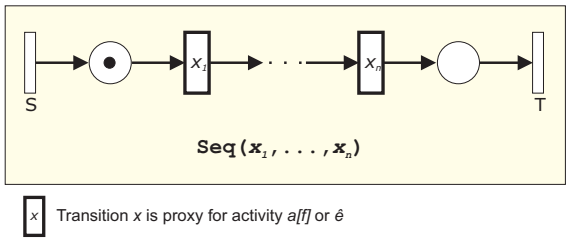

Abbildung B.2: Petri-Netz der Sequenz (`Seq`).

Definition B.4 (Erreichbarkeit)**.** Sei (N, M_0) ein Petri-Netz und $\sigma = M_0 t_1 M_1 t_2 ... t_n M_n$ oder einfach $\sigma = t_1 t_2 ... t_n$ eine Folge mit $t_i \in T$, dann wird die Markierung M_n eine *Folgemarkierung* von M_0 unter σ, in Zeichen $M_0[\sigma\rangle M_n$, genannt. σ bezeichnet man auch als eine *Schaltfolge* (*firing* oder *occurrence sequence*) und man nennt σ *aktiviert* unter M_0, in Zeichen $M_0[\sigma\rangle$. Eine Transition t heißt *aktivierbar*, wenn eine Schaltfolge σ mit $M_0[\sigma t\rangle$ existiert.

Eine Markierung M_n wird als *erreichbare Markierung* bezeichnet, wenn es eine Schaltfolge gibt, die M_0 in M_n transformiert. Die Menge aller erreichbaren Markierungen von M_0 ist gekennzeichnet durch

$$[M_0\rangle := \{M_0\sigma | \sigma \text{ Schaltfolge}\}.$$

■

Mit diesen Grundbegriffen lassen sich die Kontrollstrukturen der Mediatorkomposition über Petri-Nezte beschreiben. Sei im Folgenden $G_c = (A_c, E_c, V_c)$ ein Kontrollflussgraph.

B.1.3 Spezifikation der Sequenz

Die hintereinander Ausführung von Aktivitäten $a[f]$ und $\hat{e}$ wird durch einen Pfad in E_c beschrieben, der nur aus einfachen Aktivierungsübergängen besteht (vgl. Definition 4.5). Dieser Pfad wird in einen einfachen Pfad eines Petri-Netzes überführt. Abbildung B.2 beschreibt das dazugehörige Petri-Netz der Sequenz (`Seq`). Durch den Startpunkt (S) wird genau ein Token auf der nachfolgenden Stelle (S•) erzeugt. Die inneren Transitionen x_i, $1 \leqslant i \leqslant n$, n beliebig, des Petri-Netzes entsprechenden Aktivitäten der Form $a[f]$ und $\hat{e}$. Aufgrund der Azyklischkeit des Kontrollflussgraphen ist auch das Petri-Netz azyklisch. Daher terminiert as Petri-Netz der Sequenz, wenn die Transitionen der einzelnen Aktivitäten terminieren.

B.1.4 Spezifikation der Fallunterscheidung

Fallunterscheidungen entstehen durch bedingte Aktivierungsübergänge innerhalb eines Kontrollflussgraphen. Sie laufen nur dann parallel ab, wenn mehrere Bedingungen einzelner Ak-

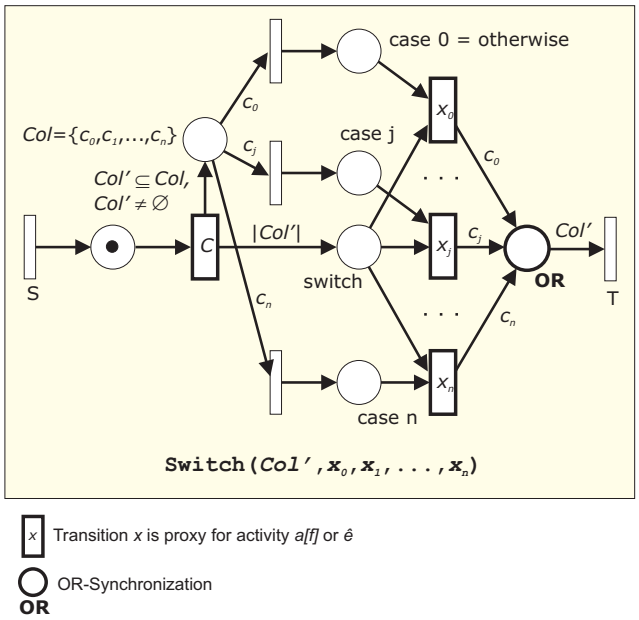

Abbildung B.3: Petri-Netz der Fallunterscheidung (`Switch`).

tivierungsübergänge gleichzeitig erfüllt sind. Es existiert immer eine *else*- oder *otherwise*-Kante, so dass mindestens ein bedingter Akrivierungsübergang ausgeführt wird. Synchronisiert werden die verschiedenen möglichen Pfade durch OR-Synchronisationsaktivitäten, die in der Sprache der Petri-Netze Stellen entsprechen. Um auf einfache Weise die verschiedenen Bedingungen in einem Petri-Netz ausdrücken zu können, werden *farbige Petri-Netze* (*Colored Petri Nets*) zur Spezifikation eingesetzt. Prinzipiell ließe sich die Fallunterscheidung auch auf Standard-Petri-Netze übertragen, dies wäre jedoch sehr unübersichtlich. Zusammenfassend ergibt sich das Petri-Netz der Fallunterscheidung (`Switch`), wie in Abbildung B.3 dargestellt.

Das Petri-Netz der Fallunterscheidung besitzt i, $1 \leqslant i \leqslant n$ konkrete Fälle sowie für $i = 0$ einen *otherwise*-Fall. Es wird gefordert, dass mindestens einer dieser $n + 1$ Fälle durch ein Token aktiviert wird und beliebig viele Fälle aktiviert werden können. Diese Semantik entspricht im wesentlichen der des Java *switch*-Blocks ohne den Einsatz von *break*-Anweisungen.

Im Petri-Netz der Fallunterscheidung werden diese Anforderungen dadurch erreicht, dass die Transition C eine Menge von farbigen Tokens $Col' \subseteq Col = \{c_0, c_1, \ldots, c_n\}$, $Col' \neq \emptyset$ erzeugt und jede Eingangstransition eines Falles genau auf eine Farbe reagiert. Die OR-Stelle nach den Transitionen für die verschiedenen Fälle x_i, $0 \leqslant i \leqslant n$ synchronisiert die komplette Fallunterscheidung. Sie wartet bis alle möglichen Fälle feuern und leitet anschließend die Aktivierung weiter. Dies geschieht ebenfalls über die verschieden Farben. Gewöhnlich erwartet die C-Transition eine spezielle Mediatorfunktionalität, und zwar einen Kompara-

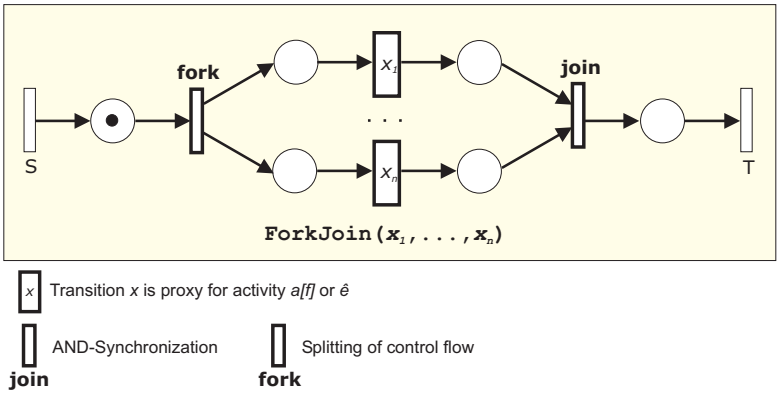

Abbildung B.4: Petri-Netz der parallelen Ausführung (`ForkJoin`).

tor, der die Evaluierung der verschiedenen Fälle vornimmt. Soll genau ein Fall ausgewählt werden, dann ist $Col' = \{c_i\}$, für $0 \leqslant i \leqslant n$. Das Petri-Netz der Fallunterscheidung macht keine Aussagen über die Reihenfolge der Ausführung, wenn mehrere Transition x_i aktiviert werden. Diese können parallel oder in einer beliebigen Reihenfolge abgearbeitet werden. Deswegen müssen die Transitionen voneinander unabhängig sein.

Unter der Forderung, dass C terminiert und mindestens einen Fall auswählt, d.h. C erzeugt $Col' \subseteq Col$, $Col' \neq \emptyset$ spezielle Farben auf der Ausgangsstelle Col und die damit ausgewählten Transitionen x_i, $0 \leqslant i \leqslant n$ terminieren, terminiert auch das Petri-Netz der Fallunterscheidung und die Transition T wird *genau* einmal erreicht.

B.1.5 Spezifikation der parallelen Ausführung

Gehen in einem Kontrollflussgraphen G_c von einer Aktivität mehrere unbedingte Aktivierungsübergänge aus, beginnt von dort eine parallele Ausführung, die in E_c auch wieder synchronisiert werden muss. Dies geschieht über AND-Synchronisationsaktivitäten. Die parallele Ausführung wird in das entsprechende `ForkJoin` Petri-Netz übersetzt. Abbildung B.4 zeigt dieses Netz. Parallele Ausführungen dienen im wesentlichen der Performanzsteigerung, wenn innerhalb einer Komposition Mediatorfunktionalitäten unabhängig voneinander ausgeführt werden können.

Die parallele Ausführung beginnt mit der Aufspaltung des Kontrollflusses an der fork-Transition und wird immer durch die join-Transition synchronisiert, die der AND-Synchronisationsaktivität entspricht. Die join-Transition wird erst dann aktiviert, wenn alle Transitionen x_i, $1 \leqslant i \leqslant n$ beendet wurden. Anschließend erzeugt sie ein Token an ihrer Ausgangsstelle (join•) und aktiviert damit die Transition T. Das Petri-Netz der parallelen Ausführung terminiert, wenn *alle* Transitionen x_i, $1 \leqslant i \leqslant n$ terminieren.

