Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Hulda Ngadjeu

Lösungsansätze für Semantic Web Services

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Hulda Ngadjeu Tchana

Lösungsansätze für Semantic Web Services

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Departement Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Michael Neitzke
Zweitgutachter: Prof. Dr. Olaf Zukunft

Abgegeben am 10. Januar 2009

Hulda Ngadjeu Tchana

Thema der Bachelorarbeit

Lösungsansätze für Semantic Web Services

Stichworte

Service-orientierte Architektur, Semantic Web, Web Services, Semantic Web Services, WSMO

Zusammenfassung

Das derzeitige World Wide Web dient als Kommunikationsmittel für zwei Technologien: Semantic Web und Web Services.

Web Services ist eine plattformunabhängige Technologie, die es ermöglicht, die Funktionalitäten eines Dienstes und ihre Schnittstellen über das Web zu veröffentlichen und zu nutzen. Da Semantic Web es ermöglicht, die Ressourcen des Internets für Maschinen verständlich zu machen, wird es zur Automatisierung der Nutzung von Webdiensten verwendet. Die Semantic Web Services stellen eine Zusammenführung von Semantic Web und Web Services dar, deren verschiedene Technologien in dieser Arbeit analysiert werden. Außerdem werden Dienste anhand der Web Service Modeling Ontologie (WSMO) semantisch beschrieben und ausgeführt.

Hulda Ngadjeu Tchana

Title of the paper

Solution Proposals for Semantic Web Services

Keywords

Service-oriented Architecture, Semantic Web, Web Services, Semantic Web Services, WSMO

Abstract

The current World Wide Web serves as a means of communication for two technologies: Semantic Web and Web Services.

Web Services is a platform independent technology, which enables the publication and the use of the service functionality with their interfaces. Because Semantic Web enables the understanding of Internet resources by machines, it is used for the automation of Web Services processing and mainly the discovery of services. This combination of both technologies is Semantic Web Services, whose different technologies will be analysed in this work. Moreover, services are semantically described and executed on the basis of Web Service Modeling Ontology (WSMO).

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listingsverzeichnis

Abkürzungsverzeichnis

DAML-S	DARPA Agent Mark-up Language for Web Services ontologies.
LP	Logische Programmierung.
OWL-S	Web Ontology Language for Web Services.
PL	Prädikatenlogik.
RDF	Resource Description Framework.
SOA	Service-Orientierte Architektur.
SOAP	Simple Object Access Protocol.
SWS	Semantic Web Service.
UDDI	Universal Description, Description and Integration.
WS	Web Services
WSDL-S	Web Services Description Language-Semantic.
WSML	Web Service Modeling Language.
WSMO	Web Service Modeling Ontology.
WSMX	Web Service Execution Environment.
XML	eXtensible Markup Language.

1 Einleitung

1.1 Motivation

Das Internet hat sich in den letzten Jahren zu einem Massenmedium entwickelt und stellt eine riesige und weit verbreitete Informationsquelle dar. Hier werden Dokumente und andere Informationsobjekte von Benutzern veröffentlicht, die mittels Suchmaschinen (z.B. „Google“) von anderen Benutzern aufgefunden und aufgerufen werden können. Eine Suche mit maschinenverständlicher Bedeutung ist jedoch nicht möglich, da die im Web enthaltenen Informationen sich nicht maschinell verarbeiten lassen. Die derzeitigen Suchmaschinen sind nicht in der Lage präzise zu suchen, da sie die Inhalte der Informationen nicht interpretieren können. Sie suchen durch Schlüsselwörter und dabei werden alle möglichen Webseiten, auf die eines der Schlüsselwörter zutrifft bzw. auf denen sie verzeichnet sind, dem Nutzer angezeigt. Dies ist für den Nutzer zeitaufwendig, da er die Angebote selbst vergleichen muss, um das beste mögliche Angebot zu finden.

Die Vision des Semantic Webs, die von Timothy Berners-Lee[1] (dem Erfinder des World Wide Web) geprägt wurde, ist nun, das im Web enthaltene Wissen nicht nur für Menschen verstehbar zu machen, sondern auch für Maschinen (Rechner). Damit soll eine automatische Nutzung von Information erreicht werden.

Web Services ist neben dem Semantic Web eine weitere Technologie, die sich dem Internet als Kommunikationsmittel bedient. Hierbei handelt es sich um Dienste, die auf einem Server laufen. Derzeitige Web Services bieten Standards, die es ermöglichen, Dienstangebote manuell zu publizieren, zu suchen, zu finden und aufzurufen.

Mit Hilfe von Semantic Web wird die Nutzung von Diensten in Web Services automatisiert, indem diese semantisch beschrieben werden. Dieser Ansatz wird „Semantic Web Services“ („semantisch beschriebene Webdienste“) genannt. Damit sollen Dienste nicht mehr manuell genutzt werden, sondern maschinell.

Semantic Web Services bieten eine Reihe von Vorteilen. Diese Vorteile resultieren einerseits aus dem Semantic Web und andererseits aus den Web Services. Im Web können die darin enthaltenen Informationen präsentiert, aber nicht maschinell verarbeitet werden. Dieser Mangel wird von Semantic Web kompensiert. Als Erweiterung des Webs stellt Semantic Web formale Sprachen zur

[1] http://www.w3c.org/People/Berners-Lee

Beschreibung der Daten bereit. Dies führt dazu, dass die Daten maschinell verarbeitet werden und die semantische Interoperabilität gewährleistet ist. Web Services ermöglichen durch ihre verwendeten Standards eine syntaktische Beschreibung der Daten (syntaktische Interoperabilität). Als weitere Vorteile werden vor allem die Unabhängigkeit vom verwendeten Betriebssystem und der verwendeten Programmiersprache genannt. Durch die Integration von Semantic Web in Web Services wird die Suche nach einem Dienst im Dienstverzeichnis verbessert, da das Dienstverzeichnis semantische Informationen enthält und der Dienstnutzer durch Softwareagenten[2] ersetzt wird.

1.2 Ziele

Durch das große Interesse im Bereich von Semantic Web Services gibt es verschiedene Ansätze, an denen derzeit geforscht wird:

- Web Services Description Language-Semantic (WSDL-S),
- Semantic Web Services Framework (SWSF),
- Web Ontology Language-Services (OWL-S) und
- Web Service Modeling Ontology (WSMO).

Im Moment ist noch nicht abzusehen, welcher dieser Ansätze sich in der Zukunft durchsetzen wird, eine Analyse ist jedoch möglich. Die Analyse soll die Technologien nach bestimmten Kriterien bewerten und zum Schluss den Ansatz, der die Anforderungen am besten erfüllt, auswählen. Das erste Ziel dieser Arbeit liegt daher darin, diese Analyse durchzuführen. Das zweite Ziel ist es, mit Hilfe der dabei ausgewählten Technologie das Design und die Implementierung eines Anwendungsbeispiels zu erreichen. In dem Anwendungsbeispiel geht es um ein Wagenreservierungssystem. Das dritte Ziel ist dann die Bewertung der Umsetzung des ausgewählten Ansatzes bei der Realisation des Wagenreservierungssystems.

1.3 Gliederung der Arbeit

Kapitel 1: Einleitung. Dieses Kapitel leitet die Arbeit mit der Erläuterung der Motivation für diese Arbeit ein. Außerdem werden die Ziele dieser Arbeit vorgestellt.

Kapitel 2: Grundlagen. Dieses Kapitel stellt die Technologien und Konzepte, die zum Verständnis der Ziele dieser Arbeit notwendig sind, vor. Konkret werden zunächst die Grundlagen von Web Services und Semantic Web beschrieben. Anschließend wird das Konzept von Semantic Web Services erläutert.

Kapitel 3: Aktueller Stand der Forschung. Hier wird der derzeitige Stand der Forschung analysiert, indem die Lösungsansätze für Semantic Web Services vorgestellt und verglichen werden.

Kapitel 4: Design des Prototyps. In diesem Kapitel wird das Design des Prototyps vorgestellt. Dabei wird erstmal ein mögliches Anwendungsbeispiel gegeben. Dann werden die notwendigen

[2] Dies wird im Abschnitt 2.7 beschrieben.

Komponenten und Anforderungen untersucht. Zum Schluss wird eine Domain-Ontologie entwickelt.

Kapitel 5: Implementierung. Die im Kapitel 4 aufgestellten Anforderungen und die entwickelte Ontologie werden in diesem Kapitel implementiert.

Kapitel 6: Bewertung. Die während der Implementierung gesammelten Erfahrungen und Bemerkungen werden in diesem Kapitel vorgestellt.

Kapitel 7: Zusammenfassung und Ausblick. Hier wird die Arbeit zusammengefasst und ein Ausblick auf mögliche Erweiterungen oder Verbesserungen gegeben.

2 Grundlagen

Zur Erreichung seiner Ziele nutzen Semantic Web Services verschiedene Technologien. Daher wird dieses Kapitel die Technologien, die für das Verständnis dieser Arbeit notwendig sind, beschreiben. Es wird in der Abbildung 2.1 ein Überblick über die Organisation dieses Kapitels gegeben und besonders die Beziehungen zwischen den dargestellten Technologien aufgezeigt. Zuerst wird XML im Abschnitt 2.1 erläutert. Dann wird die Service-orientierte Architektur in Abschnitt 2.2 vorgestellt, um die Architektur der Web Services besser verstehen zu können. Anschließend werden die Web Services in Abschnitt 2.3 beschrieben. Die formalen Grundlagen, die zur Wissensrepräsentation im Semantic Web dienen, werden im Abschnitt 2.4 behandelt. Danach werden Semantic Web und seine Ontologiesprachen im Abschnitt 2.5 beschrieben. Das Konzept der Semantic Web Services im Abschnitt 2.6 vorgestellt. Abschließend wird Softwareagent im Abschnitt 2.7 erläutert.

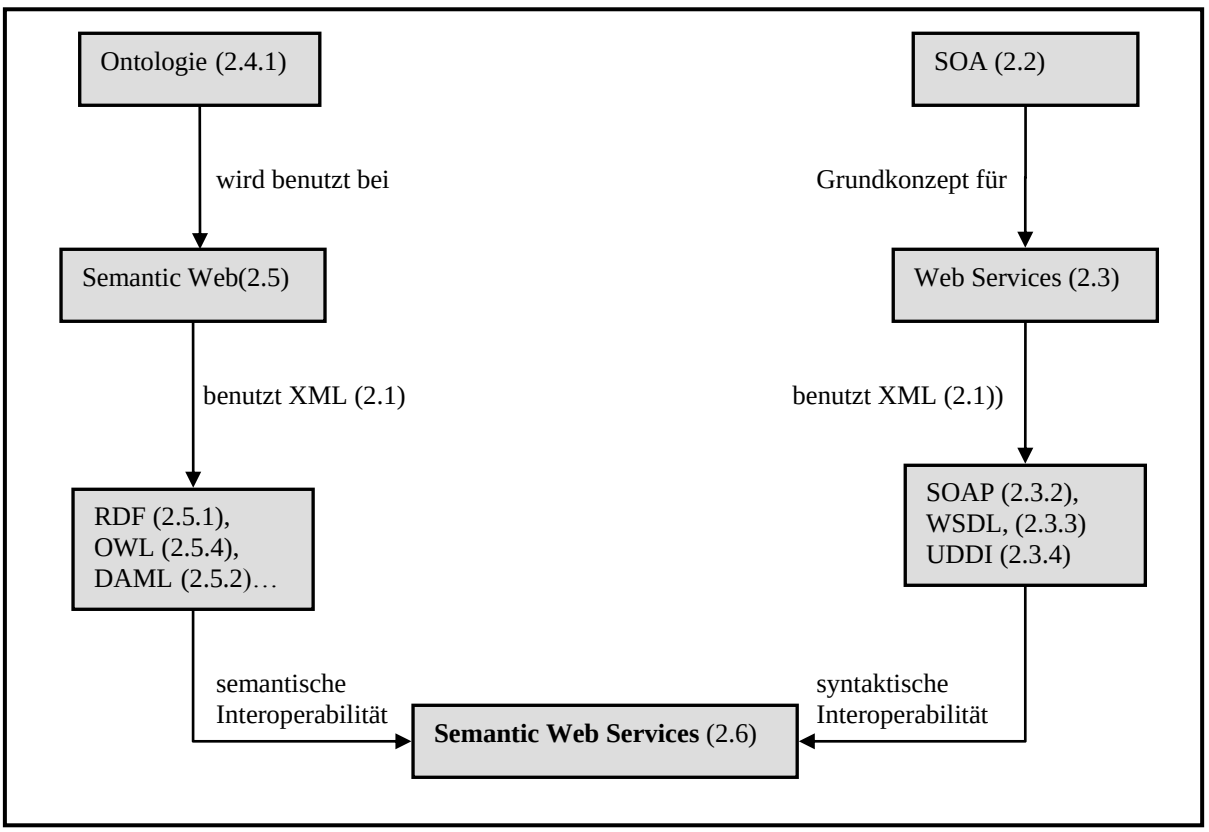

Abbildung 2.1: Kapitelüberblick (illustriert die Organisation des Kapitels; die sich im Kasten befindenden Nummern zeigen auf die zugehörigen Abschnitte).

2.1 eXtensible Markup Language

Die eXtensible Markup Language (XML) ist eine vereinfachte Untermenge von SGML[3] (Standardized General Markup Language), die aus GML (General Markup Language) abgeleitet ist. Sie ist eine Metasprache, die es ermöglicht, Daten bzw. Dokumente zu beschreiben und zu strukturieren [Von02]. Ein XML-Dokument besteht aus drei Teilen (siehe Listing 2.1):

1. Der *Prolog* ist optional und enthält z.B. die XML-Version.

2. Der *Body* enthält einen Baum aus XML-Elementen und Text. Bei den XML-Elementen gibt es genau ein Wurzelelement und mehrere Baumelemente.

3. Der *Epilog* enthält z.B. Kommentare. Er ist optional und wird so gut wie nie benutzt.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<student>
        <vorname>Hulda</vorname>
        <nachname>Ngadjeu Tchana</nachname>
        <adresse>
                <strasse>Musterstr. 1</strasse>
                <plz>99000 Hamburg</plz>
        </adresse>
</student>
```

Prolog

Body

Listing 2.1: XML-Dokument.

XML ist bekannt als ein uniformes Datenaustauschformat im Web. Sie ermöglicht die Daten zwischen verschiedenen Anwendungen in eine einheitliche Syntax zu übertragen. Außerdem kann sie auch wegen ihrer Fähigkeit Daten hierarchisch zu strukturieren in vielen Bereichen im Web eingesetzt werden.

2.2 Service-orientierte Architektur

Die Service-orientierte Architektur (SOA) ist ein Konzept, das das Anbieten, die Suche und das Finden von Diensten in einem Netzwerk ermöglichen soll [DJM+05]. Sie basiert auf der Interaktion zwischen drei Komponenten (siehe Abbildung 2.2):

1. Der *Dienstanbieter* implementiert einen Dienst und stellt diesen im Dienstverzeichnis zur Verfügung.

2. Das *Dienstverzeichnis* ist ein Verzeichnis, wo alle Dienste veröffentlicht werden. Es dient der Auffindung von Diensten durch einen Dienstnutzer.

3. Der *Dienstnutzer* sucht im Dienstverzeichnis nach einem Dienst und erhält die benötigten Informationen, um den Dienst beim Dienstanbieter abfragen zu können.

[3] http://xml.coverpages.org//sgml.html. Jul. 2008.

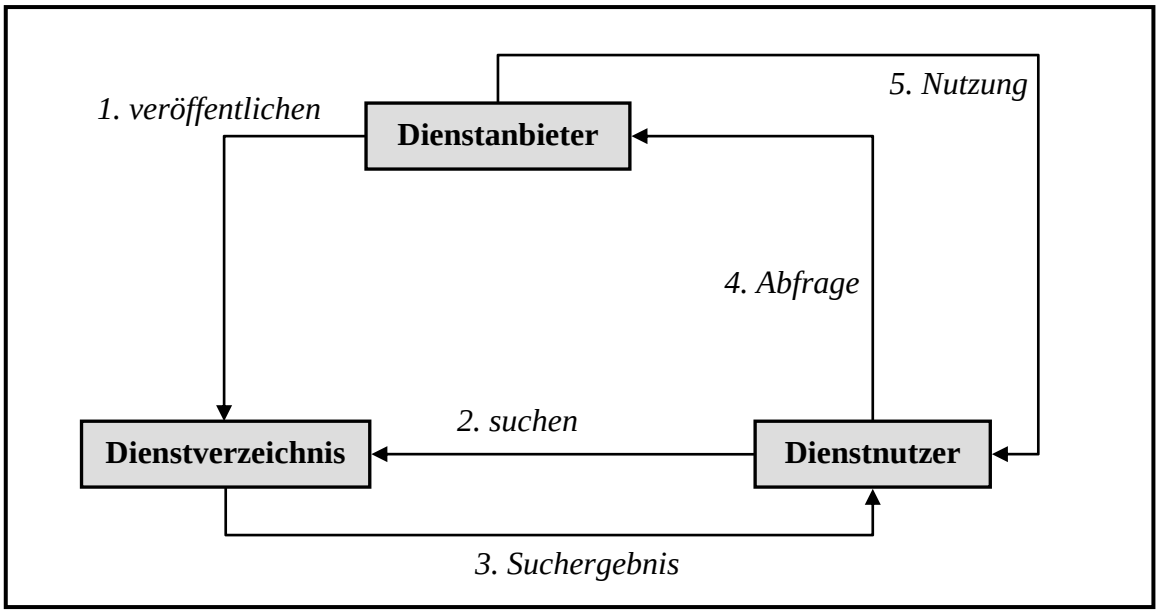

Abbildung 2.2: Das magische Dreieck einer SOA nach [DJM+05, Abb. 2.2].

Um die vorgestellten Komponenten zu realisieren, müssen einige Merkmale eingehalten werden. Diese Merkmale sind: lose Kopplung und Verzeichnisdienst.

Lose Kopplung
Die lose Kopplung ermöglicht es, ein Programm zu ändern, ohne dass es Auswirkungen auf ein anderes Programm hat. So wird die Wartung vereinfacht. Die lose Kopplung stellt in SOA sicher, dass die Abhängigkeiten zwischen Dienstanbietern und Dienstnutzern reduziert werden.

Verzeichnisdienst
Es gibt einen Verzeichnisdienst, in dem die zur Verfügung stehenden Dienste registriert werden. Die von den Anwendungen benötigten Methoden können in diesem Verzeichnis gesucht und gefunden werden.

Bei der Verwendung einer SOA müssen folgende Voraussetzungen gewährleistet sein [DJM+05]:

- *Einfachheit*. Das System soll so einfach wie möglich gehalten werden, da dieses wichtig für den späteren Dienstnutzer ist.

- *Sicherheit.* In diesem Fall versteht man unter Sicherheit die Vertraulichkeit der Daten. Das bedeutet, dass nur der gewünschte Empfänger die Daten lesen kann.

- *Standards.* Diese sind für eine SOA notwendig, damit der Nutzer den Dienst eines unbekannten Anbieters auch verstehen kann. Einige Standards werden in Rahmen von Web Services (2.3) vorgestellt.

2.3 Web Services

Web Services (WS) ist eine nutzbare Umsetzung der SOA. Zuerst soll die nachfolgende Definition von W3C[4] einen Überblick über WS verschaffen.

[4]http://www.w3.org/TR/wsa-reqs/. Nov. 2007.

„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 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.“ [DJM+05].

Nach dieser Definition verfügt Web Services über folgenden Eigenschaften:

- Web Services basieren auf XML (2.1).
- Ein Webdienst wird durch eine eindeutige Adresse (URI[5]) identifiziert.
- Die Schnittstellen eines Web Services werden mittels WSDL (2.3.3) beschrieben.
- Nachrichten werden mittels standardisierter Protokolle wie SOAP (2.3.2) und Internetprotokollen wie HTTP versendet.

Anschließend werden in den folgenden Abschnitten die Architektur und die grundlegenden Standards von Web Services behandelt.

2.3.1 Web-Services-Architektur

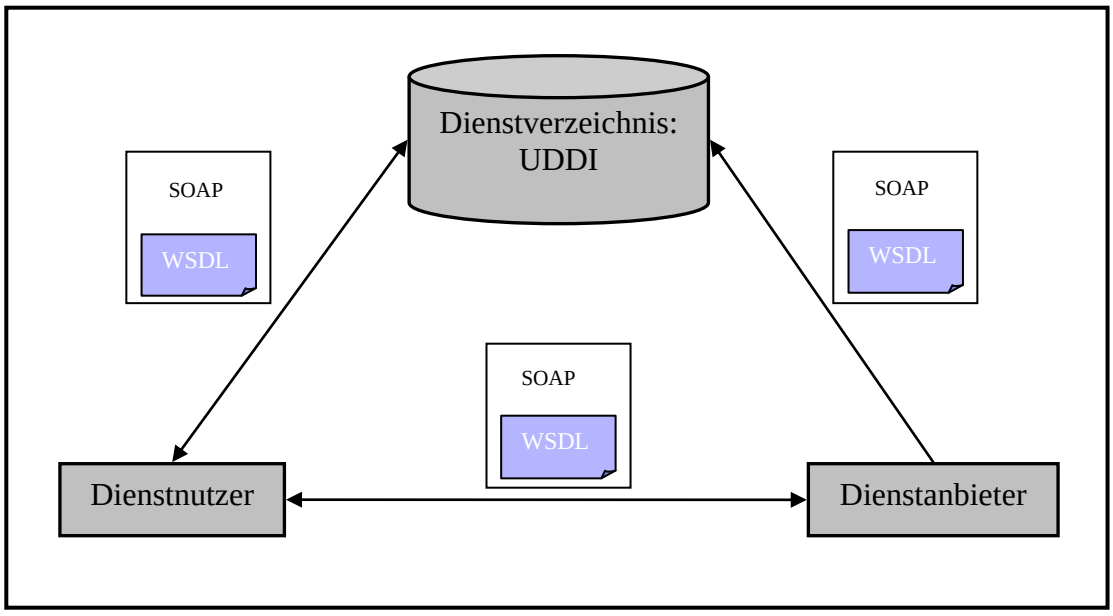

Abbildung 2.3: Web Service Architektur.

Die in SOA (siehe Abschnitt 2.2) vorgestellte Interaktion zwischen Dienstanbieter, -nutzer und -verzeichnis kann durch die Verwendung der grundlegenden Komponenten von WS konkretisiert werden (Abbildung 2.3) [DJM+05].

Die Abbildung 2.3 zeigt die in einem Anwendungsszenario von Web Services beteiligten Rollen und deren Interaktion mit den zugehörigen Protokollen. Der Dienstanbieter beschreibt die Eigenschaft seines Dienstes mittels WSDL (Abschnitt 2.3.3). Diese Beschreibungen (Web Service

[5] Uniform Resource Identifier, http://en.wikipedia.org/wiki/Uniform_Resource_Identifier. Nov. 2007.

Description, WSDs genannt) werden in einem Dienstverzeichnis (UDDI, Abschnitt 2.3.4) publiziert. Anschließend kann der Dienstnutzer im Dienstverzeichnis nach einem passenden Service suchen. War die Suche erfolgreich, gibt das Dienstverzeichnis die gefundenen WSDs zurück. Mithilfe dieser WSDs kann der Dienstnutzer mit dem Service mittels des SOAP-Protokolls (Abschnitt 2.3.2) kommunizieren.

2.3.2 Simple Object Access Protocol

Das Simple Object Access Protocol (SOAP[6]) ist ein Netzwerkprotokoll, das dazu dient Informationen zwischen verschiedenen Anwendungen auszutauschen. Es basiert auf XML, daher besteht eine SOAP-Nachricht aus einem XML-Dokument (wird SOAP-Envelope genannt). Der Envelope ist das Wurzelelement, welches die Elemente Header (optional) und Body (obligatorisch) enthält.
Zum Transport von SOAP-Nachrichten kann z.B. das Transferprotokoll HTTP[7] verwendet werden. In [DJM+05] zum Beispiel sind nähere Informationen über SOAP zu finden.

2.3.3 Web Services Description Language

Die Web Services Description Language (WSDL[8]) ist eine XML-basierte Sprache zur Beschreibung von Web Services. Sie hat sich als Standard für diese Beschreibung durchgesetzt.
Die WSD (siehe Abschnitt 2.3.1) ist hierarchisch aufgebaut und besteht nach [WBF+04] aus zwei Teilen: einem abstrakten Service-Interface-Definition- und einem konkreten Service-Implementation-Teil.

1. ***Der abstrakte Teil***, bestehend aus den Elementen *types*, *message*, *portType*, beschreibt die Schnittstellen eines Webdienstes, die in der Client-Programmierung verwendet werden. Seine Elemente werden nach [HaL04] wie folgt beschrieben:

   - *types* ist der Container für Datentyp-Definitionen, die für den Nachrichtenaustausch zwischen dem Nutzer und dem Dienst erforderlich sind.

   - *message* beschreibt abstrakt die Nachrichten, die zwischen dem Dienst und dem Nutzer ausgetauscht werden können. In einer WSD kann dieses Element mehrfach vorkommen.

   - *portType* kann beliebig viele Unterelemente (*operation*) beinhalten, wobei das *operation*-Element immer aus einem input- und einem output-Element besteht. Das *operation*-Element kann zusätzlich ein fault-Element enthalten.

2. ***Der konkrete Teil***, bestehend aus *binding* und *service*, bindet den Webdienst an ein Transportprotokoll und definiert den Ort der Implementierung des Webdienstes. Seine Elemente werden wie folgt beschrieben:

   - *binding* legt fest, welches konkrete Protokoll und Datenformat für ein bestimmtes portType zu verwenden ist.

[6] http://www.w3.org/TR/soap/. Feb 2008.
[7] http://www.w3.org/Protocols/. Feb 2008.
[8] http://www.w3.org/TR/wsdl. Feb 2008.

- *service* ist eine Kollektion von *port*-Elementen, wobei ein port die Adresse für eine Bindung ist. Das *service*-Element legt fest, an welcher Netzwerkadresse der Nutzer sich anmelden muss, um mit dem Webdienst zu kommunizieren.

Beide Teile sind Bestandteil des *definition*-Elements einer WSD. Das *definition*-Element ist das Wurzelelement. Es enthält den Namen des Dienstes und die verwendete Namensbereiche.

Das Listing 2.2 stellt die Beschreibung eines Dienstes unter Nutzung von WSDL-Elementen dar. Der Dienstanbieter (*http://example.com/*) und der Name des Dienstes (*stockquote.wsdl*) werden durch die URI: *http://example.com/stockquote.wsdl* gekennzeichnet. Die Nachrichten werden im *portType*-Element durch die In- und Outputs beschrieben.

```
<definitions name="StockQuote"
          targetNamespace="http://example.com/stockquote.wsdl"
          …

    <types>
        <schema targetNamespace="http://example.com/stockquote.xsd" …/>*
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
          <input message="tns:GetLastTradePriceInput"/>
          <output message="tns:GetLastTradePriceOutput"/>?
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding"
type="tns:StockQuotePortType">
    </binding>