B.1.6 Kombination der Petri-Netz-Module

Die oben aufgeführten Petri-Netze beschreiben Module (`Seq`, `Switch`, `ForkJoin`), die flexibel ineinander gesteckt werden können, um komplexere Systeme zu beschreiben. Diese Einbettungsfähigkeit ist wie folgt definiert: Zwei Petri-Netze der vorgestellten Module können ineinander eingebettet werden, indem eine Transition x_i des einen Petri-Netz-Moduls durch das komplette andere Petri-Netz-Modul ersetzt wird. Anschaulich entstehen dabei „geschachtelte“ Petri-Netz-Module. Die so entstehenden Petri-Netze terminieren, wenn die einzelnen Unter-Petri-Netze terminieren. Damit eine beliebige Ausführungseinheit (*execution engine*) einen Kontrollflussgraphen G_c korrekt ausführen kann, muss es mindestens eine mögliche Abbildung geben, mit der G_c mittels der Kombination der Petri-Netz-Module in ein entsprechendes Petri-Netz überführt werden kann.

B.2 Schemadefinition der Mediatorprofilsprache

```
<!-- main concept of MPL: mediator profile;
     containing all relevant information of
     one mediator -->
<owl:Class rdf:ID='MediatorProfile'>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:ObjectProperty
     rdf:about='#developerInformation'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:ObjectProperty
     rdf:about='#mediatorCategory'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:DatatypeProperty
     rdf:about='#mediatorName'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:DatatypeProperty
     rdf:about='#capabilitiesDescription'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:DatatypeProperty
     rdf:about='#wsdlLocation'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction owl:minCardinality='1'>
   <owl:onProperty>
    <owl:ObjectProperty
     rdf:about='#mediatorFunctionality'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<!-- capabilities -->
<owl:ObjectProperty
 rdf:ID='mediatorFunctionality'>
 <rdfs:domain
  rdf:resource='#MediatorProfile'/>
 <rdfs:range
  rdf:resource='#MediatorFunctionality'/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty
 rdf:ID='mediatorProperty'>
 <rdfs:domain
  rdf:resource='#MediatorProfile'/>
 <rdfs:range
  rdf:resource='#MediatorProperty'/>
</owl:ObjectProperty>

<owl:ObjectProperty
 rdf:ID='developerInformation'>
 <owl:samePropertyAs rdf:resource=
  '&owl-s;Profile.owl#contactInformation'/>
 <rdfs:range
  rdf:resource='&owl-s;Profile.owl#Actor'/>
 <rdfs:domain
  rdf:resource='#MediatorProfile'/>
</owl:ObjectProperty>

<owl:ObjectProperty
 rdf:ID='mediatorCategory'>
 <owl:samePropertyAs rdf:resource=
  '&owl-s;Profile.owl#serviceCategory'/>
 <rdfs:domain
  rdf:resource='#MediatorProfile'/>
</owl:ObjectProperty>

<owl:DatatypeProperty
 rdf:ID='mediatorName'
 rdf:type='&owl+oil;#UniqueProperty'>
 <rdfs:range rdf:resource='&xsd;string'/>
 <rdfs:domain
  rdf:resource='#MediatorProfile'/>
</owl:DatatypeProperty>

<owl:DatatypeProperty
 rdf:ID='wsdlLocation'>
 <rdfs:domain
  rdf:resource='#MediatorProfile'/>
 <rdfs:range rdf:resource='&xsd;anyURI'/>
</owl:DatatypeProperty>

<owl:DatatypeProperty
 rdf:ID='capabilitiesDescription'>
 <rdfs:subPropertyOf>
  <owl:DatatypeProperty
   rdf:about='#textDescription'/>
 </rdfs:subPropertyOf>
 <rdfs:domain
  rdf:resource='#MediatorProfile'/>
</owl:DatatypeProperty>

<owl:DatatypeProperty
 rdf:ID='textDescription'>
 <rdfs:comment>Human readable text
  description.</rdfs:comment>
 <rdfs:range rdf:resource='&xsd;string'/>
</owl:DatatypeProperty>

<owl:DatatypeProperty
 rdf:ID="generalGradation">
 <rdfs:range>
  <owl:DataRange rdf:ID='Gradation'>
   <owl:oneOf>
    <rdf:List><rdf:first
rdf:datatype="&xsd;integer">0</rdf:first>
     <rdf:rest>
      <rdf:List>
       ...
   </owl:oneOf>
  </owl:DataRange>
 </rdfs:range>
</owl:DatatypeProperty>
```

Abbildung B.5: Mediatorprofil in MPL.

```
<!-- Concept, Port, Parameter
     and Variable -->
<owl:Class rdf:ID='Concept' />

<owl:Class rdf:ID='Parameter'>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:DatatypeProperty
     rdf:about='#parameterDescription'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:DatatypeProperty
     rdf:about='#rangeOf'/>
```

```
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:DatatypeProperty
     rdf:about='#parameterName'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID='Port'>
 <rdfs:subClassOf
  rdf:resource='#Parameter'/>
</owl:Class>

<owl:Class rdf:ID='Variable'>
 <rdfs:subClassOf
  rdf:resource='#Parameter'/>
</owl:Class>

<owl:Class rdf:ID='BoolVariable'>
 <rdfs:subClassOf rdf:resource='#Variable'>
  <owl:Restriction>
  <owl:onProperty rdf:resource="#rangeOf"/>
   <owl:hasValue
    rdf:resource="&xsd;boolean"/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<!-- Relationships -->
<rdf:Property rdf:ID='conceptRef'>
 <rdfs:domain rdf:resource='#Concept'/>
<rdfs:range rdf:resource='&rdfs;Resource'/>
</rdf:Property>

<owl:ObjectProperty rdf:ID='hasConcept'>
 <rdfs:domain rdf:resource='#Port'/>
 <rdfs:range rdf:resource='#Concept'/>
</owl:ObjectProperty>

<owl:DatatypeProperty
 rdf:ID='parameterName'>
 <rdfs:domain rdf:resource='#Parameter'/>
 <rdfs:range rdf:resource='&xsd;string'/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID='rangeOf'>
 <rdfs:comment>The range of the parameter
  (syntax); XSD datatypes.</rdfs:comment>
 <rdfs:domain rdf:resource='#Parameter'/>
 <rdfs:range rdf:resource='&xsd;anyURI'/>
</owl:DatatypeProperty>

<owl:DatatypeProperty
 rdf:ID='parameterDescription'>
 <rdfs:subPropertyOf>
  <owl:DatatypeProperty
   rdf:about='#textDescription'/>
 </rdfs:subPropertyOf>
 <rdfs:domain rdf:resource='#Parameter'/>
</owl:DatatypeProperty>

<owl:DatatypeProperty
 rdf:ID='conceptGradation'>
 <rdfs:subPropertyOf
  rdf:resource='#generalGradation'/>
 <rdfs:domain rdf:resource='#Concept'/>
</owl:DatatypeProperty>
```

Abbildung B.6: Parameter- und Portdefinition in MPL.

```
<owl:Class rdf:ID='MediatorProperty'>
 <rdfs:subClassOf rdf:resource='#Port'/>
 <rdfs:subClassOf>
  <owl:Restriction owl:minCardinality='1'>
   <owl:onProperty>
    <owl:ObjectProperty
     rdf:about='#propertyEffect'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#belongs
    ToComplexFunctionality"/>
   <owl:minCardinality rdf:datatype="&xsd;
nonNegativeInteger">0</owl:minCardinality>
    <owl:maxCardinality rdf:datatype="&xsd;
nonNegativeInteger">1</owl:maxCardinality>
```

```
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID='PropertyEffect'>
 <rdfs:subClassOf>
  <rdfs:Class rdf:about='#Effect'/>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:ObjectProperty
     rdf:about='#effectOn'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:ObjectProperty
     rdf:about='#effectOf'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID='effectOn'>
 <rdfs:range
  rdf:resource='#MediatorFunctionality'/>
 <rdfs:domain
  rdf:resource='#PropertyEffect'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='effectOf'>
 <rdfs:range
  rdf:resource='#MediatorProperty'/>
 <rdfs:domain
  rdf:resource='#PropertyEffect'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='effectedBy'>
 <rdfs:range
  rdf:resource='#PropertyEffect'/>
 <rdfs:domain
  rdf:resource='#MediatorFunctionality'/>
</owl:ObjectProperty>

<owl:ObjectProperty
 rdf:ID='propertyEffect'>
 <rdfs:domain
  rdf:resource='#MediatorProperty'/>
 <rdfs:range
  rdf:resource='#PropertyEffect'/>
</owl:ObjectProperty>
```