    <service name="StockQuoteService">*
    </service>

 </definitions>
```

Listing 2.2: WSDL-Dokument.

2.3.4 Universal Description Discovery Integration

Die Universal Description Discovery Integration (UDDI[9]) ist ein Dienstverzeichnis, welches es ermöglicht, Dienste für Dienstnutzer zu finden und für Dienstanbieter zu publizieren. UDDI wurde ursprünglich von IBM, Microsoft und Ariba entwickelt und hat sich heutzutage als Standard für

[9] http://www.uddi.org/pubs/uddi_v3.htm. Jul. 2008.

Webdienst-Verzeichnisse durchgesetzt. Das UDDI-Verzeichnis besteht nach [Mel06] aus vier Komponenten: White, Yellow und Green pages und Service Type Registration (STR).

1. *White Pages* enthalten die grundlegenden Informationen wie Kontaktdaten über den Anbieter eines Dienstes.

2. *Yellow Pages* klassifizieren die veröffentlichten Diensten sowie ihren Anbieter.

3. *Green Pages* enthalten alle technischen Informationen eines Dienstes.

4. *Service Type Registration (STR)* ist im Prinzip den Green Pages ähnlich. Der Unterschied ist, dass die Green Pages primär für den Menschen gedacht sind, während die Dienste in STR in maschinenlesbarer Form stehen.

Jeder zu publizierende Dienst wird in UDDI unter einer UDDI-Datenstruktur, welche auf XML basiert, abgelegt. Diese besteht aus vier Elementen: businessEntity, businessService, bindingTemplate und tModel [DJM+05].

1. Die *businessEntity* wird benutzt, um White Pages zu beschreiben. Sie enthält relevante Informationen über den Dienstanbieter: Namen, Kontaktdaten der Ansprechpartner und eine Beschreibung.

2. Der *businessService*, der die Yellow Pages unterstützt, enthält Informationen über die Funktionalität eines Dienstes. Er kann mehrere bindingTemplates enthalten.

3. Die *bindingTemplate*, die von Green Pages verwendet wird, enthält die technische Information (z.B die Angabe der Endpunkte über eine URL), wie auf den Dienst zugegriffen werden muss.

4. Das *tModel* ist an ein oder mehrere bindingTemplates gebunden und präsentiert eine Reihe von detaillierten Informationen über die bindingTemplates.

2.4 Wissensrepräsentation

In Bereich der Informatik versteht man unter Wissensrepräsentation die Darstellung der Informationen (Wissen) in einer maschinenlesbaren Form [Sch06]. Im Zusammenhang mit Semantic Web (2.5) hat sich die Ontologie zur Definition von Metainformationen hervorgetan. In diesem Abschnitt wird die Ontologie vorgestellt und erläutert, welche logischen Formalismen zu ihrer Repräsentation herangezogen werden.

2.4.1 Ontologie

In diesem Abschnitt wird der Begriff „Ontologie“ genauer dargestellt. Einführend wird im Abschnitt 2.4.1.1 die Ontologie definiert. Anschließend werden im Abschnitt 2.4.1.2 die verschiedenen Kategorien einer Ontologie vorgestellt.

2.4.1.1 Definition der Ontologie

Die Ontologie (von den griechischen Worten „on“ und „logos“, die „seiend“ bzw. „Lehre“ bedeuten) ist in seiner originalen Bedeutung in der Philosophie die Lehre vom Sein.
In den letzten Jahren spielt die Ontologie in der Informatik eine große Rolle für die semantische Verarbeitung von Informationen (Semantic Web). In der Literatur gibt es eine Menge von Definitionen für Ontologien. Hier werden die zwei markantesten davon aufgezeigt.

1. *Definition von Tom Gruber*
   “An ontology is an explicit specification of a conceptualization”. Diese kurze Definition erlaubt eine prägnante Beschreibung einer Ontologie und ist in der Literatur am meisten zu finden. Sie definiert die Ontologie als eine explizite Spezifikation einer Konzeptualisierung [Gru93]. Die Konzeptualisierung ist eine abstrakte, vereinfachte Sicht auf die Welt, die wir vertreten wollen[10].

2. *Definition von Neches und Kollegen*
   “An ontology defines the basic terms and relations comprising the vocabulary of topic area as well as the rules for combining terms and relations to define extensions to the vocabulary.” Diese Definition erwähnt die Bestandteile einer Ontologie. Diese sind: die grundlegenden Begriffe, die Beziehungen zwischen diesen Begriffen und die Regeln für die Benutzung der Begriffe oder Konzepte [NFF+91].

2.4.1.2 Ontologietypen

Für die Klassifizierung von Ontologien gibt es mehrere Ansätze [GFC04]. Einer wird hier dargestellt, und zwar der Guarino-Ansatz. Dieses Klassifizierungssystem ist das prominenteste und unterscheidet vier Ontologietypen:

- *Top-level ontology*: beschreibt sehr generelle Konzepte, die nicht von einer bestimmten Domäne abhängen.

- *Domain ontology*: bietet ein grundlegendes Vokabular von Konzepten und deren Beziehungen bezogen auf eine Domäne. Die Konzepte und Beziehungen einer Top-level-ontology werden hier spezialisiert.

- *Task ontology*: beschreibt ein grundlegendes Vokabular bezogen auf eine allgemeine Aufgabe oder Aktivität. Die Beschreibung spezialisiert die Konzepte einer geeigneten Top-level-ontology.

- *Application ontology*: verfügt über alle spezifische Definitionen, die notwendig sind, um eine bestimmte Domäne zu modellieren.

[10] http://foldoc.org/. Jul. 2008.

2.4.2 Aussagenlogik

Die Aussagenlogik ist die elementare Form der Logik. Gegenstände der Aussagenlogik sind in sich geschlossene Aussagen, die als wahr oder falsch bewertet werden können [GoJ90]. Ein Aussagenbeispiel wäre: „Die Sonne scheint."

2.4.3 Prädikatenlogik

Die Prädikatenlogik erster Ordnung (auch PL1 genannt), die eine Erweiterung der Aussagenlogik ist, ist der am gründlichsten untersuchte Formalismus zur Repräsentation von Wissen. Die Prädikaten- und die Aussagenlogik gehen davon aus, dass die Welt aus Objekten, die bestimmte Eigenschaften enthalten, besteht. Es kann eine Beziehung zwischen diesen Objekte geben. Dies wird wie in natürlichen Sprachen in einer Tripelform (Subjekt, Prädikat, Objekt) dargestellt. PL1 lässt sich über Variablen, Quantoren (z.B. $\forall$) und Konnektive (z.B. ¬, →↔) formulieren [Fre04]. Ein mögliches Beispiel für PL1 ist folgendes:

Beispiel
Aussage: Ein Autohändler verkauft ein Auto.
PL1: $\forall$ (x,y): (verkauft (x,y) → Autohändler (x) und Auto (y)). Hiermit wird die Beziehung (verkauft) zwischen Autohändler und Auto definiert.

2.4.4 Beschreibungslogik

Die Beschreibungslogik (engl. Description logic(DL genannt)), die eine Teilsprache der Prädikatenlogik ist, wurde als ein Formalismus zur Wissensrepräsentation und –verarbeitung entwickelt. Sie beschreibt das Wissen unter Nutzung von Konzepten, Rollen und Konstanzen.
Eine auf einer Beschreibungslogik aufgebaute Wissensbasis besteht aus einer so genannten T-Box und einer A-Box [DLN+96]:

- *T-Box (terminological box):* beinhaltet das terminologische Wissen (Definition von Konzepten und Rollen). Eine Beispielaussage ist „Autos sind Fahrzeuge".

- *A-Box (assertional box):* spezifiziert die Konzepte der T-Box mittels Instanzen, z.B. „Audi ist ein Auto".

2.4.5 Logische Programmierung

Die logische Programmierung basiert auf der Syntax der Prädikatenlogik (2.4.3). PROLOG ist die erste und bekannteste logische Programmierung [Sow00]. PROLOG ist eine deklarative Sprache, die sich von einer prozeduralen Sprache (z.B. Programmiersprache C) unterscheidet. Bei prozeduralen Sprachen wird vom Benutzer festgelegt, wie ein Problem zu lösen ist. Dagegen steht bei deklarativen Sprachen das Problem im Vordergrund [Fre03].

2.5 Semantic Web

“The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.” [BHL01].

In diesem Zitat von Berners-Lee, Hendler und Lassila, das 2001 erschienen ist, wird das Konzept des Semantic Webs[11] in wenigen Worten zusammengefasst. Dabei soll das Semantic Web ermöglichen, dass die sich im Web befindende Datenmenge auch für Maschinen (Rechner) interpretierbar wird.

Der Mensch hat die kognitive Fähigkeit Informationen, die auf Webseiten veröffentlicht sind, zu interpretieren. Der Computer dagegen hat diese Fähigkeit nicht. Die Idee des Semantic Webs ist nun, diese Informationen so zu formulieren, dass sie vom Computer eindeutig interpretiert und weiterverarbeitet werden können. Das Semantic Web bedient sich hierzu der grundlegenden Repräsentationssprache des Internets. Die Informationen werden dabei in Form von Ontologien erfasst. Wie sich diese Ontologie in das Schichtenmodell des Semantic Webs integriert, wird in der Abbildung 2.4 gezeigt.

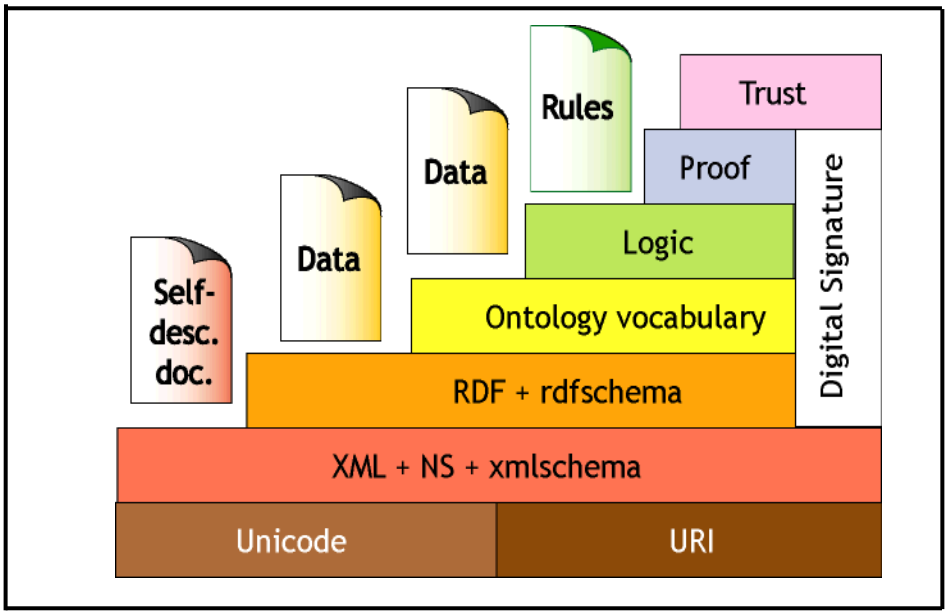

Abbildung 2.4: Semantic-Web-Schichtenmodell aus [BCH+01].

Im Folgenden werden die verschiedenen Techniken, die in der Abbildung dargestellt sind, beschrieben:

- Die *Unicode*[12] *und URI-Schicht*. Der Unicode ist ein Standard für Computerdarstellung von Zeichen und der URI (Uniform Resource Identifier) dient dazu Ressourcen im Web eindeutig zu identifizieren.

- Die *XML-Schicht* ermöglicht wie im Abschnitt 2.1 erwähnt eine strukturierte Darstellung von Dokumenten.

- Die *RDF und RDF-S Schicht* (siehe Abschnitt 2.5.1).

[11] http://www.w3.org/2001/sw/. Feb 2008.
[12] http://www.unicode.org. Juli 2008.

- Die *Ontologieschicht* ermöglicht die Definition und die Darstellung der Beziehungen zwischen Ontologiekonzepten.

- Die *Logik-Schicht* kontrolliert, dass die Daten nur in maschinenverarbeitbarer Form abgelegt werden können.

- Die *Proof-Schicht* soll durch eine logische Beweisführung die Konsistenz der Daten sicherstellen.

- Die *Trust-Schicht* soll die Richtigkeit der Daten überprüfen.

- Die *Digital Signature* wird zur Authentifizierung des Absenders der Nachricht oder des Dokuments verwendet.

Die Ontologiesprachen des Semantic Webs werden in den Abschnitten 2.5.1, 2.5.2, 2.5.3 und 2.5.4 vorgestellt.

2.5.1 Resource Description Framework

Das Resource Description Framework (RDF) ist eine Sprache, die vom W3C entwickelt wurde, um Metadaten[13] zur Beschreibung von Web-Ressourcen zu erzeugen [LaS99].
Das RDF-Modell ist einfach gehalten und besteht aus drei Objekttypen: Ressourcen[14], Eigenschaften[15] und Aussagen. Eine Aussage besteht aus einem Tripel (Subjekt, Prädikat, Objekt).

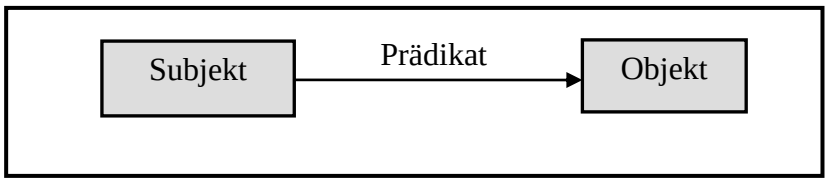

Abbildung 2.5: RDF-Modell.

Das Subjekt ist die zu beschreibende Ressource, das Prädikat ist eine Eigenschaft dieser Ressource und das Objekt dient als ein Wert der Eigenschaft [SGA07]. Hier folgt ein Beispiel:

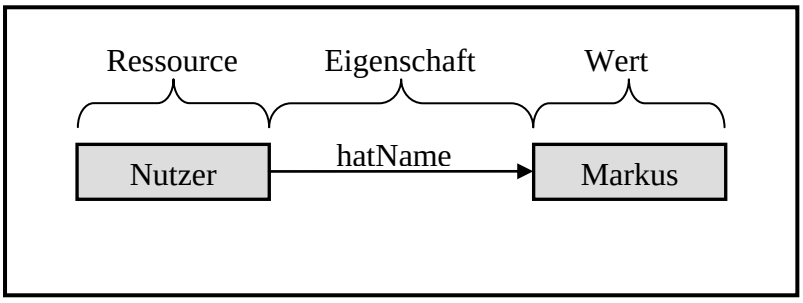

Abbildung 2.6: RDF-Beispiel.

[13] Daten, die Informationen über andere Daten beinhalten.
[14] Dies ist ein Objekt der Welt, über das eine Aussage gemacht werden soll. Bsp.: Autoren, Bücher,…
[15] Diese beschreiben eine Beziehung zwischen zwei Ressourcen.

Das RDF-Schema (RDF-S[16]) ist eine Erweiterung von RDF und definiert auf Basis von RDF spezielle Typen von Ressourcen und Eigenschaftselementen. RDF-S stellt Basisstrukturen für Klassen und deren Eigenschaften zur Verfügung.

2.5.2 DARPA Agent Markup Language

Die DARPA Agent Markup Language (DAML[17]) wurde im Rahmen eines Forschungsprogramms der Defense Advanced Research Projects Agency in Jahr 2000 entwickelt. DAML ist eine formale Sprache, die benutzt wird, um die Integration der Ontologie im Semantic Web zu erleichtern.

2.5.3 Ontology Inference Layer

Ontology Inference Layer (OIL[18]) ist als eine Erweiterung von RDF gebaut worden, mit dem Ziel webbasierendes Wissen effizient und verständlich aufzubereiten. Weiterführende Informationen finden sich in [HFB+00]. OIL wird jetzt durch DAML+OIL[19] ersetzt.

2.5.4 Web Ontology Language

Wie RDF (2.5.1) wurde die Web Ontology Language (OWL) vom W3C entwickelt. OWL ist eine Standardsprache zur Repräsentation von Informationen im Semantic Web. Sie stammt von DAML+OIL und besteht aus drei Sprachebenen (OWL Lite, OWL DL, OWL Full) mit ansteigender Ausdrucksmächtigkeit.

1. *OWL Lite* ist die Version mit der geringsten Ausdrucksmächtigkeit.

2. *OWL DL* besitzt maximale Ausdrucksmächtigkeit, ist eine Subsprache von OWL Full, deren Sprachkonstrukte teilweise im Gebrauch eingeschränkt sind.

3. *OWL Full* ist eine Obermenge von RDF und hat maximale Ausdrucksmächtigkeit. Sie besteht aus denselben Sprachkonzepten wie OWL DL, aber ohne Einschränkung.

2.6 Semantic Web Services

Die Beschreibung eines Dienstes (syntaktisch) kann von einem Mensch erkannt werden, aber eine Maschinenverarbeitbarkeit der Daten ist nicht möglich. Semantic Web Services löst dieses Problem durch die semantische Beschreibung der Daten.
Die Vision von Semantic Web Services (SWS) kann als eine Kombination von Web Services (2.3) und den beschriebenen Technologien des Semantic Webs (2.5) dargelegt werden (Abbildung 2.7).

[16] http://www.w3.org/TR/rdf-schema/. Feb.2008.
[17] http://www.daml.org/. Feb.2008.
[18] http://www.ontoknowledge.org/oil/. Feb.2008.
[19] http://www.w3.org/TR/daml+oil-reference. Feb.2008.

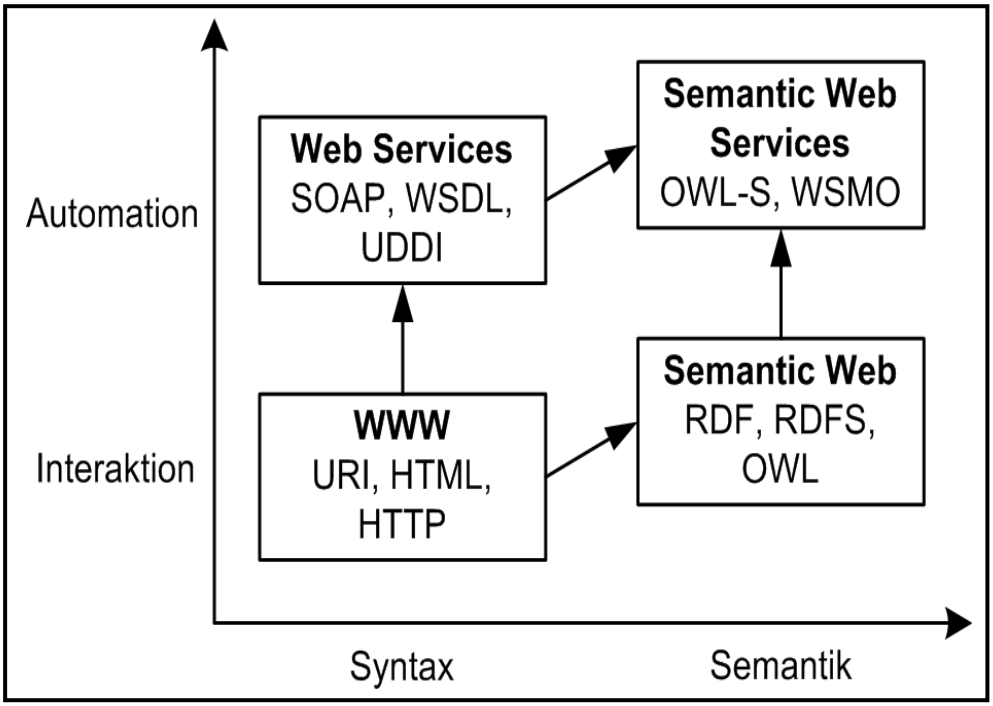

Abbildung 2.7: Semantic Web Services nach [DoJ04].

SWS als eine Erweiterung von Web Services ermöglicht das automatische Auffinden und Ausführen von Web Services, indem die semantische Annotation in der Dienstbeschreibung verwendet wird. Der Dienst ist semantisch so beschrieben, dass ein Softwareagent (Abschnitt 2.7) in der Lage ist, Dienstinformationen zu interpretieren und zu nutzen [MSZ01].

Es existieren vier Lösungsansätze: WSMO, OWL-S, WSDL-S und SWSF, die es ermöglichen, die Vision von SWS zu realisieren. Diese Ansätze werden im Kapitel 3 vorgestellt und analysiert.

2.7 Softwareagenten

Softwareagenten (SA) werden definiert als Computerprogramme, welche anstelle seines Benutzers agieren und versuchen eine bestimmte Aufgabe zu erledigen. Sie handeln selbständig (d.h. sie erledigen ihre Arbeit ohne dass ihnen von einem Benutzer geholfen wird). Die Forschung über SA begann im Jahr 1980 als ein Feld der Künstlichen Intelligenz. Heute gehört sie zu anderen Forschungsgebieten wie Verteilte Systeme [Woo02].

Softwareagenten haben folgende Eigenschaften [WoJ95]:

Reaktivität
SA sollen in der Lage sein, ihre Umgebung wahrzunehmen und auf die Änderungen in ihrer Umgebung zu reagieren.

Autonomie
Sie sollen fähig sein, ohne Intervention von Menschen oder andere Agenten zu funktionieren.

Soziales Verhalten
Agenten sollen in der Lage sein, mit anderen Agenten über eine Agenten-Nachrichtensprache (ACL[20]) zu interagieren.

Proaktivität
Agenten ergreifen selbst die Initiative, um ein definiertes Ziel zu erreichen.

[20] http://en.wikipedia.org/wiki/Agent_Communications_Language. Aug. 2008.

3 Aktueller Stand der Forschung

Semantic Web Services stellen ein noch sehr dynamisches Forschungsgebiet dar, dessen aktueller Stand in diesem Kapitel untersucht werden soll. Die Abbildung 3.1 zeigt die verschiedenen Lösungsansätze von SWS und die jeweilige Ontologiesprache, die ihren Einsatz ermöglichen dar. Diese Ansätze werden zunächst im Abschnitt 3.1 vorgestellt und beschrieben, anschließend werden sie im Abschnitt 3.2 bewertet.

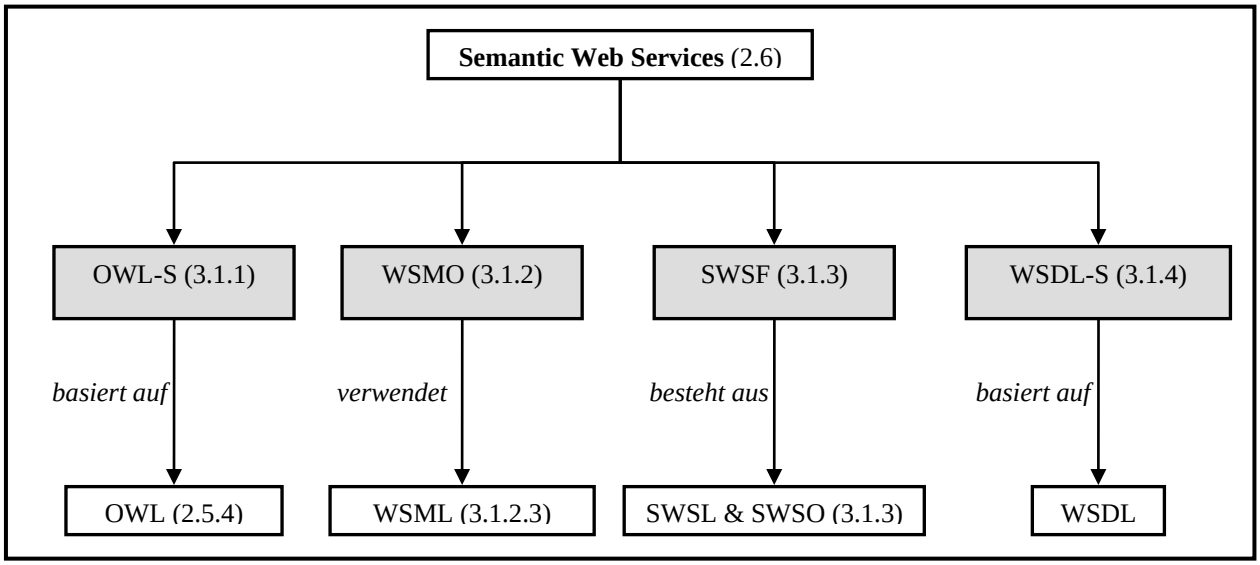

Abbildung 3.1: Organisation der Technologien (die sich im Kasten befindenden Nummern zeigen auf die zugehörigen Abschnitte).

3.1 Beschreibung der verschiedenen Lösungsansätze

Um einen besseren Überblick über die verschiedenen Technologien zu haben, werden OWL-S, WSMO, SWSF und WSDL-S jeweils in den Abschnitten 3.1.1, 3.1.2, 3.1.3 und 3.1.4 beschrieben.