Abbildung B.7: Mediatorproperty in MPL.

```
<owl:Class rdf:ID='MediatorFunctionality'>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:DatatypeProperty
     rdf:about='#functionDescription'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:DatatypeProperty
     rdf:about='#functionName'/>
   </owl:onProperty>
    </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
 rdf:resource="#isComplexFunctionality"/>
   <owl:minCardinality rdf:datatype="&xsd;
 nonNegativeInteger">0</owl:minCardinality>
   <owl:maxCardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:maxCardinality>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class
 rdf:ID="ComparatorFunctionality">
 <rdfs:subClassOf
  rdf:resource='#MediatorFunctionality'/>
</owl:Class>

<owl:Class rdf:ID='MediatorClass' />
```

```
<owl:Class rdf:ID='Effect'>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:DatatypeProperty
     rdf:about='#effectDescription'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction owl:cardinality='1'>
   <owl:onProperty>
    <owl:ObjectProperty
     rdf:about='#effectCategory'/>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty
 rdf:ID='functionName'>
 <rdfs:range rdf:resource='&xsd;string'/>
 <rdfs:domain
  rdf:resource='#MediatorFunctionality'/>
</owl:DatatypeProperty>

<owl:DatatypeProperty
 rdf:ID='functionDescription'>
   <rdfs:subPropertyOf>
  <owl:DatatypeProperty
   rdf:about='#textDescription'/>
 </rdfs:subPropertyOf>
 <rdfs:domain
  rdf:resource='#MediatorFunctionality'/>
</owl:DatatypeProperty>

<owl:ObjectProperty
 rdf:ID='functionCategory'>
 <rdfs:domain
  rdf:resource='#MediatorFunctionality'/>
 <rdfs:range rdf:resource='&owl-s;
  Profile.owl#Category'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='hasMClass'>
 <rdfs:domain
  rdf:resource='#MediatorFunctionality'/>
 <rdfs:range
  rdf:resource='#MediatorClass'/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID='inputPort'>
 <rdfs:domain
  rdf:resource='#MediatorFunctionality'/>
 <rdfs:range rdf:resource='#Port'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='outputPort'>
 <rdfs:domain
  rdf:resource='#MediatorFunctionality'/>
 <rdfs:range rdf:resource='#Port'/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="mClassRef">
 <rdfs:domain
  rdf:resource='#MediatorClass'/>
 <rdfs:range>
  <owl:DataRange rdf:ID='Classification'>
   <owl:oneOf><rdf:List>
     <rdf:first rdf:datatype=
      "&xsd;string">filter</rdf:first>
      <rdf:rest><rdf:List><rdf:first
       rdf:datatype="&xsd;string">
       transformer</rdf:first>
      <rdf:rest><rdf:List><rdf:first
       rdf:datatype="&xsd;string">
       sorter</rdf:first>
      <rdf:rest><rdf:List><rdf:first
       rdf:datatype="&xsd;string">
       parser</rdf:first>
      <rdf:rest><rdf:List><rdf:first
       rdf:datatype="&xsd;string">
       dereferencer</rdf:first>
      <rdf:rest><rdf:List><rdf:first
       rdf:datatype="&xsd;string">
       mapper</rdf:first>
      <rdf:rest><rdf:List><rdf:first
       rdf:datatype="&xsd;string">
       converter</rdf:first>
      <rdf:rest rdf:resource="&rdf;nil" />
                ...
  </owl:oneOf></owl:DataRange></rdfs:range>
</owl:DatatypeProperty>

<owl:DatatypeProperty
 rdf:ID='mClassGradation'>
 <rdfs:subPropertyOf
  rdf:resource='#generalGradation'/>
 <rdfs:domain
  rdf:resource='#MediatorClass'/>
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty
 rdf:ID='effectDescription'>
 <rdfs:subPropertyOf>
  <owl:DatatypeProperty
   rdf:about='#textDescription'/>
 </rdfs:subPropertyOf>
 <rdfs:domain rdf:resource='#Effect'/>
</owl:DatatypeProperty>
```

```
<owl:ObjectProperty
 rdf:ID='effectCategory'>
 <rdfs:domain rdf:resource='#Effect'/>
 <rdfs:range rdf:resource='&owl-s;
  Profile.owl#Category'/>
</owl:ObjectProperty>
```

Abbildung B.8: Mediatorfunktionalität in MPL.

```
<owl:Class rdf:ID="Activity" />

<owl:Class rdf:ID="DataActivity">
<rdfs:subClassOf rdf:resource='#Activity'/>
</owl:Class>

<owl:Class rdf:ID="FunctionalityActivity">
 <rdfs:subClassOf
  rdf:resource='#DataActivity'/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#functionRef"/>
   <owl:cardinality rdf:datatype="&xsd;
   nonNegativeInteger">1</owl:cardinality>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="ThisActivity">
 <rdfs:subClassOf
  rdf:resource='#DataActivity'/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#thisRef"/>
   <owl:cardinality rdf:datatype="&xsd;
    nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="ComparatorActivity">
 <rdfs:subClassOf
  rdf:resource='#DataActivity'/>
 <rdfs:subClassOf>
  <owl:Restriction>
```

```
   <owl:onProperty
    rdf:resource="#comparatorRef"/>
    <owl:minCardinality rdf:datatype="&xsd;
 nonNegativeInteger">0</owl:minCardinality>
    <owl:maxCardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:maxCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#xquery"/>
   <owl:minCardinality rdf:datatype="&xsd;
 nonNegativeInteger">0</owl:minCardinality>
   <owl:maxCardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:maxCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="JoinActivity">
 <rdfs:subClassOf
       rdf:resource='#Activity'/>
</owl:Class>

<owl:Class rdf:ID="ANDActivity">
 <rdfs:subClassOf
       rdf:resource='#JoinActivity'/>
</owl:Class>

<owl:Class rdf:ID="ORActivity">
 <rdfs:subClassOf
       rdf:resource='#JoinActivity'/>
</owl:Class>

<owl:Class rdf:ID="EmptyActivity">
<rdfs:subClassOf rdf:resource='#Activity'/>
</owl:Class>
```

```
<owl:ObjectProperty rdf:ID='thisRef'>
<rdfs:domain rdf:resource='#ThisActivity'/>
 <rdfs:range
  rdf:resource='#ComplexFunctionality'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='functionRef'>
 <rdfs:domain
  rdf:resource='#FunctionalityActivity'/>
 <rdfs:range
  rdf:resource='#MediatorFunctionality'/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID='comparatorRef'>
 <rdfs:domain
  rdf:resource='#ComparatorActivity'/>
 <rdfs:range
  rdf:resource='#ComparatorFunctionality'/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID='xquery'>
 <rdfs:domain
  rdf:resource='#ComparatorActivity'/>
 <rdfs:rangerdf:resource='&xsd;string'/>
</owl:DatatypeProperty>
```

Abbildung B.9: Aktivitäten komplexer Funktionalitäten in MPL.

```
<owl:Class rdf:ID="ComplexFunctionality">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#hasInterface"/>
   <owl:cardinality rdf:datatype="&xsd;
   nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID='hasInterface'>
 <rdfs:domain rdf:resource=
  '#ComplexFunctionality'/>
 <rdfs:range rdf:resource=
  '#MediatorFunctionality'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='hasProperty'>
 <rdfs:domain rdf:resource=
  '#ComplexFunctionality'/>
 <rdfs:range rdf:resource=
  '#MediatorProperty'/>
</owl:ObjectProperty>

<owl:ObjectProperty
 rdf:ID='belongsToComplexFunctionality'>
 <rdfs:domain
  rdf:resource='#MediatorProperty'/>
 <rdfs:range
  rdf:resource='#ComplexFunctionality'/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty
 rdf:ID='isComplexFunctionality'>
 <rdfs:domain
  rdf:resource='#MediatorFunctionality'/>
 <rdfs:range
  rdf:resource='#ComplexFunctionality'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='activity'>
 <rdfs:domain
  rdf:resource='#ComplexFunctionality'/>
 <rdfs:range rdf:resource='#Activity'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='dataLink'>
 <rdfs:domain
  rdf:resource='#ComplexFunctionality'/>
 <rdfs:range rdf:resource='#DataLink'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='contolLink'>
 <rdfs:domain
  rdf:resource='#ComplexFunctionality'/>
 <rdfs:range rdf:resource='#ControlLink'/>
</owl:ObjectProperty>

<owl:ObjectProperty
 rdf:ID='propertyComparatorLink'>
 <rdfs:domain
  rdf:resource='#ComplexFunctionality'/>
 <rdfs:range
  rdf:resource='#PropertyComparatorLink'/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID='propertyGroup'>
 <rdfs:domain
  rdf:resource='#ComplexFunctionality'/>
 <rdfs:range
  rdf:resource='#PropertyGroup'/>
</owl:ObjectProperty>
```

Abbildung B.10: Komplexe Mediatorfunktionalität in MPL.

```
<owl:Class rdf:ID="DataLink">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#source"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#target"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="ParameterMapping">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#from"/>
    <owl:minCardinality rdf:datatype="&xsd;
 nonNegativeInteger">0</owl:minCardinality>
    <owl:maxCardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:maxCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#set"/>
   <owl:minCardinality rdf:datatype="&xsd;
 nonNegativeInteger">0</owl:minCardinality>
   <owl:maxCardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:maxCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#to"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="ParameterSetting">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#parameterValue"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#coding"/>
    <owl:minCardinality rdf:datatype="&xsd;
 nonNegativeInteger">0</owl:minCardinality>
    <owl:maxCardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:maxCardinality>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID='dataMapping'>
 <rdfs:domain rdf:resource='#DataLink'/>
 <rdfs:range
  rdf:resource='#ParameterMapping'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='source'>
 <rdfs:domain rdf:resource='#DataLink'/>
 <rdfs:range rdf:resource='#DataActivity'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='target'>
 <rdfs:domain rdf:resource='#DataLink'/>
 <rdfs:range rdf:resource='#DataActivity'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='from'>
 <rdfs:domain
  rdf:resource='#ParameterMapping'/>
```

```
 <rdfs:range rdf:resource='#Parameter'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='to'>
 <rdfs:domain
  rdf:resource='#ParameterMapping'/>
 <rdfs:range rdf:resource='#Parameter'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='set'>
 <rdfs:domain
  rdf:resource='#ParameterMapping'/>
 <rdfs:range
  rdf:resource='#ParameterSetting'/>
```

```
</owl:ObjectProperty>

<owl:DatatypeProperty
 rdf:ID='parameterValue'>
 <rdfs:domain
  rdf:resource='#ParameterSetting'/>
 <rdfs:range rdf:resource='&xsd;anyType'/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID='coding'>
 <rdfs:domain
  rdf:resource='#ParameterSetting'/>
 <rdfs:range rdf:resource='&xsd;string'/>
</owl:DatatypeProperty>
```

Abbildung B.11: Daten-Links in MPL.

```
<owl:Class rdf:ID="PropertyComparatorLink">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#propSource"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#compTarget"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class
 rdf:ID="PropertyComparatorMapping">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#fromProp"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
```

```
   <owl:onProperty rdf:resource="#toPort"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty
 rdf:ID='propCompMapping'>
 <rdfs:domain
  rdf:resource='#PropertyComparatorLink'/>
 <rdfs:range
rdf:resource='#PropertyComparatorMapping'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='propSource'>
 <rdfs:domain
  rdf:resource='#PropertyComparatorLink'/>
 <rdfs:range rdf:resource='#DataActivity'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='compTarget'>
 <rdfs:domain
  rdf:resource='#PropertyComparatorLink'/>
 <rdfs:range
  rdf:resource='#ComparatorActivity'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='fromProp'>
 <rdfs:domain
rdf:resource='#PropertyComparatorMapping'/>
```

```
 <rdfs:range
  rdf:resource='#MediatorProperty'/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID='toPort'>
 <rdfs:domain
rdf:resource='#PropertyComparatorMapping'/>
 <rdfs:range rdf:resource='#Port'/>
</owl:ObjectProperty>
```