3.1.1 Web Ontology Language-Services

Seit der ersten Veröffentlichung des Vorgängers im Mai 2001 unter dem Namen DAML-S[21] bis zur aktuellen Version 1.2[22], welche im März 2006 veröffentlicht wurde, hat die Web Ontology Language-Services (OWL-S) zahlreiche Verbesserungen und Weiterentwicklungen erfahren. Sie basiert auf OWL (2.5.4). OWL-S [MBH+04] besteht aus drei grundlegenden Elementen, die jeweils eine Teilontologie modellieren (siehe Abbildung 3.2):

1. Das *Service Profile* definiert, was der Dienst leistet. Dies gibt den Softwareagenten die Möglichkeit den Service zu finden und auszuwählen.

2. Das *Service Model* definiert das Ausführungsmodell eines Dienstes. Es legt also fest, wie ein Dienst arbeitet.

3. Das *Service Grounding* beschreibt, wie der Dienst verwendet wird, d.h. wie ein Kunde den Dienst aufrufen kann.

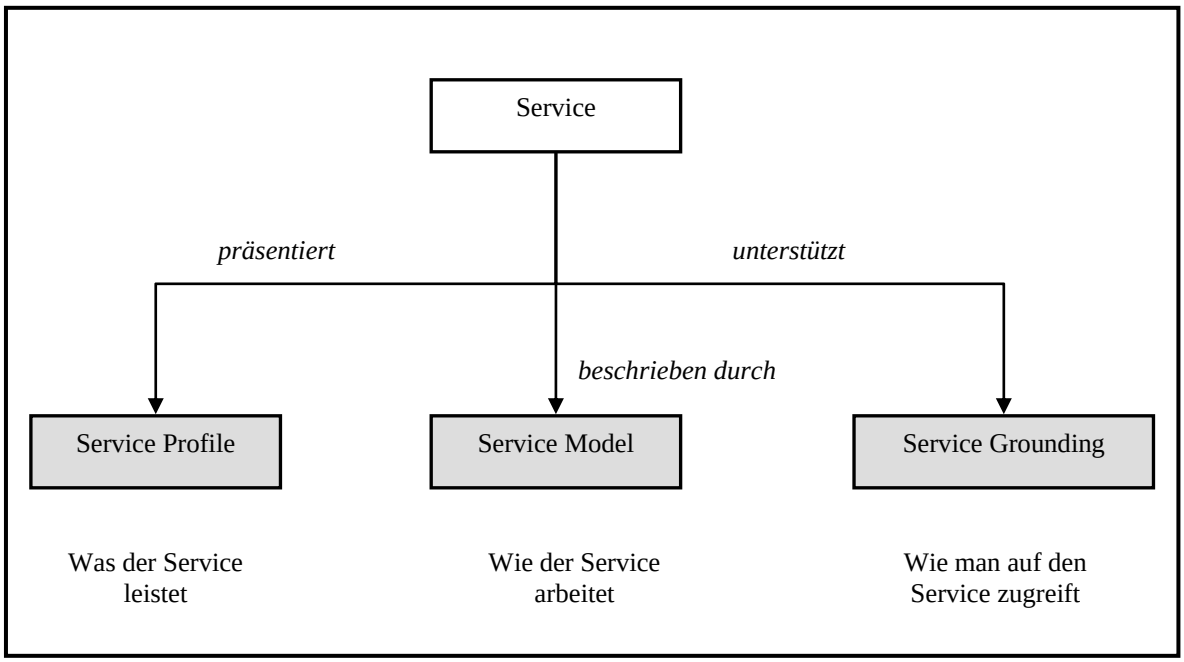

Abbildung 3.2: OWL-S Ontologien nach [MBH+04, Abb. 1].

3.1.1.1 Service Profile

Die Klasse der Profile, welche die Subklasse des Service Profile ist (siehe Abbildung 3.3), definiert funktionale und nicht-funktionale Eigenschaften.

[21] http://www.daml.org/services/daml-s/0.7/. Mär. 2008.
[22] http://www.daml.org/services/owl-s/. Mär. 2008.

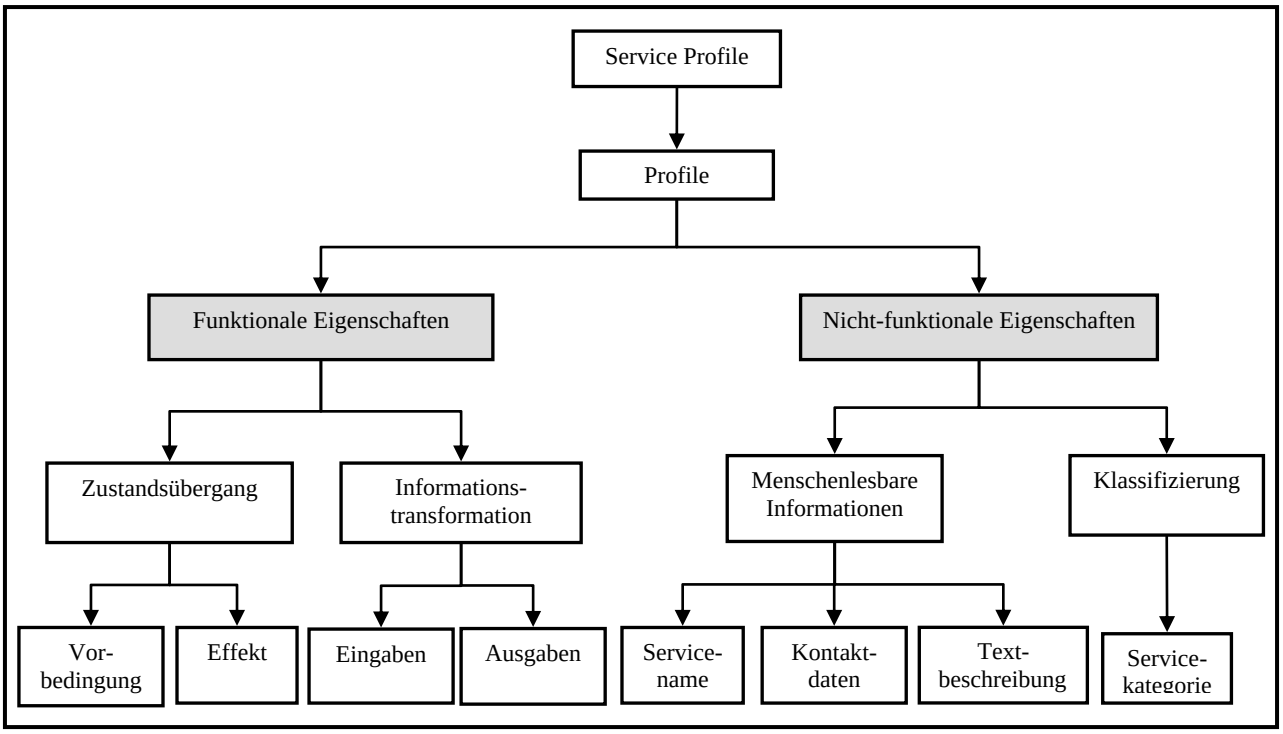

Abbildung 3.3: Service-Profile-Ontologie von OWL-S[23].

Funktionale Eigenschaften
Diese umfassen die Funktionalität des Dienstes, also was der Dienst leistet. Bei der Beschreibung des Dienstes wird zwischen Informationstransformation und Zustandsübergang unterschieden. Die Informationstransformation beinhaltet die *Eingaben* (Inputs) und die *Ausgaben* (Outputs), während der Zustandsübergang aus *Vorbedingung* (Precondition) und *Effekt* (Effect) besteht.

1. Die *Eingaben* beschreiben, welche Information der Service benötigt. Bei einem Onlinebanking wäre dies z.B. die Eingabe der Transaktionsnummer.

2. Die *Ausgaben* beschreiben, welche Information durch den Service erzeugt wird, zum Beispiel eine Bestätigung der Geldüberweisung.

3. Die *Vorbedingung* ist die Bedingung, die wahr sein muss, um den Dienst erfolgreich auszuführen, zum Beispiel die Gültigkeit der Transaktionsnummer.

4. Der *Effekt* beschreibt die Bedingung, die nach der Dienstausführung wahr ist, zum Beispiel die Transaktion ist durchgeführt.

Nicht-funktionale Eigenschaften
Diese unterteilen sich in zwei Abschnitte:

1. Die *Menschenlesbare Informationen* sind: der *Servicename*, die *Kontaktdaten* des Dienstanbieters und eine *textuelle Beschreibung*.

2. Die *Klassifizierung des Webdienstes* erfolgt mittels der *Servicekategorie*. Dadurch kann auf Kategorien wie zum Beispiel NAICS[24] und UNSPSC[25] verwiesen werden.

[23] http://www.w3.org/Submission/OWL-S/. Jul. 2008.
[24] North American Industry Classification System, http://www.naics.com/. Feb. 2008.

Die Klasse Profile spielt in der Anwendung zwei Rollen. Einerseits kann ein Profil vom Dienstanbieter verwendet werden, um seine Dienste zu publizieren. Diese Profile nennt man *Advertisements*. Der Dienstnutzer definiert andererseits mit dem Profil die Beschreibung des gesuchten Dienstes. Diesen Vorgang nennt man *Request*. Um passende Dienste zu finden, werden dann *Request* und *Advertisement* verglichen. Dieser Vorgang wird Matchmaking genannt (4.3.4).

3.1.1.2 Service Model

Ziel des Service Models ist es zu beschreiben, wie ein Service automatisch ausgeführt werden kann. Dazu wird der Ablauf des Services als ein Prozess modelliert. Ein Prozess kann mit Hilfe von Kontrollkonstrukten zwei Arten von Zwecken erfüllen. Einerseits werden die Informationen transformiert, indem die Eingaben in Ausgaben umgewandelt werden. Andererseits findet mittels Vorbedingungen und Effekten ein Zustandsübergang während der Ausführung des Prozesses statt.

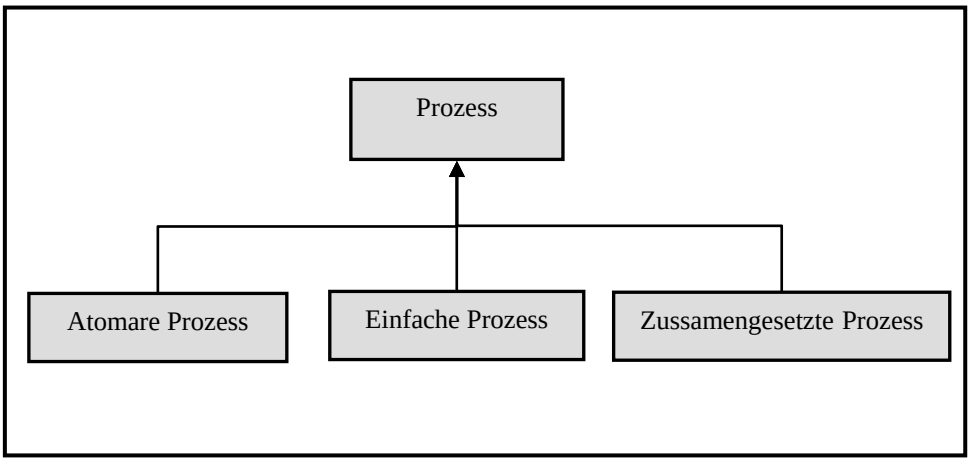

Abbildung 3.4: Prozessarten von OWL-S.

OWL-S unterscheidet drei verschiedene Arten von Prozessen (siehe Abbildung 3.4), aus denen der Ablauf des Services sich dann zusammenstellt:

1. Die *Atomaren Prozesse* werden aus Sicht des Dienstnutzers in einem einzigen Schritt ausgeführt, d.h. es findet nur ein einziger Nachrichtenaustausch zwischen Dienstnutzer und Dienstgeber statt. Sie erhalten keine weiteren Subprozesse. Jedem atomaren Prozess ist mindestens ein Grounding (Abschnitt 3.1.1.3) zugeordnet.

2. Die *Einfachen Prozesse* dienen als Intermediäre zwischen atomaren und zusammengesetzten Prozessen. Sie sind nicht ausführbar und auch mit keinem Grounding (Abschnitt 3.1.1.3) verbunden.

3. Die *Zusammengesetzten Prozesse* sind Prozesse, die aus einer definierten Abfolge von atomaren, einfachen und/oder weiteren zusammengesetzten Prozessen bestehen.

[25] United Nations Standard Products and Services Codes, http://www.unspsc.org/. Feb. 2008.

3.1.1.3 Service Grounding

Das Service Grounding liefert konkrete Details, wie auf den Dienst zugegriffen werden muss. Es spezifiziert Informationen wie die Kommunikationsprotokolle (z.B. HTTP), die Portnummer des Dienstes und das Format der Nachricht. Um die Kommunikation zwischen den einzelnen Komponenten des Netzwerks zu ermöglichen, werden die Ein- und Ausgaben der atomaren Prozesse des Service Models auf konkrete Nachrichten abgebildet.

Das Grounding definiert die Nachrichtenformate nicht selbst, sondern bedient sich bereits existierender Standards wie WSDL (2.3.3). OWL-S und WSDL haben einige korrespondierende Beschreibungselemente, die in der folgenden Tabelle dargestellt werden.

OWL-S	WSDL
Atomarer Prozess	WSDL-Operation
Input bzw. Output eines atomaren Prozesses	Parts einer Input- bzw. Outputmessage einer Operation
Die Typen (d.h. OWL-Klassen) der In- und Outputs	Erweiterbare Konzepte der abstrakten Typen

Tabelle 3.1: Beziehung zwischen OWL-S und WSDL nach [MBH+04].

3.1.2 Web Service Modeling Ontology

Um Web Service Modeling Ontology (WSMO[26]) zu beschreiben, wird zunächst eine kurze Einführung gegeben (siehe Abschnitt 3.1.2.1). Anschließend werden seine Hauptelemente dargestellt (siehe Abschnitt 3.1.2.2). Diese Hauptelemente werden anhand der WSML-Sprache (siehe Abschnitt 3.1.2.3) modelliert. Abschließend wird die Ausführungsumgebung von WSMO dargelegt (siehe Abschnitt 3.1.2.4).

3.1.2.1 Kurze Einführung

Die WSMO wurde im April 2005 als Standardvorschlag beim W3C eingereicht. Sie ist ein konzeptionelles Modell für semantische Servicebeschreibung und zugleich eine vollständige Beschreibungssprache zur Annotierung von Web Services [BBD+05]. Sie basiert auf WSMF und wurde davon abgeleitet. Ziel von WSMO ist es, Verfahren bereitzustellen, mit denen die semantische Dienstnutzung automatisiert werden kann.

Das WSMO-Projekt ist nach [CaS06] in drei Arbeitsgruppen unterteilt:

1. Die *WSMO Working Group*[27] bietet ein konzeptionelles Modell zur Beschreibung von Diensten.

2. Die *WSML Working Group*[28] definiert eine konkrete und formale Sprache für die Beschreibung von SWS in WSMO.

[26] http://www.w3.org/Submission/2005/06/. Mär. 2008
[27] http://www.wsmo.org/. Mär. 2008
[28] http://www.wsmo.org/wsml/. Mär. 2008

3. Die *WSMX Working Group*[29] definiert und bietet die Ausführungsumgebung für SWS.

3.1.2.2 WSMO-Hauptelemente

WSMO basiert auf vier Hauptelementen: Ontologies, Goals, Web Services und Mediators, wie in Abbildung 3.5 dargestellt.

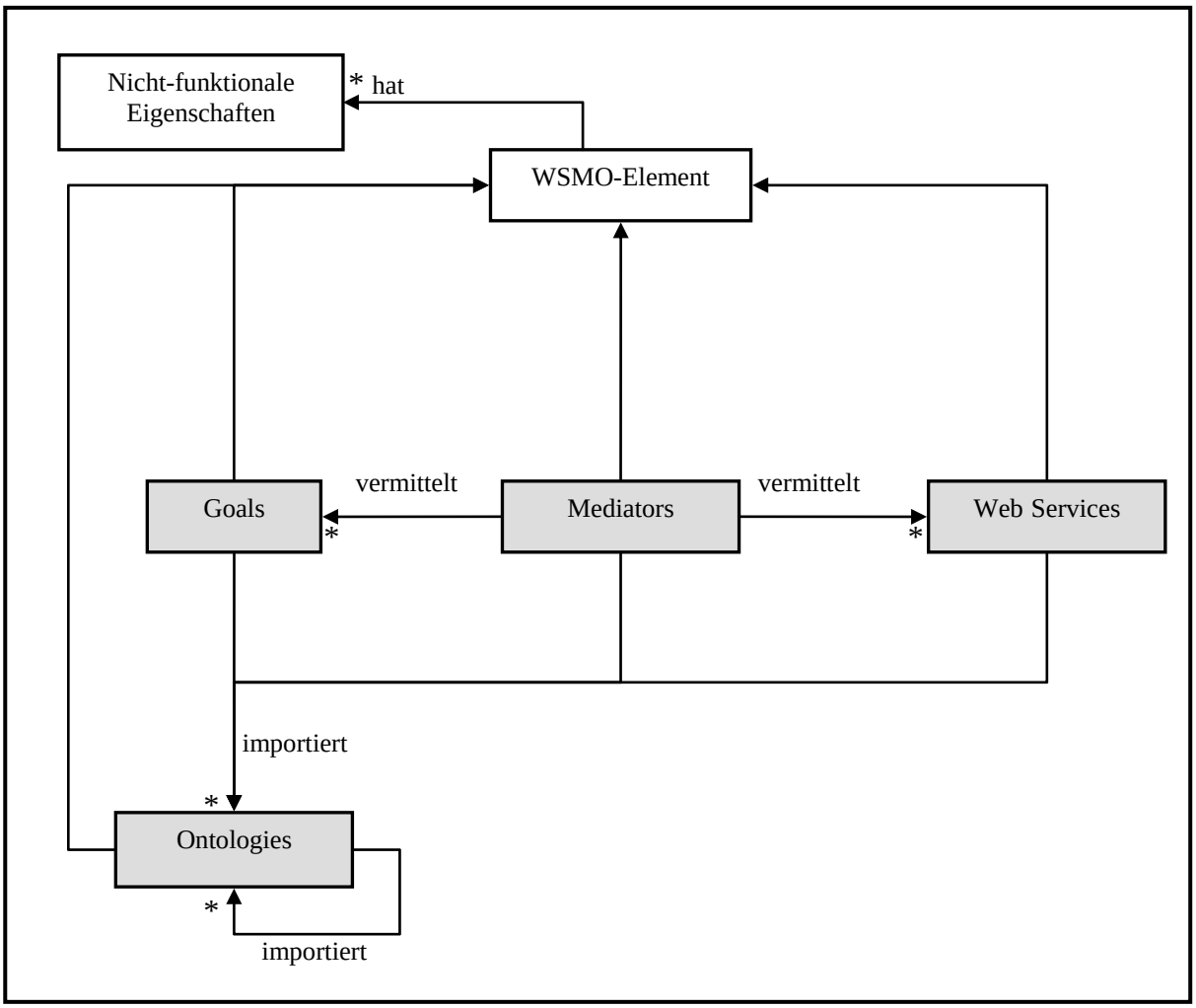

Abbildung 3.5: Die vier Hauptelemente des konzeptionellen Modells von WSMO nach [RLK05].

Zu notieren ist, dass alle WSMO-Hauptelemente (Ontologies, Goals, Web Services und Mediators) mehr oder weniger über nicht-funktionale Eigenschaften, importierte Ontologien und verwendete Mediators verfügen.

3.1.2.2.1 Ontologies

Die *Ontologies* bieten eine formale spezifizierte Terminologie, die von allen anderen Komponenten verwendet wird [StH05]. Die wesentlichen Elemente von WSMO-Ontologies sind *Konzept*, *Attribut*, *Instanz*, *Relation*, *Funktion* und *Axiom*.

[29] http://www.wsmx.org/. Mär. 2008

- In dem *Konzept*-Element findet die Speicherung des Basisvokabulars der zu modellierenden Domäne statt.
- Das Konzept-Element enthält *Attribute*.
- Ein Konzept lässt sich über eine *Instanz* instanziieren.
- Die *Relation* ist eine Beziehung zwischen mehreren Konzepten ohne einen Rückgabewert.
- Eine *Funktion* ist eine spezielle Relation mit genau einem Rückgabewert.
- Ein *Axiom* ist ein logischer Ausdruck, die in WSMO benutzt wird [CaS06].

Die nicht-funktionalen Eigenschaften dienen dazu, Eigenschaften wie Titel, Autor, Datum, Sprache und weitere Informationen wie Qualitätsinformationen (Sicherheit, Leistung, usw.), Version und Kosten zu beschreiben. Importierte Ontologien ermöglichen es, bereits definierte Ontologien in ein Projekt zu importieren. Dadurch wird die Wiederverwendung von Ontologien ermöglicht. Bei Inkompatibilität zwischen den verwendeten Ontologien werden Mediators verwendet, um das Problem zu lösen. In diesem Fall werden OO-Mediators (siehe 3.1.2.2.4) benutzt.

3.1.2.2.2 Web Services

Unter Web Services in WSMO ist die semantische Beschreibung von angebotenen Diensten zu verstehen. Diese Beschreibung besteht aus einer Capability und einem Interface (siehe Abbildung 3.6).

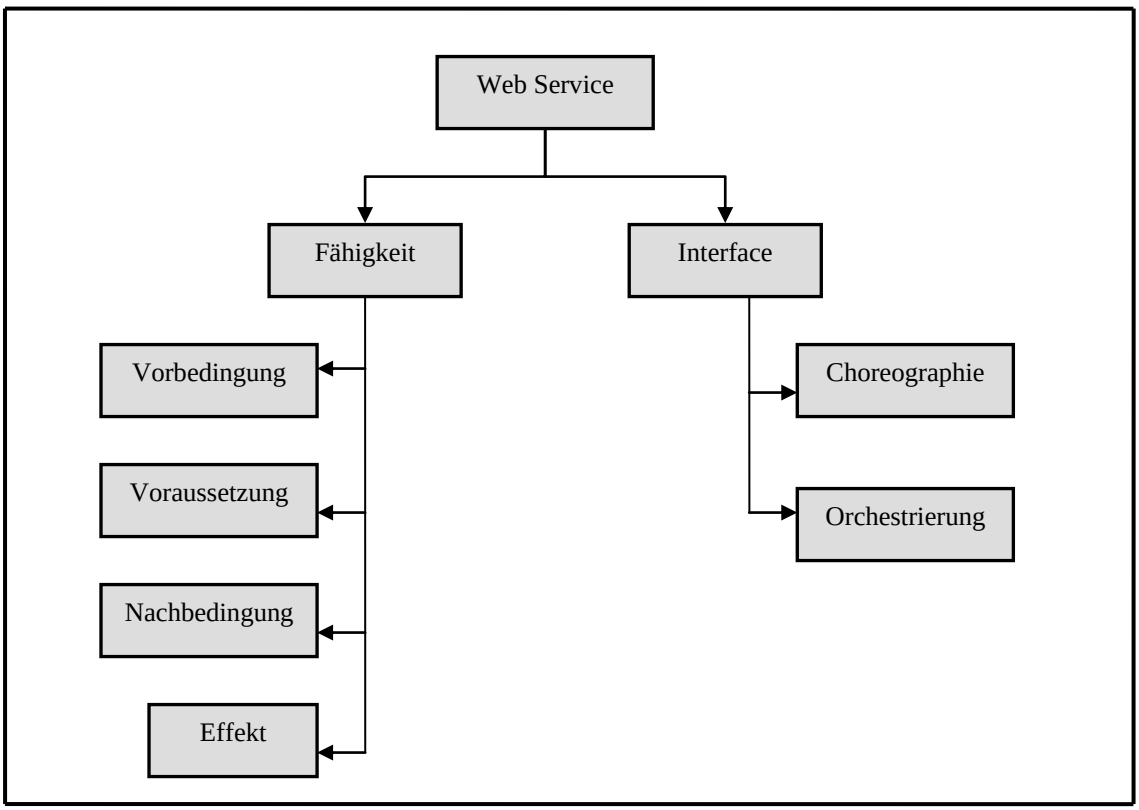

Abbildung 3.6: WSMO-Web-Service.

Eine Fähigkeit definiert die Funktionalität des Webdiensts und besteht aus *Vorbedingung*, *Voraussetzung*, *Nachbedingung* und *Effekt*. Dabei legt Vorbedingung fest, welche Information ein Webdienst erwartet. Die Voraussetzung beschreibt einen Zustand, der vor der Ausführung des Dienstes gelten soll. Nachbedingung und Effekt drücken einen Zustand aus, der nach einer erfolgreichen Ausführung des Dienstes erreicht werden soll.

Ein Interface beschreibt, wie der Webdienst verwendet werden kann. Es gibt zwei Arten davon: die *Choreographie* und die *Orchestrierung*. In der Choreographie wird festgelegt, wie der Dienstnutzer mit dem Dienst zu kommunizieren hat [CaS06]. Die Orchestrierung beschreibt, wie ein Service andere Services nutzt, um ihre Funktionalität zu erreichen.

3.1.2.2.3 Goals

Goals sind die gewünschten Funktionalitäten eines Dienstes, welche ein Dienstnutzer von einem Dienstanbieter erwartet. Sie werden genauso definiert wie WSMO-Web-Services (siehe Abschnitt 3.1.2.2.2), d.h. durch eine Fähigkeit und ein Interface.

3.1.2.2.4 Mediators

Mediators haben zum Ziel, die Heterogenitäten zwischen den verschiedenen Hauptelementen von WSMO zu beseitigen. Dazu definiert WSMO verschiedene Typen von Mediators: OO-, GG-, WW- und WG-Mediators [BBD+05]. Dabei werden die WSMO-Hauptelemente durch diese Mediators verbunden (siehe Abbildung 3.7).

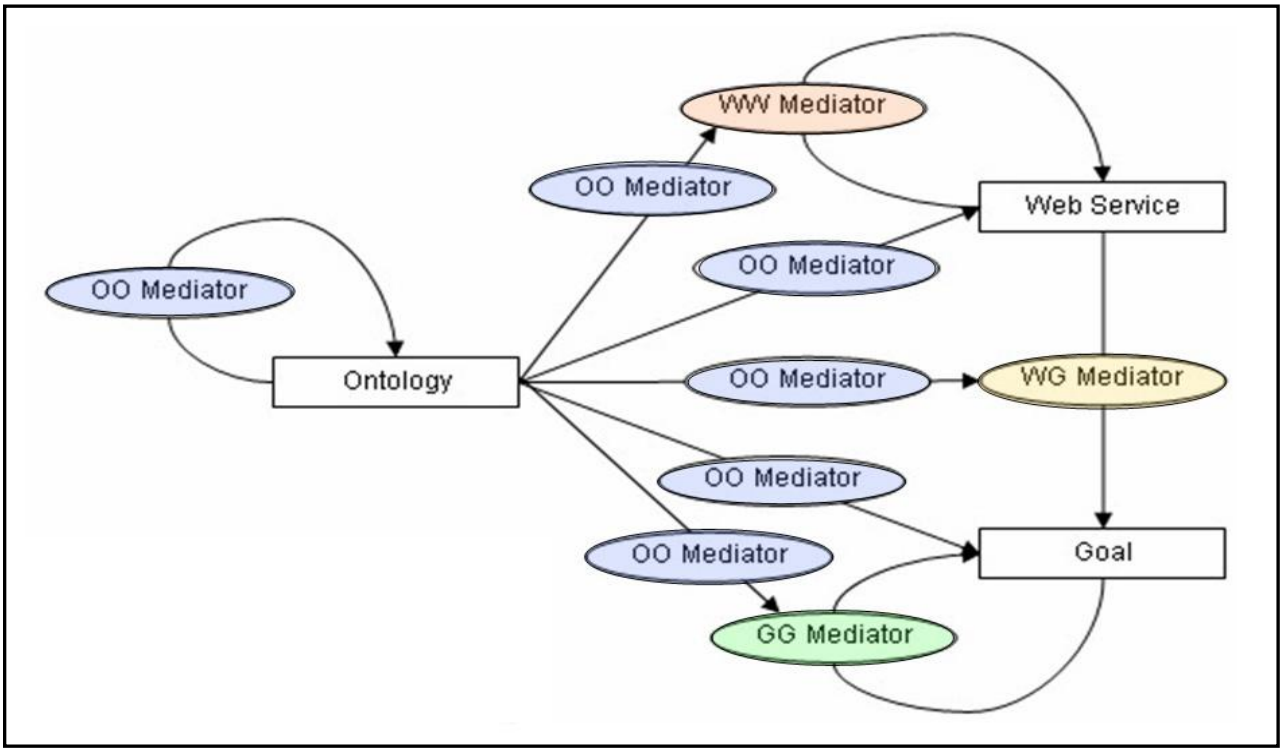

Abbildung 3.7: Kombination von Mediators und WSMO-Hauptelemente [BMR+05].

1. Der *Ontology-Ontology-Mediator (OO-Mediator)*: importiert die Ontologien und löst mögliche Unterschiede in der Repräsentation auf.

2. Der *Goal-Goal-Mediator (GG-Mediator)*: ermöglicht es, Einzelziele mit komplexeren Zielen zu verknüpfen.

3. Der *Webservice-Webservice-Mediator (WW-Mediator)*: verbindet zwei Webdienste, die nicht kompatibel sind.

4. Der *Webservice-Goal-Mediator (WG-Mediator)*: verbindet semantisch passende Webdienste mit einem Ziel.

3.1.2.3 Web Service Modeling Language

Die Web Service Modeling Language (WSML) ist eine formale Sprache, die eine formale Syntax und Semantik für WSMO anbietet. Sie basiert auf verschiedenen logischen Formalismen (insbesondere DL (Abschnitt 2.4.4), PL1 (Abschnitt 2.4.3) und LP (Abschnitt 2.4.5)), welche für die Modellierung von SWS wertvoll sind [BFK+05].
WSML besteht aus einer Reihe von fünf Varianten: WSML-Core, WSML-Full, WSML-DL, WSML-Flight und WSML-Rule.

- Der *WSML-Core* ist die grundlegende Sprache und hat die niedrigste Ausdruckskraft. Er basiert aus einer Mischung von DL und LP, wodurch er in WSML-DL bzw. in WSML-Flight und WSML-Rule erweitert werden kann.

- Die *WSML-DL* als Erweiterung von WSML-Core basiert auf DL.

- *WSML-Flight* als Erweiterung von WSML-Core bietet eine ausdrucksstärkere Sprache.

- Die *WSML-Rule* erweitert WSML-Flight mit weiteren Eigenschaften der LP.

- *WSML-Full* ist eine Mischung von WSML-DL und WSML-Rule und basiert auf FOL. Sie ist die ausdrucksstärkste von allen.

3.1.2.4 Web Service Execution Environment

Die Web Service Execution Environment (WSMX) ist eine Ausführungsumgebung für SWS, die auf dem konzeptionellen Modell von WSMO basiert. Sie hat die Hauptaufgabe Web Services zu suchen, auszuwählen, auszuführen und die Zusammenarbeit zwischen Web Services zu steuern [CMO+05].

WSMX in der Version 0.5 besteht aus folgenden Komponenten:

- *WSMX Integration API* ist eine Kollektion von Bibliotheken, die für die Integration von lose gekoppelten Komponenten zur Verfügung steht.

- *WSMX Core* ist die zentrale Komponente mit einer Reihe von Sub-Komponenten, die verschiedene Schnittstellen in der WSMX Integration API implementiert.
- *WSMX Components* stellen Funktionalitäten zur Verfügung, die in die WSMX Core integriert werden können.

- *WSMX Runtime Data Mediator* ist eine Komponente des WSMX-Systems, die als ein selbstständiger Web Service für „Runtime Data Mediation“ zur Verfügung steht.

- *Web Services Modeling Toolkit (WSMT)* stellt ein System für schnelle Erzeugung und Entwicklung von Tools für SWS zur Verfügung.

Im Folgenden werden einige Komponenten der *WSMX Components* beschrieben (Abbildung 3.8):

Der Parser
Der Parser hat die Aufgabe, WSMO-Beschreibungen, die mit einem WSMO-Editor realisiert wurden, zu validieren und im Service Repository zu speichern.

Der Ressource Manager
Der Ressource Manager verwaltet die verschiedenen Speicher von WSMO-Beschreibungen (Web Services, Goals, Ontologies und Mediators). Ein Beispielspeicher ist der Web-Service-Repository.

Data Mediator
Wenn es bei Auffindung, Komposition, Selektion und Aufruf eines Webdienstes Inkompatibilität zwischen Daten (Instanzen von Ontologiekonzepten), die vom Dienstnutzer kommen und denen vom Dienstanbieter gibt, wird der Data Mediator verwendet, um diese Heterogenitätsprobleme zu lösen[MZM+04].

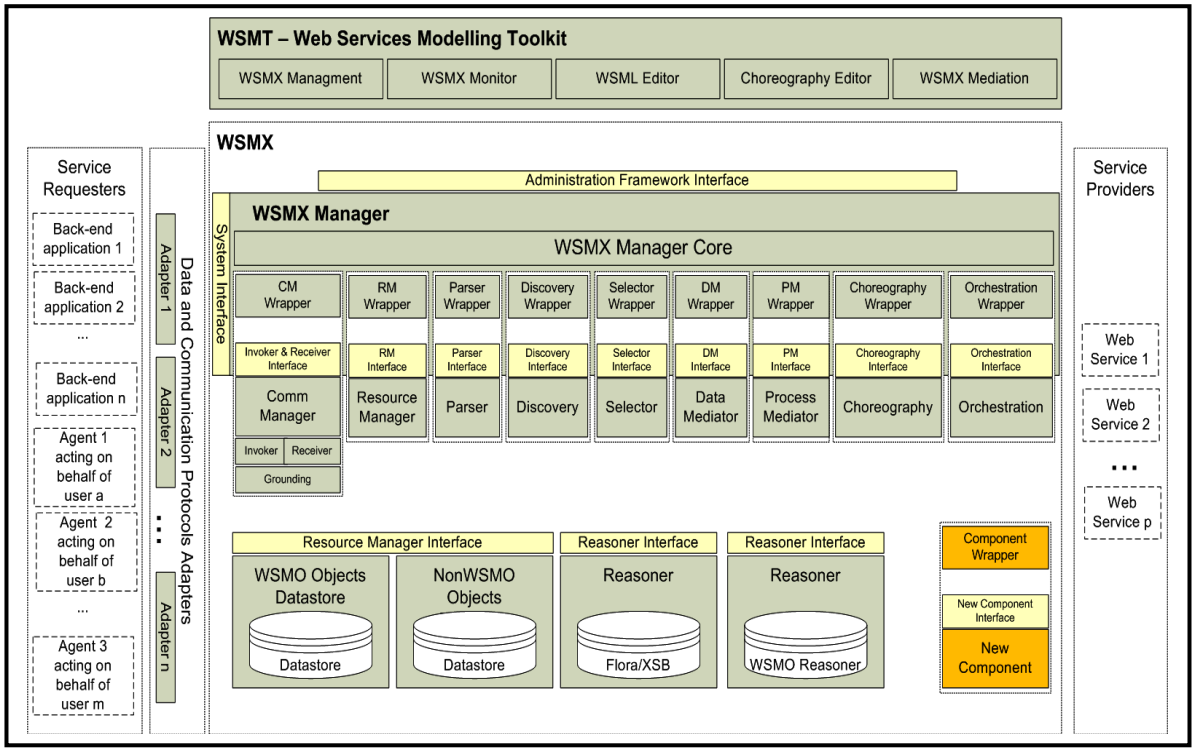

Abbildung 3.8: WSMX-Architektur [ZMH+05a].

Der Communication Manager (CM)
Der CM hat die Aufgabe, Protokolle für das Senden und Empfangen von Nachrichten zu und von WSMX zu verantworten [ZMH+05a]. Das bedeutet, dass die Nachrichten, die vom Dienstnutzer kommen, weitergeleitet werden und die Antworten, die von anderen Komponenten kommen an den Dienstnutzer geliefert werden. Der CM besteht aus zwei weiteren Subkomponenten: *Receiver*

und *Invoker*. Der Receiver sorgt dafür, dass alle empfangenen Nachrichten in WSML formuliert sind. Hingegen spielt der Invoker eine Rolle, wenn ein Webdienst aufgerufen werden soll.

Die Choreography Engine
Die Choreographie eines Webdienstes beschreibt, wie ein Dienstnutzer mit dem Dienst kommunizieren kann. Wie der Webdienst hat der Dienstnutzer sein eigenes Kommunikationsmuster. Falls beide Kommunikationsmuster unterschiedlich sind, ist keine Kommunikation zwischen beiden möglich. Deswegen hat die Choreography Engine die Aufgabe zu prüfen, dass beide Kommunikationsmuster zusammenpassen [BCF+05].