Abbildung B.12: Beziehungen zwischen Properties und Komparatoren in MPL.

```
<owl:Class rdf:ID="ControlLink">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#controlSource"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID='controlSource'>
 <rdfs:domain rdf:resource='#ControlLink'/>
 <rdfs:range rdf:resource='#Activity'/>
</owl:ObjectProperty>

<owl:Class rdf:ID="SimpleLink">
 <rdfs:subClassOf
  rdf:resource='#ControlLink'/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#controlTarget"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID='controlTarget'>
 <rdfs:domain rdf:resource='#SimpleLink'/>
 <rdfs:range rdf:resource='#Activity'/>
</owl:ObjectProperty>

<owl:ObjectProperty
 rdf:ID='conditionalActivity'>
 <rdfs:domain
  rdf:resource='#ConditionalTarget'/>
 <rdfs:range rdf:resource='#Activity'/>
</owl:ObjectProperty>
```

```
<owl:Class rdf:ID="ConditionalLink">
 <rdfs:subClassOf
  rdf:resource='#ControlLink'/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#defaultTarget"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#conditionalTarget"/>
   <owl:minCardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:minCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>


<owl:Class rdf:ID="ConditionalTarget">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
    rdf:resource="#activatedBy"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
   rdf:resource="#conditionalActivity"/>
   <owl:cardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

```
<owl:ObjectProperty rdf:ID='activatedBy'>
 <rdfs:domain
  rdf:resource='#ConditionalTarget'/>
 <rdfs:range rdf:resource='#BoolVariable'/>
</owl:ObjectProperty>

<owl:ObjectProperty
 rdf:ID='conditionalTarget'>
 <rdfs:domain
  rdf:resource='#ConditionalLink'/>
 <rdfs:range
  rdf:resource='#ConditionalTarget'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='defaultTarget'>
 <rdfs:domain
  rdf:resource='#ConditionalLink'/>
 <rdfs:range
  rdf:resource='#ConditionalTarget'/>
</owl:ObjectProperty>
```

Abbildung B.13: Kontrollflussverbindungen in MPL.

```
<owl:Class rdf:ID="PropertyGroup">
 <rdfs:subClassOf><owl:Restriction>
   <owl:onProperty rdf:resource="#valid"/>
   <owl:minCardinality rdf:datatype="&xsd;
 nonNegativeInteger">0</owl:minCardinality>
   <owl:maxCardinality rdf:datatype="&xsd;
 nonNegativeInteger">1</owl:maxCardinality>
 </owl:Restriction></rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="PropertyLink">
 <rdfs:subClassOf><owl:Restriction>
   <owl:onProperty
    rdf:resource="#targetPropActivity"/>
   <owl:cardinality rdf:datatype="&xsd;
    nonNegativeInteger">1</owl:cardinality>
 </owl:Restriction></rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="PropertyMapping">
 <rdfs:subClassOf
  rdf:resource="#ParameterMapping"/>
 <rdfs:subClassOf><owl:Restriction>
   <owl:onProperty rdf:resource="#from" />
   <owl:hasValue
    rdf:resource="#MediatorProperty" />
 </owl:Restriction></rdfs:subClassOf>
 <rdfs:subClassOf><owl:Restriction>
   <owl:onProperty rdf:resource="#to" />
   <owl:hasValue
    rdf:resource="#MediatorProperty" />
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID='propertyLink'>
 <rdfs:domain
  rdf:resource='#PropertyGroup'/>
 <rdfs:range rdf:resource='#PropertyLink'/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID='valid'>
 <rdfs:domain
  rdf:resource='#PropertyGroup'/>
 <rdfs:range rdf:resource='#BoolVariable'/>
</owl:ObjectProperty>

<owl:ObjectProperty
  rdf:ID='targetPropActivity'>
 <rdfs:domain
  rdf:resource='#PropertyLink'/>
 <rdfs:range
  rdf:resource='#FunctionalityActivity'/>
</owl:ObjectProperty>

<owl:ObjectProperty
 rdf:ID='propertyMapping'>
 <rdfs:domain
  rdf:resource='#PropertyLink'/>
 <rdfs:range
  rdf:resource='#PropertyMapping'/>
</owl:ObjectProperty>
```

Abbildung B.14: Property-Gruppen in MPL.

B.3 WSDL-Beschreibungen verwendeter Services

```
<message name="getNucSeqRequest"
 xmlns:tns="http://www.ebi.ac.uk/XEMBL">
 <part name="format" type="xsd:string">
  <documentation>Input parameter that
 indicates the result format that should
 be returned. Legit values: Bsml or sciobj.
 Defaults to Bsml if format not recognised.
  </documentation>
 </part>
 <part name="ids" type="xsd:string">
  <documentation>A space delimited list of
  international Nucleotide Sequence
  accession numbers (IDs). For example:
  "HSERPG U83300 AC000057". Minimum number
  of IDs is 1.</documentation>
 </part>
</message>
```

```
<message name="getNucSeqResponse">
 <part name="result" type="xsd:string">
 <documentation>An XML formatted result in
  either Bsml or AGAVE format.
 </documentation>
 </part>
</message>


<portType name="XEMBLPortType">
 <operation name="getNucSeq">
  <input message="tns:getNucSeqRequest"
   name="getNucSeqRequest" />
  <output message="tns:getNucSeqResponse"
   name="getNucSeqResponse" />
 </operation>
</portType>
```

Abbildung B.15: Port-Typ des XEMBL Web Service.

```
<message name="searchSimple1In">
 <part name="program" type="xsd:string">
 <documentation>specify blastn, blastp,
 blastx, tblastn or tblastx</documentation>
 </part>
 <part name="database" type="xsd:string">
  <documentation>specify database, e.g.
  DDBJ, DDBJ_EXEST, DDBJNEW, DDBJNEW_EXEST
  </documentation>
 </part>
 <part name="query" type="xsd:string">
  <documentation>query sequence
  </documentation>
 </part>
</message>

<message name="searchSimple1Out">
 <part name="Result" type="xsd:string">
  <documentation />
 </part>
</message>
```

```
<portType name="Blast">
 <operation name="searchSimple"
  parameterOrder="program database query">
  <documentation>
  Execute BLAST specified with program,
  database and query. Example searchSimple
  ("blastp","SWISS","MSSRIARALALVVTLLHLTRL
  ALSTCPAACHCPLEAPKCAPGVGLVRDGCGCCKVCAKQL")
  </documentation>
  <input name="searchSimple1In"
   message="tns:searchSimple1In" />
  <output name="searchSimple1Out"
   message="tns:searchSimple1Out" />
 </operation>
</portType>
```

Abbildung B.16: Ausschnitt des Port-Typs des DDBJ Web Service.

B.4 Generiertes Anfrageprofil

```
<profile:MediatorProfile rdf:ID="Query">
 <!-- Query-Functionality -->
 <profile:mediatorFunctionality>
  <profile:MediatorFunctionality
   rdf:ID="idealFunctionality">
   <profile:functionName>
    IdealFunctionality
   </profile:functionName>
   <profile:functionDescription/>

   <profile:inputParameter>
    <profile:Parameter rdf:ID="result">
     <profile:parameterName>result
     </profile:parameterName>
     <profile:parameterDescription>
      An XML formatted result in either
      Bsml or AGAVE format.
     </profile:parameterDescription>
     <profile:rangeOf
      rdf:resource="&xsd;string" />

     <profile:hasConcept>
      <profile:Concept rdf:ID='con104'>
       <profile:conceptRef rdf:resource=
        'BioOntology.owl#dataformat'/>
       <profile:conceptGradation>2
       </profile:conceptGradation>
      </profile:Concept>
     </profile:hasConcept>

     <profile:hasConcept>
      <profile:Concept rdf:ID='con179'>
       <profile:conceptRef rdf:resource=
        'BioOntology.owl#bsml'/>
       <profile:conceptGradation>1
       </profile:conceptGradation>
      </profile:Concept>
     </profile:hasConcept>

     <profile:hasConcept>
      <profile:Concept rdf:ID='con323'>
       <profile:conceptRef rdf:resource=
        'BioOntology.owl#agave'/>
       <profile:conceptGradation>1
       </profile:conceptGradation>
      </profile:Concept>
     </profile:hasConcept>
     <profile:hasConcept>
      <profile:Concept rdf:ID='con303'>
       <profile:conceptRef rdf:resource=
        'BioOntology.owl#xml'/>
       <profile:conceptGradation>1
        </profile:conceptGradation>
      </profile:Concept>
     </profile:hasConcept>
    </profile:Parameter>
   </profile:inputParameter>

   <profile:outputParameter>
    <profile:Parameter rdf:ID="program">
     <profile:parameterName>program
     </profile:parameterName>
     <profile:parameterDescription>
      specify blastn, blastp, blastx,
      tblastn or tblastx
     </profile:parameterDescription>
     <profile:rangeOf
      rdf:resource="&xsd;string" />
    </profile:Parameter>

    <profile:Parameter rdf:ID="database">
     <profile:parameterName>database
     </profile:parameterName>
     <profile:parameterDescription>
      specify database, e.g. DDBJ,
      DDBJ_EXEST, DDBJNEW, DDBJNEW_EXEST
     </profile:parameterDescription>
     <profile:rangeOf
      rdf:resource="&xsd;string" />
    </profile:Parameter>

    <profile:Parameter rdf:ID="query">
     <profile:parameterName>query
     </profile:parameterName>
      <profile:parameterDescription>
       query sequence
      </profile:parameterDescription>
     <profile:rangeOf
      rdf:resource="&xsd;string" />