Für mehr Information über die Komponenten von WSMX (siehe [ZMH+05b]).

3.1.3 Semantic Web Services Framework

Das Semantic Web Services Framework[30] (SWSF) ist ein weiterer alternativer Standard im Bereich von SWS, welcher von der Semantic Web Services Initiative[31] (SWSI) vorangetrieben wird. Es befindet sich seit Mitte 2005 in der Version 1.0 und besteht (ähnlich wie WSML und WSMO) aus zwei Teilen: der Semantic Web Services Language (SWSL) und der darauf aufbauenden Semantic Web Services Ontology (SWSO).

Semantic Web Services Language
Die Semantic Web Services Language (SWSL) ist eine Logiksprache und besteht aus zwei Teilen: der SWSL-FOL (basierend auf Prädikatenlogik, siehe Abschnitt 2.4.3) sowie den SWSL-Rules (eine regelbasierte Sprache).

Semantic Web Services Ontology
Die Semantic Web Services Ontology (SWSO) ist die Ontologie für SWSL, mit deren Hilfe Services beschrieben werden können. Diese Ontologie hat zwei Teile: die First-order Logic Ontology for Web Services (FLOWS) und die Rules Ontology for Web Services (ROWS).

SWSO hat einige Ähnlichkeiten zu OWL-S bei der Dienstbeschreibung, die auch dreigeteilt (Service Descriptors, Process Model und Grounding) ist.

3.1.4 Web Services Description Language-Semantic

OWL-S, WSMO und SWSF definieren eine Sprache, um Webdienste zu beschreiben. Die Web Services Description Language-Semantic (WSDL-S[32]) geht einen anderen Weg. Sie ist eine Erweiterung von WSDL, die es erlaubt die Schnittstellenbeschreibung mit semantischen Konzepten zu annotieren, indem die Elemente in WSDL durch Verweise auf eine externe Ontologie mit semantischen Informationen verknüpft werden (siehe Abbildung 3.9). Im November 2005 wurde WSDL-S zur Standardisierung beim W3C eingereicht.

[30] http://www.w3.org/Submission/SWSF/. Apr. 2008.
[31] http://www.swsi.org/. Apr. 2008.
[32] http://www.w3.org/Submission/WSDL-S/. Apr. 2008.

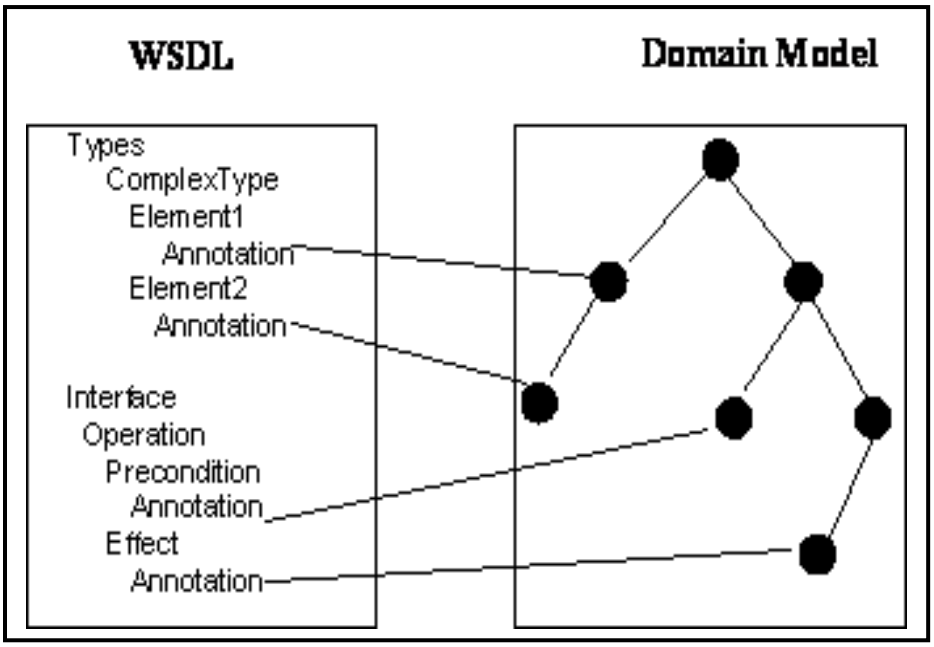

Abbildung 3.9: Verknüpfung von WSDL-Elementen mit Semantik [AFJ+05].

Die Erweiterung von WSDL betrifft konkret die Eingaben, Ausgaben und Operationen eines Webdienstes, die auf ein Domänenmodell abgebildet werden. Die Operationen bestehen aus jeweils einer *Vorbedingung* und beliebig vielen *Effekt*-Elementen. Als Mechanismus für die Umwandlung von Informationen aus Ontologien und deren Äquivalenz in WSDL werden Transformationssprachen wie XSLT verwendet.

Die Einfachheit und die große Akzeptanz der Sprache WSDL sind die wichtigsten Vorteile dieses Ansatzes gegenüber den bereits vorgestellten Ansätzen für SWS.

3.2 Analyse der Ansätze für Semantic Web Services

Im letzten Abschnitt wurden vier Ansätze zur semantischen Dienstbeschreibung erläutert: OWL-S, WSMO, SWSF und WSDL-S. Obwohl diese vier Technologien dasselbe Ziel haben, nämlich Webdienste semantisch zu beschreiben, weisen sie einige Unterschiede auf. In diesem Abschnitt geht es darum diese Ansätze miteinander zu vergleichen.
In den folgenden Tabellen soll eine Bewertung dieser vier vorgestellten Technologien erfolgen. Dabei bezeichnen die Zahlen 1 bis 4 die Erfüllungsgrade der Anforderungen (wobei 1 den höchsten und 4 den niedrigsten Grad darstellt) und die Kennzeichnung mit „-“ bedeutet, dass in diesem Fall die Anforderung nicht erfüllt wurde.

Ausdrucksmächtigkeit
Wenn eine Sprache ausdrucksstark genug ist, können existierende Dienste präzise beschrieben werden.

	WSDL-S	**OWL-S**	**WSMO**	**SWSF**
Sprache	-	OWL	WSML	SWSL
Ausdruckmächtigkeit	4	3	2	1

Tabelle 3.2: Bewertung der Ausdruckskraft.

Im Gegensatz zu WSDL-S, die zur Beschreibung ihrer Dienste keine Ontologiesprache definiert, bauen die anderen Technologien jeweils auf eine bestimmte Sprache.
Die Abbildung 3.10 zeigt die Klassifizierung dieser Sprachen. Wie OWL bestehen WSML und SWSL aus verschiedenen Varianten, die auf unterschiedlichen logischen Formalismen (DL, PL1 und LP) basieren.
PL1 (siehe Abschnitt 2.4.3) bietet umfangreichere und ausdrucksstärkere Modellierungsmöglichkeiten als DL (siehe Abschnitt 2.4.4) und LP (siehe Abschnitt 2.4.5). Der Ausdruck von LP wiederum ist stärker als der von DL. Da OWL auf DL basiert, ist ihre Ausdrucksmächtigkeit niedriger als die von WSML und SWSL, die auf DL und LP bzw. auf PL1 und LP basieren. Genauso ist die Ausdrucksmächtigkeit von WSML niedriger als die von SWSL. Da WSDL-S keine eigene Sprache definiert, ist seine Ausdrucksmächtigkeit auch am schlechtesten.

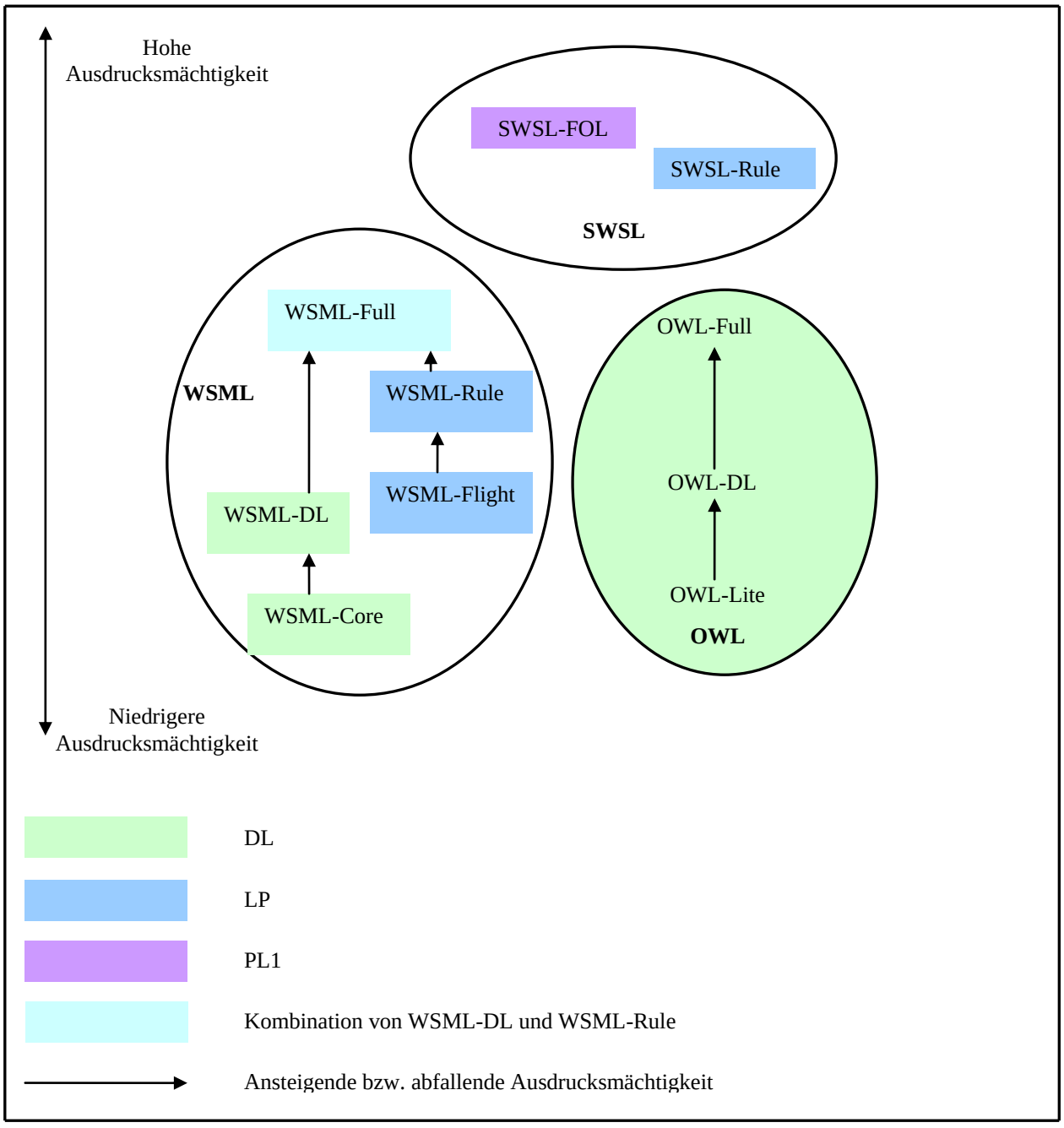

Abbildung 3.10: Klassifizierung von OWL, WSML und SWSL.

Dienstbeschreibung

Zur Beschreibung der Dienste definieren alle Ansätze funktionale und nicht-funktionale Eigenschaften, außer WSDL-S, die nur eine nicht-funktionale Eigenschaft definiert. Die nicht-funktionalen Eigenschaften ermöglichen in SWS die Verfeinerung der Suche nach einem Service. Ein Nutzer sucht z.B. nach einem roten Wagen in Hamburg, der nicht über vier Jahre alt ist. In OWL-S und SWSF wird die Definition dieser nicht-funktionalen Eigenschaften in Service Profile bzw. in Service Description eingeschränkt. WSMO jedoch definiert sie in fast allen seiner Hauptelemente (Ontologies, Goals und Web Services). Dies ist ein wichtiger Vorteil für WSMO, da auf diese Weise die Suche nach einem Service erleichtert wird. Für WSDL-S wird die Suche somit erschwert.

	WSDL-S	**OWL-S**	**WSMO**	**SWSF**
Funktionale Eigenschaft	1	1	1	1
Nicht- funktionale Eigenschaft	3	2	1	2

Tabelle 3.3: Bewertung der Dienstbeschreibung.

Automatisierung

Es soll nicht mehr manuell nach Diensten im Web gesucht werden, wie es bei Web Services der Fall ist. Das automatische Suchen und Aufrufen sind die Hauptziele von SWS. Bei der automatischen Service Discovery geht es darum, einen Service mit Hilfe von Agenten im Dienstverzeichnis zu suchen. Dies wird bei WSMO, OWL-S und SWSF vollständig erfüllt, aber bei WSDL-S nicht.
Die automatische Service Invocation wird durch den Grounding-Mechanismus ermöglicht. Bei WSMO erfolgt dieser Mechanismus anhand der WSMO-Grounding[33] und bei OWL-S mittels Service Grounding (siehe 3.1.1.3). Das Grounding von OWL-S ist ausgereifter als das von WSMO [ABB+04]. Da WSDL-S eine Erweiterung von WSDL ist, geschieht der Grounding-Mechanismus direkt. Die Invocation bei SWSF erfolgt nur durch die Ein- und Ausgaben von atomaren Prozessen [BBB+05] und seine Ausführung ist noch nicht definiert, deswegen ist es hier mit „-“ kategorisiert.

	WSDL-S	**OWL-S**	**WSMO**	**SWSF**
Service Discovery	3	1	1	1
Service Invocation	1	2	3	-

Tabelle 3.4: Bewertung der Automatisierung.

Kommunikation

In offenen Systemen wie dem Internet herrscht sehr oft das Problem der Systemsheterogenität, da verschiedene Systeme interagieren. Damit wird die Kommunikation in solchen Systemen behindert. WSMO ist der Einzige von den in dieser Arbeit vorgestellten Ansätzen, der dieses Problem explizit löst. Dazu werden Mediators verwendet. OWL-S und WSDL-S behandelt dieses Problem implizit durch Mapping-Verfahren, aber SWSF nicht.

WSMO und SWSF sind die Einzigen, die den Dienstanbieter und den Dienstnutzer unterstützen, da sie die angebotenen Dienste und die Erwartungen des Nutzers getrennt definieren. Bei WSMO

[33] http://www.wsmo.org/TR/d24/d24.2/v0.1/. Jul. 2008.

ist diese Trennung bei dessen Hauptelementen deutlich zu sehen, aber bei SWSF ist sie implizit [BBB+05]. Diese Trennung hat den Vorteil, dass es keine Abhängigkeit zwischen dem Dienstnutzer und dem -anbieter gibt. Jedoch werden sie bei OWL-S und WSDL-S nicht getrennt.