     <profile:hasConcept>
      <profile:Concept rdf:ID='con258'>
       <profile:conceptRef rdf:resource=
        'BioOntology.owl#sequence'/>
```

```
      <profile:conceptGradation>1
      </profile:conceptGradation>
     </profile:Concept>
    </profile:hasConcept>
   </profile:Parameter>
```

```
    </profile:outputParameter>
   </profile:MediatorFunctionality>
  </profile:mediatorFunctionality>
</profile:MediatorProfile>
```

Abbildung B.17: Auszug aus dem generierten Anfrageprofil basierend auf den Services XEMBL und DDBJ BLAST.

B.5 Testmenge für die Fallstudie

Zur Analyse der Effektivität und Robustheit der software-unterstützten Prozedur wurde eine Testmenge aus verschiedenen Services entwickelt. Vorweg hierzu zwei Bemerkungen:

- Es gibt gegenwärtig keine anerkannten Benchmarks und Testsets zur Evaluierung von Lösungsansätzen, die Service-Interoperabilität adressieren. Deswegen kann die Arbeit zur Evaluierung der Ergebnisse auch auf keine zurückgreifen.
- Zum Zeitpunkt der Arbeit sind über 700 Service Provider im Bereich der Bioinformatik bekannt (siehe MBDC in Kapitel 1), jedoch bieten nur sehr wenige ihre im Internet verfügbaren Dienste als Standard Web Services an. D.h. die meisten Services sind für die Mensch-Maschinen Kommunikation konzipiert und besitzen keine entsprechenden WSDL-Beschreibungen. Daher wurden in dieser Arbeit einige Dienste durch Web Services gekapselt (*Wrapper-Service*) und die Informationen der Web Seiten sowie die Elemente der Web Forms als Dokumentationsgrundlage bzw. als Attributnamen verwendet, um möglichst die Authentizität der Dienste zu bewahren.

Für die verschiedenen Versuche, die in Kapitel 6 diskutiert werden, wurde eine Stichprobe von 30 Service Providern aus der MBDC gezogen, von denen 21 BLAST-Methoden über verschiedene Datenbanken zur Verfügung stellen. Diesen 21 Services liefern insgesamt 34 BLAST-Operationen, die, falls nicht als Web Service vorhanden, als Web Services gekapselt wurden. Die Service Provider und die Anzahl ihrer BLAST-Operationen sind in Tabelle B.1 aufgeschlüsselt. Von den 34 BLAST-Operationen reagieren 11 nur auf Proteinsequenzen, 5 nur auf Nukleotidsequenzen und 18 auf beide Sequenztypen. Damit lassen sich insgesamt 29 Protein-BLAST-Operationen und 23 Nukleotid-BLAST-Operationen unterscheiden. Neben den BLAST-Wrapper-Services wurden drei Datenanbieter als Web Services mit insgesamt 7 Operationen zur Verfügung gestellt, die den Zugriff auf biologische Informationen unterstützen: XEMBL[1], DDBJ GetEntry[2] und NCBI Entrez[3].

Der Mediatorpool bestand bei den Versuchen aus 14 Service-Mediatoren mit insgesamt 24 Mediatorfunktionalitäten. Von diesen Funktionalitäten gehörten sechs der Bio-Domäne,

[1] www.ebi.ac.uk/xembl/
[2] xml.ddbj.nig.ac.jp/wsd/
[3] eutils.ncbi.nlm.nih.gov/entrez/query/static/esoap_help.html

Service Provider	URL	# BLAST-Ops.
AtGDB	`www.plantgdb.org/AtGDB/`	1
BarleyBase	`www.barleybase.org/`	1
BGI-RISe	`rise.genomics.org.cn/`	1
DDBJ	`xml.nig.ac.jp/wsdl/`	2
Diatom EST Database	`avesthagen.sznbowler.com/`	1
ch.EMB.org	`www.ch.embnet.org/`	2
ExPASy	`www.expasy.org/tools/`	6
FlyBase	`flybase.net/`	2
GabiPD	`gabi.rzpd.de/`	1
MaizeGDB	`www.maizegdb.org/`	1
MGI Mouse Protein	`mouseblast.informatics.jax.org/`	2
NPS PBIL	`npsa-pbil.ibcp.fr/`	3
OsGDB	`www.plantgdb.org/OsGDB/`	1
PIR	`pir.georgetown.edu/`	1
PlantGDB	`www.plantgdb.org/`	1
NCBI	`www.ncbi.nlm.nih.gov/`	1
European rRNA database	`www.psb.ugent.be/rRNA/`	1
SYSTERS	`systers.molgen.mpg.de/`	2
TAIR	`www.arabidopsis.org/`	2
TIGR plant repeat database	`www.tigr.org/tdb/e2k1/plant.repeats`	1
EBI	`www.ebi.ac.uk/Tools/webservices/`	1

Tabelle B.1: Service Provider und Anzahl unterstützter BLAST-Operationen

drei der Geo-Domäne und fünf der Finanz-Domäne an. Die restlichen zehn Funktionalitäten wurden als domänenunspezifische klassifiziert. Die komplette Domänenontologie der Registriereinheit beinhaltete insgesamt 67 Konzepte.

Gemessen wurden Precision p, Recall r und, wo möglich, die R-Precision p_r der verwendeten Verfahren (Baeza-Yates und Ribeiro-Neto, 1999). Allgemein, sei *rel* die Menge der relevanten Elemente, *ret* die Menge der zurückgelieferten Elemente, *retrel* die Menge der zurückgelieferten, relevanten Elemente und $retrel_j$ die Menge der relevanten Elemente, die in den ersten j Elementen vorhanden sind. Dann definieren wir

$$p = \frac{|retrel|}{|ret|},\ r = \frac{|retrel|}{|rel|},\ p_r = p_{|rel|} = \frac{|retrel_{|rel|}|}{|rel|}.$$

Vorgehensweise bei der Erstellung der Wrapper-Services

Um die Authentizität der realisierten Wrapper-Services zu garantieren, wurde die folgende, transparente Vorgehensweise gewählt:

BLAST-Operation liegt als *Web Service* zur Verfügung In diesem Fall wurde der Service inklusive seines WSDL-Dokumentes direkt übernommen. Lag die Dokumentation, wie es für DDBJ BLAST der Fall war, in einem externen Dokument zur Verfügung, wurde diese in das WSDL-Dokument unverändert eingefügt.

BLAST-Operation liegt als *Common Gateway Interface* (CGI) zur Verfügung CGI Schnittstellen dienen bereits der Maschinen-Maschinen Kommunikation zwischen einer Anwendung und einem CGI Programm, welches über das Internet auf einem Server verfügbar ist. CGI Programme besitzen jedoch keine Möglichkeit ihre Schnittstelle zu offenbaren. Ihre Schnittstelle wird häufig in Form von *Application Programming Interface* (API) Dokumentationen oder durch Umgangssprache beschrieben.

Lag eine BLAST-Operation als CGI vor, wurden der Operationsname, die unterstützten Parameter (Name und Datentyp), sowie deren Dokumentation aus der API Beschreibung extrahiert und als WSDL-Dokument zur Verfügung gestellt. Dieses Vorgehen wurde beispielsweise bei NCBI's QBLAST angewendet.

BLAST-Operation liegt als *HTML Web Form* zur Verfügung HTML Web Forms dienen der Mensch-Maschinen Kommunikation. Sie werden als Formulare in Web Browsern angezeigt und übermitteln ihre Inhalte zu einem konkreten Server-Programm. Diese Server-Programme sind häufig ebenfalls CGI Programme. Zu den Eingabemöglichkeiten der Web Forms zählen unter anderem Textfelder, Radiobuttons, Checkbuttons und Comboboxen. Auch diese Elemente besitzen konkrete Attributnamen, um die Einträge entsprechend zuordnen zu können.

Bei der Erstellung der Wrapper-Services wurden die Attibutnamen sowie der Operationsname direkt aus den Web Formularen genommen. Die Datentypen wurden aus der jeweiligen Eingabemöglichkeit geschlossen. Beispielsweise folgt aus einer Textbox der Typ String oder aus einem Checkbutton der boolsche Typ. Die inhaltliche Beschreibung zu einer Eingabemöglichkeit wurde abschließend in die Dokumentation des Elementes im WSDL-Dokument übernommen.

Tabellenverzeichnis

Abbildungsverzeichnis

"What is the use of a book", thought Alice,
"without pictures or conversations?"
— Lewis Carroll (1832-90), *Alice's Adventures in Wonderland (1865) ch. 1*

Literaturverzeichnis

Aalst, W. van der: *Don't Go with the Flow: Web Services Composition Standards Exposed.* IEEE Intelligent Systems, Trends & Controversies, 18(1):72–76, Januar/Februar 2003. Web Services: Been There, Done That?

Aalst, W. van der: *The Application of Petri Nets to Workflow Management.* Journal of Circuits, Systems and Computers, 8(1):21–66, 1998.

Aalst, W. van der und A. ter Hofstede: *Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages.* In: K. Jensen (Herausgeber), *Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002)*, Band 560, S. 1–20. Aarhus, Denmark, August 2002.

Aalst, W. van der, A. ter Hofstede, B. Kiepuszewski und A. Barros: *Workflow Patterns.* Distributed and Parallel Databases, 14(3):5–51, Juli 2003.

Adatia, R. : *Professional EJB.* Wrox Press Ltd., Birmingham, UK, 2001.

Alda, S. und A. B. Cremers: *Collaborative Support for Planning Processes through Component-Based Peer Services.* In: *10th International Conference for Concurrent Engineering (CE 2003).* Madeira, Portugal, Juli 2003.

Alda, S. , T. Mitrov und A. Palij: *Semantic Integrity Concepts for Service-Oriented Peer-to-Peer Architectures.* In: *Proceedings of the 2nd IST Workshop on Metadata Management in Grid and P2P Systems (MMGPS)*, LNCS, Springer. London, England, Dezember 2004.

Altschul, S. F. , W. Gish, W. Miller, E. W. Meyers und D. J. Lipman: *Basic local alignment search tool.* Journal of Molecular Biology, 215(3):403–410, Oktober 1990.

Andrews, T. , F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic und S. Weerawarana: *Business Process Execution Language for Web Services - Version 1.1.* BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems, 5. Mai 2003.

Armstrong, E. , J. Ball, S. Bodoff, D. B. Carson, I. Evans, D. Green, K. Haase und E. Jendrock: *The J2EE(TM) 1.4 Tutorial*. Sun Microsystems, `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/J2EETutorial.pdf`, 17. Juni 2004.

Assmann, M. M. und U. Radetzki: *Entwurf und Realisierung eines Toolkits zur einfachen Entwicklung individueller Client-Anwendungen für OPALIS auf Basis der Softwarekomponenten-Technologie*. Diplomarbeit, Rheinische Friedrich-Wilhelms-Universität Bonn, März 2000.

Backofen, R. , F. Bry, P. Clote, H.-P. Kriegel, T. Seidl und K. Schulz: *Bioinformatik: Ziele der Bioinformatik*. Informatik Spektrum, 22:376–378, Oktober 1999.

Baeza-Yates, R. und B. Ribeiro-Neto: *Modern Information Retrieval*. Addison Wesley, 1999.

Baumgarten, B. : *Petri-Netze: Grundlagen und Anwendungen*, Band 2. Spektrum Akademischer Verlag, Heidelberg, 1996.

Bausch, W. , C. Pautasso und G. Alonso: *Programming for Dependability in a Service-based Grid*. In: *Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*. Tokyo, Japan, Mai 2003.

Baxevanis, A. D. : *The Molecular Biology Database Collection: an online compilation of relevant database resources*. Nucleic Acids Research, 28(1):1–7, Januar 2000.

Baxevanis, A. D. : *The Molecular Biology Database Collection: an updated compilation of biological database resources*. Nucleic Acids Research, 29(1):1–10, Januar 2001.

Baxevanis, A. D. : *The Molecular Biology Database Collection: 2002 update*. Nucleic Acids Research, 30(1):1–12, Januar 2002.

Baxevanis, A. D. : *The Molecular Biology Database Collection: 2003 update*. Nucleic Acids Research, 31(1):1–12, Januar 2003.

Baxevanis, A. D. und B. F. F. Ouellette: *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*, Band 2. John Wiley & Sons, 2001.

Beach, B. W. : *Connecting software components with declarative glue*. In: *Proceedings of the 14th International Conference on Software Engineering (ICSE)*, S. 120–137. ACM Press, Melbourne, Australia, 11.-15. Mai 1992. ISBN 0-89791-504-6.

Bernard, L. , U. Einspanier, S. Haubrock, S. Hübner, W. Kuhn, R. Lessing, M. Lutz und U. Visser: *Ontologies for Intelligent Search and Semantic Translation in Spatial Data Infrastructures*. Photogrammetrie - Fernerkundung - Geoinformation (PFG), 6:451–462, 2003.

Bilek, J. , M. Theiß, D. Hartmann, U. Meissner und U. Rüppel: *Integration of Productmodel Databases into Multi-Agent Systems.* In: *10th International Conference on Computing in Civil and Building Engineering (ICCCBE).* Weimar, Germany, 2004.

Biron, P. V. und A. Malhotra: *XML Schema Part 2: Datatypes (XSD).* W3C, `http://www.w3.org/TR/xmlschema-2/`, 2. Mai 2001. W3C Recommendation.

Bode, T. , A. B. Cremers, U. Radetzki und S. Shumilov: *COBIDS: A component-based framework for the integration of geo-applications in a distributed spatial data infrastructure.* In: *Annual Conference of the International Association for Mathematical Geology (IAMG).* Berlin, Germany, 15.-20. September 2002.

Bode, T. , A. B. Cremers, U. Radetzki und S. Shumilov: *The development of COBIDS: A Component-Based Framework for Sharing Standardized and Non-Standardized Geo-Services.* In: *Proceedings of the 18th International Conference Informatics for Environmental Protection (EnviroInfo).* Geneva, Switzerland, 2004.

Booth, D. , H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris und D. Orchard: *Web Services Architecture.* W3C Working Draft, 8. August 2003.

Bowers, S. und B. Ludäscher: *An Ontology-Driven Framework for Data Transformation in Scientific Workflows.* In: *Int. Workshop on Data Integration in the Life Sciences (DILS'04).* 25.-26. März 2004.

Box, D. , D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte und D. Winer: *Simple Object Access Protocol (SOAP) 1.1: W3C Note.* W3C, `http://www.w3.org/TR/SOAP`, Mai 2000.

Brandner, M. , M. Craes, F. Oellermann und O. Zimmermann: *Web services-oriented architecture in production in the finance industry.* Informatik-Spektrum, Springer-Verlag Heidelberg, 27(2):136–145, April 2004.

Brickley, D. und R. V. Guha: *RDF Vocabulary Description Language 1.0: RDF Schema: W3C Recommendation.* W3C, `http://www.w3.org/TR/rdf-schema/`, Februar 2004.

Burks, C. : *Molecular Biology Database List.* Nucleic Acids Research, 27(1):1–9, Januar 1999.

Bussler, C. , D. Fensel und A. Maedche: *A conceptual architecture for semantic web enabled web services.* ACM SIGMOD Record, 31(4):24–29, Dezember 2002. Special Issue: Special section on semantic web and data management.

Carey, M. J. , L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams und E. L. Wimmers: *Towards heterogenous multimedia information systems: The Garlic approach.* In: *Proceedings of the International Workshop on Research Issues in Data Engineering*, S. 161–172. März 1995.

Chawathe, S. , H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman und J. Wisom: *The TSIMMIS project: Integration of heterogeneous information sources.* In: *Proceedings of IPSJ Conference*, S. 7–18. Tokyo, Japan, Oktober 1994.

Christensen, E. , F. Curbera, G. Meredith und S. Weerawarana: *Web Service Description Language (WSDL) 1.1: W3C Note.* W3C, `http://www.w3.org/TR/wsdl`, März 2001.

Cremers, A. B. , S. Alda und U. Radetzki: *Towards Semantic Grid in Construction Informatics.* In: *Proceedings of the 22nd International Conference Information Technology in Construction (CIB-W78).* Dresden, Germany, Juli 2005. Invited Keynote Paper.

Curbera, F. , M. Duftler, R. Khalaf, W. Nagy, N. Mukhi und S. Weerawarana: *Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI.* IEEE Internet Computing, 6(2):86–93, März/April 2002.

Curbera, F. , W. A. Nagy und S. Weerawarana: *Web Services: Why and How.* In: *Workshop on Object-Oriented Web Services (OOPSLA 2001).* Tampa, Florida, USA, Oktober 2001.

Czajkowski, K. , D. F. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling und S. Tuecke: *From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution*, 5. März 2004a.

Czajkowski, K. , F. D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke und W. Vambenepe: *The WS-Resource Framework, Version 1.0*, März 2004b.

Discala, C. , X. Benigni, E. Barillot und G. Vaysseix: *DBcat: a catalog of 500 biological databases.* Nucl. Acids Research, 28(1):8–9, Januar 2000.

Do, H.-H. und E. Rahm: *COMA - A System for Flexible Combination of Schema Matching Approaches.* In: *Proc. 28th Intl. Conference on Very Large Databases (VLDB).* Hongkong, 2002.

Dong, X. , A. Halevy, J. Madhavan, E. Nemes und J. Zhang: *Similarity Search for Web Services.* In: *Proceedings of the 30th VLDB Conference.* Toronto, Canada, 2004.

EBI: *XEMBL Project.* European Bioinformatics Institute, `http://www.ebi.ac.uk:80/xembl/`, September 2001.

Elysium-Gates: *Historical Iris*. `http://www.elysiumgates.com/`, 2001.

Eskelin, P. : *Component Interaction Patterns*. In: *Proceedings of PLoP 1999 Conference (Pattern Languages of Programs'99)*. 15.-18. August 1999.

Eyle, B. van: *Web Services: A Business Perspective on Platform Choice*. TheServerSide.com, August 2001.

Fensel, D. und C. Bussler: *The Web Service Modeling Framework WSMF*. Electronic Commerce Research and Applications, 1(2):113–137, 2002.

Fischer, G. : *User Modeling in Human-Computer Interaction*. User Modeling and User-Adapted Interaction (UMUAI), 11:65–86, 2001.

Foster, I. , D. Berry, A. Djaoui, A. Grimshaw, B. Horn, H. Kishimoto, F. Maciel, A. Savva, F. Siebenlist, R. Subramaniam, J. Treadwell und J. V. Reich: *The Open Grid Services Architecture, Version 1.0*. Global Grid Forum Draft Recommendation, 12. Juli 2004a.

Foster, I. , J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe und S. Weerawarana: *Modeling Stateful Resources with Web Services Version 1.1*. Globus, 5. März 2004b.

Foster, I. : *What is the Grid? A Three Point Checklist*. GRIDToday, 20. Juli 2002.

Foster, I. , C. Kesselman, J. M. Nick und S. Tuecke: *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. Open Grid Service Infrastructure WG, Global Grid Forum, 22. Juni 2002.

Foster, I. , C. Kesselman und S. Tuecke: *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International J. Supercomputer Applications, 15(3), 2001.

Galperin, M. Y. : *The Molecular Biology Database Collection: 2004 update*. Nucleic Acids Research, 32:D3–D22, Januar 2004. Database issue.

Galperin, M. Y. : *The Molecular Biology Database Collection: 2005 update*. Nucleic Acids Research, 33:D5–D24, 2005. Database issue.

Gamma, E. , R. Helm, R. Johnson und J. Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

Glass, G. : *The Web services (r)evolution: Applying Web services to applications*. IBM - developerWorks, November 2000.

Gruber, T. R. : *A translation approach to portable ontologies*. Knowledge Acquisition, 5(2):199–220, 1993.

Han, Y. , A. Sheth und C. Bussler: *A Taxonomy of Adaptive Workflow Management.* In: *Workshop Towards Adaptive Workflow Systems (CSCW-98).* 1998.

Harrison, J. und M. Reichardt: *Introduction to OGC Web Services.* Open GIS Consortium, Mai 2001.

Henderson, A. und M. Kyng: *There's No Place Like Home: Continuing Design in Use.* In: J. Greenbaum und M. Kyng (Herausgeber), *Design at Work: Cooperative Design of Computer Systems*, Kapitel 11, S. 219–240. Lawrence Erlbaum Associates, Publishers, 1991.

Hendler, J. und D. L. McGuinness: *The DARPA Agent Markup Language.* IEEE Intelligent Systems, 15(6):67–73, November/Dezember 2000.

Hofestädt, R. : *Berichte und Meldungen/Bioinformatik 2000.* Informatik Spektrum, 22:385–390, Oktober 1999.

Hull, D. , R. Stevens, P. Lord und C. Goble: *Integrating bioinformatics resources using shims.* In: *Poster session for Intelligent Systems in Molecular Biology (ISMB).* Glasgow, Scotland, UK, 2004.

Hunt, J. : *A Brief History of Service-Oriented Architectures, Part I.* IONAsphere, Mai 2003.

Ibba, A. : *Erweiterungen und Verbesserungen in der neuen J2EE-Version 1.4.* javamagazin, S. 37–43, Februar 2004.

Jones, K. S. : *A statistical interpretation of term specificity and its applications in retrieval.* Journal of Documentation, 28(1):11–21, 1972.

Kanapin, A. A. , W. Fleischmann und R. Apweiler: *Integration of Biological Databases: the Example of InterPro.* In: N. El-Mabrouk, T. Lengauer und D. Sankoff (Herausgeber), *Currents in Computational Molecular Biology (RECOMB 2001)*, S. 227–228. Les Publications CRM, Montreal, Canadia, 2001. ISBN 2-921120-35-6.

Kao, J. : *Developer's Guide to Bilding XML-based Web Service: With the Java 2 Platform, Enterprise Edition (J2EE).* The Middleware Company, Juni 2001.

Keidl, M. , A. Kemper, S. Seltzsam und K. Stocker: *Web Services.* In: E. Rahm und G. Vossen (Herausgeber), *Web & Datenbanken: Konzepte, Architekturen, Anwendungen*, S. 293–331. dpunkt-Verlag, Heidelberg, 2003.

Klusch, M. und K. P. Sycara: *Brokering and Matchmaking for Coordination of Agent Societies: A Survey.* In: A. Omicini, F. Zambonelli, M. Klusch und R. Tolksdorf (Herausgeber), *Coordination of Internet Agents: Models, Technologies, and Applications*, S. 197–224. Springer, 2001.

Kossmann, D. und F. Leymann: *Web Services.* Informatik-Spektrum, Springer-Verlag Heidelberg, 27(2):117–128, April 2004.

Krause, A. und M. Vingron: *A set-theoretic approach to database searching and clustering.* Bioinformatics, 14:430–438, 1998.

Krishnan, N. : *The Jxta Solution to P2P.* JavaWorld, 19. Oktober 2001.

Langner, T. : *Implementierung der Direktbank-Anwendung in .NET und Java: eine Bewertung.* javamagazin, S. 40–42, Dezember 2003.

Lassila, O. : *The Resource Description Framework.* IEEE Intelligent Systems, 15(6):67–73, November/Dezember 2000.

Lassila, O. und R. R. Swick: *Resource Description Framework (RDF) Model and Syntax Specification: W3C Recommendation.* W3C, `http://www.w3.org/TR/REC-rdf-syntax/`, Februar 1999.

Leser, U. : *Query Planning in Mediator-based Information Systems.* Doktorarbeit, Technische Universität Berlin, September 2000. Graduate School for Distributed Information Systems.

Levy, A. Y. , A. Rajaraman und J. J. Ordille: *Querying Heterogeneous Information Sources Using Source Descriptions.* In: *Proceedings of the Twenty-second International Conference on Very Large Databases*, S. 251–262. VLDB Endowment, Saratoga, Calif., Bombay, India, September 1996.

Leymann, F. : *Web Services Flow Language: (WSFL 1.0).* `http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`, Mai 2001.

Ludäscher, B. , I. Altintas und A. Gupta: *Compiling Abstract Scientific Workflows into Web Service Workflows.* In: *Proceedings of the 15th International Conference on Scientific and Statistical Database Management (SSDBM 2003)*, S. 251–254. IEEE Computer Society, Cambridge, MA, USA, 9.-11. Juli 2003.

Lutz, M. , C. Riedemann und F. Probst: *A Classification Framework for Approaches to Archieving Semantic Interoperability Between GI Web Services.* Lecture Notes in Computer Science, 2825, 2003.

Madhavan, J. , P. A. Bernstein, K. Chen, A. Halevy und P. Shenoy: *Corpus-based Schema Matching.* In: *Workshop on Information Integration on the Web at the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'2003).* Acapulco, Mexico, 2003.

Maedche, A. und S. Staab: *Learning Ontologies for the Semantic Web.* In: *Proceedings of the 2nd International Workshop on the Semantic Web - SemWeb'2001.* CEUR Workshop Proceedings, Hongkong, China, Mai 2001.

Manes, A. T. : *New Web service need structure.* eeTimes.com, Juni 2001.

Martin, D. , M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, B. P. Deborah McGuinness, T. Payne, M. Sabou, M. Solanki, N. Srinivasan und K. Sycara: *Bringing Semantics to Web Services: The OWL-S Approach.* In: *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004).* San Diego, CA, USA, 6.-9. Juli 2004.

McIlraith, S. A. , T. C. Son und H. Zeng: *Mobilizing the Semantic Web with DAML-Enabled Web Services.* In: *Proceedings of the 2nd International Workshop on the Semantic Web - SemWeb'2001.* CEUR Workshop Proceedings, Hongkong, China, Mai 2001.

McIlroy, M. D. : *Mass produced software components.* In: *Proceedings Nato Software Engineering Conference*, S. 138–155. 1968.

Mohebbian, S. : *Ein Clusterverfahren zur Ähnlichkeitssuche von Services und Service-Mediatoren.* Diplomarbeit, Rheinische Friedrich-Wilhelms-Universität Bonn, 2006. (In Arbeit).

Moran, M. und A. Mocan: *WSMX – An Architecture for Semantic Web Service Discovery, Mediation and Invocation.* In: *Third International Semantic Web Conference (ISWC 2004).* Hiroshima, Japan, 2004. Poster.

Mørch, A. : *Three Levels of End-user Tailoring: Customization, Integration, and Extension.* In: *Proceedings of the Third Decennial Aarhus Conference*, S. 157–166. Aarhus, DK, August 1995.

Murata, T. : *Petri Nets: Properties, Analysis and Applications.* In: *Proceedings of the IEEE*, Band 77 (4), S. 541–580. April 1989.

Narayanan, S. und S. A. McIlraith: *Simulation, Verification and Automated Composition of Web Services.* In: *Proceedings of the Eleventh International World Wide Web Conference (WWW2002)*, S. 77–88. Honolulu, Hawaii, USA, 7.-11. Mai 2002.

Nardi, D. und R. J. Brachman: *An Introduction to Description Logics.* In: F. Baader, D. Calvanese, D. MyGuinness, D. Nardi und P. Patel-Schneider (Herausgeber), *The Description Logic Handbook - Theory, Implementation and Applications*, S. 5–44. Cambridge University Press, UK, 2004.

Naumann, F. und U. Leser: *Density Scores for Cooperative Query Answering.* In: *Proceedings of the 4th Workshop: Föderierte Datenbanken.* Berlin, Germany, November 1999.

Newport, B. : *Requirements for Building Industrial Strength Web Services: The Service Broker.* TheServerSide.com, Juli 2001.

Nierstrasz, O. und F. Achermann: *A Calculus for Modeling Software Components.* In: F. S. de Boer, M. M. Bonsangue, S. Graf und W. P. de Roever (Herausgeber), *Formal Methods for Components and Objects, First International Symposium (FMCO 2002)*, Band 2852 von *Lecture Notes in Computer Science (LNCS)*, S. 339–360. Springer, Leiden, The Netherlands, 5.-8. November 2003.

Noy, N. F. und D. L. McGuinness: *Ontology Development 101: A Guide to Creating Your First Ontology.* Forschungsbericht, Knowledge Systems Laboratory (KSL), ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-01-05.pdf, März 2001.

Oasis: *UDDI Specification*, 2003.

Oinn, T. , M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, M. R. Pocock, A. Wipat und P. Li: *Taverna: a tool for the composition and enactment of bioinformatics workflows.* Bioinformatics, 2004. accepted.

Open GIS Consortium: `http://www.opengis.net/gml/01-029/GML2.html`. *Geography Markup Language (GML) 2.0: OpenGIS Implementation Specification*, Februar 2001.

Pallos, M. S. : *Service-Oriented Architecture: A Primer.* eAI Journal, S. 32–35, Dezember 2001.

Paolucci, M. , T. Kawamura, T. R. Payne und K. Sycara: *Semantic Matching of Web Services Capabilities.* In: I. Horrocks und J. Hendler (Herausgeber), *First International Semantic Web Conference (ISWC)*, Band 2342 von *Lecture Notes in Computer Science (LNCS)*, S. 333–347. Springer-Verlag Heidelberg, Sardinia, Italy, Juni 2002.

Preece, A. D. , K. Y. Hui, W. A. Gray, P. Marti, T. J. M. Bench-Capon, D. M. Jones und Z. Cui: *The KRAFT architecture for knowledge fusion and transformation.* Knowledge Based Systems, 13(2-3):113–120, 2000.

Qian, X. und T. F. Lunt: *Semantic interoperation: A query mediation approach.* Forschungsbericht SRI-CSL-94-02, Computer Science Laboratory, SRI International, April 1994.

Radetzki, U. , S. Alda, T. Bode und A. B. Cremers: *First Steps in the Development of a Web Service Framework for Heterogeneous Environmental Information Systems.* In: W. Pillmann und K. Tochtermann (Herausgeber), *Proceedings of the 16th International Conference Informatics for Environmental Protection (EnviroInfo)*, Band 1, S. 384–391. ISEP, Vienna, Austria, 25.-27. September 2002a.

Radetzki, U. , T. Bode und A. B. Cremers: *Mediatorbasierte ad hoc Integration autonomer Web Services.* In: *Informatik 2004, 34. Jahrestagung der Gesellschaft für Informatik (GI), Workshop: Dynamische Informationsfusion.* Gesellschaft für Informatik (GI), Ulm, Germany, 20.-24. September 2004a.

Radetzki, U. , T. Bode, G. Witterstein, M. Gnasa und A. B. Cremers: *A Service-Centric Computing Environment for Heterogeneous Biological Databases and Methods.* In: *Currents in Computational Molecular Biology (RECOMB 2003)*. Berlin, Germany, April 2003.

Radetzki, U. und A. B. Cremers: *IRIS: A Framework for Mediator-Based Composition of Service-Oriented Software.* In: *2004 IEEE International Conference on Web Services (ICWS 2004)*, S. 752–755. IEEE, San Diego, California, USA, 6.-9. Juli 2004.

Radetzki, U. , U. Leser und A. B. Cremers: *IRIS: A Mediator-Based Approach Achieving Interoperability of Web Services in Life Science Applications.* In: *3rd E-BioSci / ORIEL Annual Workshop*, S. 25–26. Hinxton, England, 12.-15. Oktober 2004b. Invited Talk Paper.

Radetzki, U. , S. Mancke und A. B. Cremers: *IRIS: A Framework Supporting Composition and Device-Specific Access of Software Services.* In: *Proceedings of the 18th International Conference Informatics for Environmental Protection (EnviroInfo)*. Geneva, Switzerland, 2004c.

Radetzki, U. , G. Witterstein und A. B. Cremers: *An Integration Platform for Heterogeneous Services in Life Science Applications.* In: *Workshop on Bioinformatics - 8th International Symposium on Methodologies for Intelligent Systems (ISMIS)*. Lyon, France, 26. Juni 2002b.

Rahm, E. und P. A. Bernstein: *A survey of approaches to automatic schema matching.* VLDB Journal, 10(4):334–350, 2001.

Reichert, M. , S. Rinderle und P. Dadam: *ADEPT Workflow Management System: Flexible Support For Enterprise-wide Business Processes (Tool Presentation).* In: *International Conference on Business Process Management (BPM '03)*, Nummer 2678 in LNCS, S. 370–379. Eindhoven, The Netherlands, Juni 2003.

Reinefeld, A. und F. Schintke: *Grid Services: Web Services zur Nutzung verteilter Ressourcen.* Informatik-Spektrum, Springer-Verlag Heidelberg, 27(2):129–135, April 2004.

Rodriguez-Tomé, P. : *The BioCatalog.* Bioinformatics, 14(5):469–470, 1998.

Roman, E. , S. Ambler und T. Jewell: *Mastering Enterprise JavaBeans.* Wiley, 2. Auflage, 2002. 3rd edition is in review.

Salton, G. : *The SMART Retrieval System - Experiments in Automatic Document Processing.* Prentice Hall, 1971.

Sattler, K.-U. , I. Geist, R. Habrecht und E. Schallehn: *Konzeptbasierte Anfrageverarbeitung in Mediatorsystemen.* In: G. Weikum, H. Schöning und E. Rahm (Herausgeber), *BTW 2003: Datenbanksysteme für Business, Technologie und Web, Tagungsband*

der 10. BTW-Konferenz, 26.-28. Februar 2003, Leipzig, Nummer P-26 in GI-Edition - Lecture Notes in Informatics (LNI), S. 78–97. Bonner Köllen Verlag, 2003. ISBN 3-88579-355-5.

Schneider, J. : *Convergence of Peer and Web Services.* O'Reilly OpenP2P.com, 20. Juli 2001.

Schuler, C. , R. Weber, H. Schuldt und H.-J. Schek: *Scalable Peer-to-Peer Process Management – The OSIRIS Approach.* In: *2004 IEEE International Conference on Web Services (ICWS 2004)*, S. 26–34. IEEE, San Diego, California, USA, 6.-9. Juli 2004.

Seiter, L. M. , M. Mezini und K. J. Lieberherr: *Dynamic Component Gluing.* In: *OOPSLA'99: Workshop on Multi-Dimensional Separation of Concerns (MDSOC).* ACM Sigplan, November 1999. OOSPLA'99: MDSOC Workshop.

Shirky, C. : *What Is P2P... And What Isn't.* O'Reilly Network, 11. November 2000.

Shumilov, S. : *Integrating existing object-oriented databases with distributed object management platforms.* Doktorarbeit, Rheinische Friedrich-Wilhelms-Universität Bonn, 2003.

Shumilov, S. , A. Thomsen, A. B. Cremers und B. Koos: *Management and visualization of large, complex and time-dependent 3D objects in distributed GIS.* In: *Proceedings of the 10th International Symposium on Advances in Geographic Information Systems.* McLean, USA, November 2002.

Siepel, A. C. , A. N. Tolopko, A. D. Farmer, P. A. Steadman, F. D. Schilkey, B. D. Perry und W. D. Beavis: *An integration platform for heterogeneous bioinformatics software components.* IBM Systems Journal, 40:570–591, 2001.

Stevens, R. D. , A. J. Robinson und C. A. Goble: *myGrid: personalised bioinformatics on the information grid.* Bioinformatics, 19:302–304, 2003.

Stevens, R. D. , H. J. Tipney, C. J. Wroe, T. M. Oinn, M. Senger, P. W. Lord, C. A. Goble, A. Brass und M. Tassabehji: *Exploring Williams-Beuren syndrome using myGrid.* Bioinformatics, 20:I303–I310, 2004.

Stevens, R. , C. Goble, P. Baker und A. Brass: *A classification of tasks in bioinformatics.* Bioinformatics, 17(2):180–188, 2001.

Stojanovic, Z. und A. Dahanayake: *Service-Oriented Software System Engineering: Challenges and Practices.* Idea Group Inc., 2005. ISBN 1-59140-427-4.

Sun Microsystems, Inc.: Santa Clara, CA, U.S.A. *J2EE(TM) Connector Architecture Specification - Version 1.5*, November 2003.

Sycara, K. , M. Klusch, S. Widoff und J. Lu: *Dynamic Service Matchmaking Among Agents in Open Information Environments.* ACM SIGMOD Records, 28(1):47–53, 1999.

Sycara, K. , S. Widoff, M. Klusch und J. Lu: *LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace.* Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, 5:173–203, 2002.

Szyperski, C. : *Component Software: Beyond Object-Oriented Programming.* Addison Wesley, 1998.

Szyperski, C. , D. Gruntz und S. Murer: *Component Software: Beyond Object-Oriented Programming.* Component Software Series. ACM Press, Addison-Wesley, 2. Auflage, 2002. ISBN 0-201-74572-0.

Tatarinov, I. , Z. G. Ives, J. Madhavan, A. Y. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau und P. Mork: *The Piazza peer data management project.* SIGMOD Record, 32(3), 2003.

Thatte, S. : *XLANG - Web Services for Business Process Design.* Microsoft Corporation, 2001.

Tuecke, S. , K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt und D. Snelling: *Open Grid Services Infrastructure (OGSI) Version 1.0.* Global Grid Forum Draft Recommendation, 27. Juni 2003.

Uschold, M. und M. Gruninger: *Ontologies: Principles, Methods and Applications.* Knowledge Engineering Review, 11(2), Juni 1996.

Vasudevan, V. : *A Web Service Primer.* O'Reilly XML.com, April 2001.

Vawter, C. und E. Roman: *J2EE vs. Microsoft.NET: A comparsion of building XML-based web services.* The Middleware Company, Juni 2001.

W3C: `http://www.w3.org/TR/owl-guide/`. *OWL Web Ontology Language Guide*, 10. Februar 2004a. W3C Recommendation.

W3C: `http://www.w3.org/TR/owl-features/`. *OWL Web Ontology Language Overview*, 10. Februar 2004b. W3C Recommendation.

Wache, H. , T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann und S. Hübner: *Ontology-based Integration of Information - A Survey of Existing Approaches.* In: H. Stuckenschmidt (Herausgeber), *IJCAI-01 Workshop: Ontologies and Information Sharing*, S. 108–117. 2001.

Wang, D. : *Einführung in UDDI und JAXR.* javamagazin, S. 43–49, Dezember 2003.

Werner, C. , C. Buschmann und S. Fischer: *Compressing SOAP Messages by using Differential Encoding.* In: *2004 IEEE International Conference on Web Services (ICWS 2004)*, S. 540–547. IEEE, San Diego, California, USA, 6.-9. Juli 2004.

WFMC: *Terminology & Glossary.* Forschungsbericht WFMC-TC-1011, Workflow Management Coalition, Hampshire, UK, Februar 1999.

Wiederhold, G. : *Mediators in the Architecture of Future Information Systems.* IEEE Computer, 25(3):38–49, März 1992.

Wohed, P. , W. van der Aalst, M. Dumas und A. ter Hofstede: *Pattern-Based Analysis of BPEL4WS.* QUT Technical report FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.

Wojciechowski, R. und C. Weinhardt: *Web Services und Peer-to-Peer-Netzwerke.* In: D. Schoder, K. Fischbach und R. Teichmann (Herausgeber), *Peer-to-Peer: Ökonomische, technologische und juristische Perspektiven*, S. 99–117. Springer, August 2002.

Won, M. : *Interaktive Integritätsprüfung für komponentenbasierte Architekturen: Technische Unterstützung für Endanwender beim Anpassen komponentenbasierter Software.* Doktorarbeit, Rheinische Friedrich-Wilhelms-Universität Bonn, 2004.

Yan, L.-L. , M. T. Oezsu und L. Liu: *Accessing Heterogeneous Data Through Homogenization and Integration Mediators.* In: *Proceedings of the Intern. Conference on Cooperative Information Systems (CoopIS)*, S. 130–139. Kiawah Island, North Carolina, 1997.

Zaremski, A. M. und J. M. Wing: *Specification Matching of Software Components.* ACM Transactions on Software Engineering and Methodology (TOSEM), 6(4):333–369, 1997.