	WSDL-S	**OWL-S**	**WSMO**	**SWSF**
Lösung der Heterogenitätsprobleme	2	2	1	-
Unterstützung der Dienstanbieter	2	2	1	1
Unterstützung der Dienstnutzer	2	2	1	1

Tabelle 3.5: Bewertung der Kommunikation.

Ansatz

OWL-S, WSMO und SWSF weisen einen Top-Down-Ansatz zur semantischen Dienstbeschreibung auf, WSDL-S hingegen einen Bottom-Up-Ansatz. Der Top-Down-Ansatz definiert die Web Services und ihre Semantik, unabhängig von den bestehenden WS-Technologien. Dies wird mittels einer Ontologiesprache ermöglicht. Der Bottom-Up-Ansatz dagegen versucht die bestehenden WS-Technologien um Semantik anzureichern. Die Vorteile des Bottom-Up-Ansatzes gegenüber dem Top-Down-Ansatz sind seine Einfachheit und seine Flexibilität. Das hat zur Folge, dass WSDL-S der einzige Ansatz ist, der WS-Standards voll und ganz akzeptiert. Somit wird die Interoperabilität zu bestehenden Systemen gewährleistet. Das SOAP-Protokoll, WSDL (wegen seiner Unterstützung bei der Service Invocation) und UDDI werden bei allen Ansätzen verwendet, außer WSMO und SWSF, die kein UDDI-Verzeichnis benutzt.

	WSDL-S	**OWL-S**	**WSMO**	**SWSF**
Ansatz	Bottom-Up	Top-Down	Top-Down	Top-Down
SOAP	1	1	1	1
WSDL	1	2	3	3
UDDI	1	1	-	-

Tabelle 3.6: Vergleichtabelle der verwendeten Ansätze.

Konzeptionelles Modell

Wegen des Bottom-Up-Ansatzes wird in WSDL-S nicht versucht ein neues konzeptionelles Modell zu erstellen wie etwa in OWL-S, SWSF und WSMO. Jedoch fehlt in OWL-S und SWSF ein einheitliches konzeptionelles Modell (es gibt mehrere unterschiedliche Definitionen für den zentralen Begriff „Service“). WSMO hingegen definiert sein Modell auf Meta Object Facility (MOF) als eine aus vier Schichten bestehende Metadaten-Architektur [FLP+07]:

1. Die *Informationsschicht* enthält die Daten, die beschrieben werden sollen.

2. Die *Modellschicht* enthält die Metadaten, die die Daten der Informationsschicht beschreiben (Ontologies, Goals, Web Services und Mediators).

3. Die *Meta-Modellschicht* beschreibt die Struktur und die Semantik der Metadaten (WSMO-Eigenschaften).

4. Die *Meta-Meta-Modellschicht* beinhaltet die eigentlichen Sprachdefinitionen von WSMO.

Da WSMO im Gegensatz zu OWL-S und SWSF eine klare Trennung der Schichten definiert, hat WSMO das beste Modell.

	WSDL-S	**OWL-S**	**WSMO**	**SWSF**
Konzeptionelles Modell	3	2	1	2

Tabelle 3.7: Bewertung des konzeptionellen Modells.

Implementierung

Bei der Implementierung kommt WSDL-S an die erste Stelle, da seine Entwicklungsumgebung (METEOR-S[34]) einfacher ist zu benutzen als die von OWL-S (z.B.: OWL-S IDE[35]) und WSMO (WSMX[36]). Seine Einfachheit resultiert daraus, dass hier keine Mediatoren verwendet werden, wie es bei WSMX der Fall ist. Auch OWL-S benutzt explizit keine Mediatoren, belegt aber nur den dritten Platz, da hier viele Komponenten (OWL-S Matchmaker, JUDDI, OWL-S IDE, etc.) zusammenarbeiten müssen, was die Implementierung erschwert. In der Literatur wurde kein Tool für SWSF gefunden [BBB+05].

	WSDL-S	**OWL-S**	**WSMO**	**SWSF**
Entwicklungsumgebung	1	3	2	-

Tabelle 3.8: Bewertung der verwendeten Werkzeuge.

Fazit

	WSDL-S	**OWL-S**	**WSMO**	**SWSF**
Ausdrucksmächtigkeit	4	3	2	1
Dienstbeschreibung	3	2	1	2
Automatisierung	2	1	2	3
Kommunikation	3	3	1	2
Ansatz	1	2	4	4
Konzeptionelles Modell	3	2	1	2
Implementierung	1	3	2	4
Endbewertung	**17**	**16**	**13**	**18**

Tabelle 3.9: Endbewertung.

Nach dieser Endbewertungstabelle zeigt SWSF die schlechtesten Ergebnisse. Außerdem existieren bislang keinerlei Implementierungen, die darauf basieren. Deswegen wird dieser Ansatz im Rahmen dieser Arbeit nicht weiter betrachtet.
Aufgrund seiner Einfachheit und der hohen Akzeptanz der bestehenden WS-Technologien hat WSDL-S einen Vorteil gegenüber OWL-S und WSMO. Aber WSDL-S besitzt zur Beschreibung seiner Dienste nur eine nicht-funktionale Eigenschaft. Da für eine semantische Dienstsuche die nicht-funktionalen Eigenschaften wichtig sind, sind in dieser Hinsicht OWL-S und WSMO als besser zu bewerten.

[34] http://lsdis.cs.uga.edu/projects/meteor-s/. Mai 2008.
[35] http://projects.semwebcentral.org/projects/owl-s-ide/. Mai 2008.
[36] http://www.w3.org/Submission/WSMX/. Mai 2008.

Nach der unteren ist WSMO die beste Technologie für SWS. Zusätzlich ist WSMO konzeptionell gesehen stärker als OWL-S. Trotz der Weiterentwicklung von OWL-S wird WSMO aufgrund seiner Vorteile im Laufe dieser Arbeit verwendet, um die semantische Dienstbeschreibung zu leisten.

4 Design des Prototyps

In diesem Kapitel werden designtechnische Überlegungen angestellt, die zur Realisierung eines Prototyps mit Semantic-Web-Services-Technologie notwendig sind.

Um die konkrete Realisierung von SWS zu zeigen, wird im Abschnitt 4.1 ein Anwendungsbeispiel eingeführt. Die Anforderungen und die Architektur des Prototyps werden im Abschnitt 4.3 bzw. im Abschnitt 4.3 erläutert. Da Ontologien sich zur Definition von Metadaten im Zusammenhang mit Semantic Web bewährt haben, wird in Abschnitt 4.4 eine ausführliche Entwicklung dieser Definition beschrieben.

4.1 Anwendungsbeispiel

In diesem Beispiel geht es um die Online-Reservierung eines Wagens, welche unter Berücksichtigung des folgenden Szenarios dargestellt werden könnte:

Ein Tourist namens Markus möchte einen Wagen für zwei Tage (vom 26.06.08 zum 28.06.08) im Web reservieren. Der Wagen soll zum Beispiel von der Marke Audi sein, sowie eine rote Farbe und eine Fahrerlaubnis außerhalb des Mietstandorts (Hamburg) haben. Markus trägt seine Suchkriterien in ein Suchformular im Webbrowser ein. Diese Suchkriterien werden durch den von ihm installierten Softwareagenten in einem XML-Format formuliert und dann an das Dienstverzeichnis gesendet. Nachdem der passende Mietwagenanbieter gefunden ist, will Markus die Reservierung ausführen, indem er mit dem Anbieter direkt kommuniziert.

In diesem Szenario sind drei Beteiligte involviert: der Dienstnutzer (Markus), der Dienstanbieter (Mietwagenanbieter) und das Dienstverzeichnis. Diesen drei Teilnehmern entsprechen genau die drei Komponenten der SOA. Die WS-Standards ermöglichen das Publizieren, die Suche und den Aufruf eines Dienstes. Jedoch unterstützen die aktuellen Web Services nur eine schlüsselwortbasierte Suche nach einem Dienst. Dies hat zur Folge, dass die Suche nach dem passenden Dienst eine zeitaufwendige Angelegenheit für den Dienstnutzer ist. Dieses Problem wäre innerhalb von SWS vermeidbar, da die Webdienste anhand der Ontologie semantische Informationen enthalten. Diese semantischen Informationen würden dazu führen, dass die Softwareagenten an Stelle des Dienstnutzers den geeigneten Dienst suchen und finden.

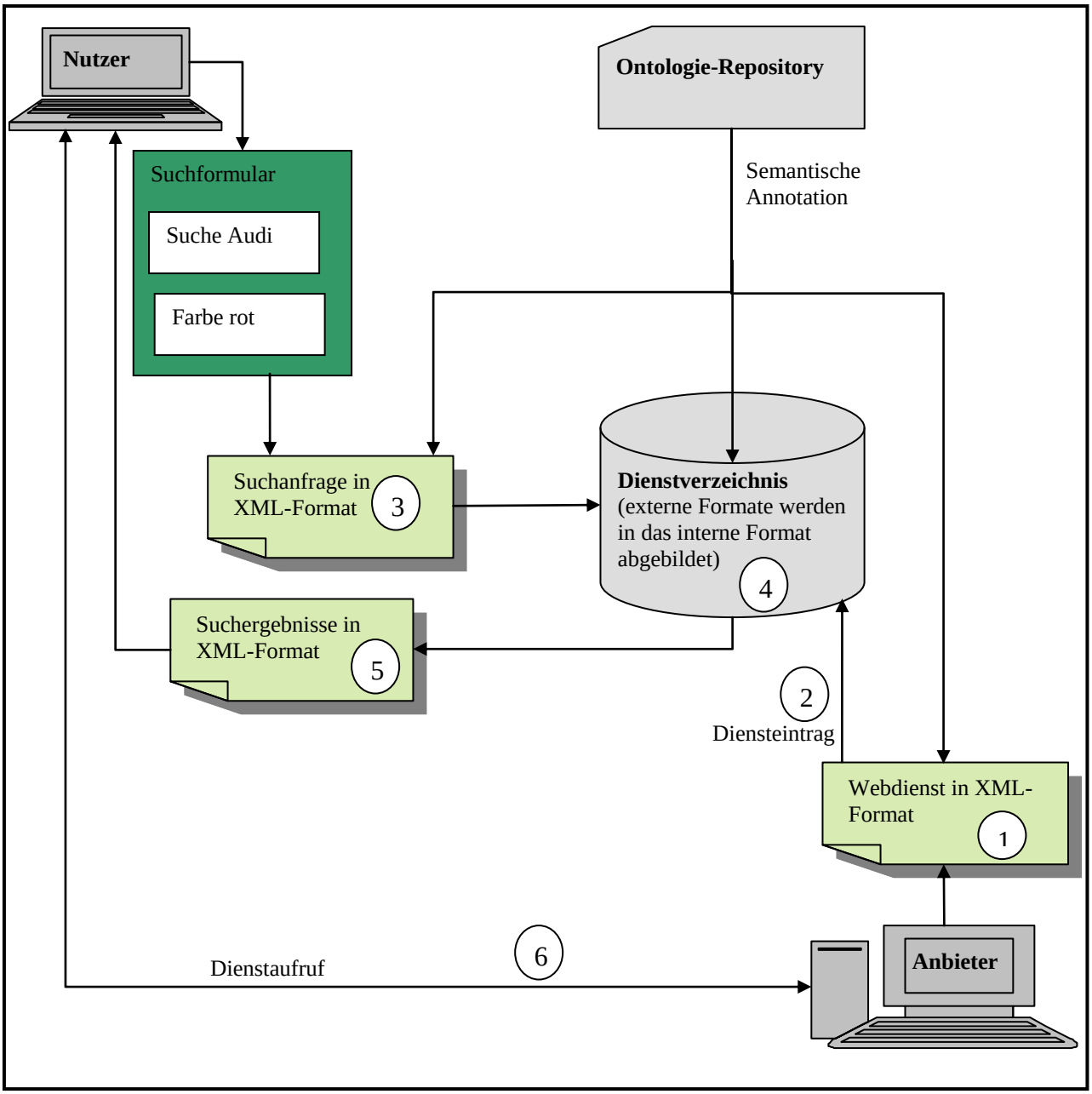

Abbildung 4.1: Anwendungsfall.

In der in Abbildung 4.1 dargestellten Anwendung startet der Prozess beim Anbieter. Er definiert seinen Dienst in XML-Format und annotiert ihn mit Konzepte der Ontologie (1). Dann veröffentlicht er seinen semantisch annotierten Dienst im Dienstverzeichnis (2). Anhand eines Webbrowsers gibt der Nutzer seine gewünschten Dienstfunktionalitäten in eine Art Suchformular ein. Diese Eingaben werden in einem XML-Format formuliert. Dann werden sie mit Metadaten der Ontologie annotiert (3). Die semantisch annotierte Anfrage wird an das Dienstverzeichnis (4) gesendet (zum Eintrag von semantisch beschriebenen Diensten siehe Abschnitt 4.2.1), wo sie zuerst in das interne Format abgebildet und dann mit vorhandenen Webdiensten verglichen werden (zum automatischen Suchen siehe Abschnitt 4.2.2). Nach dem Vergleich wird dem Nutzer eine Liste von gefundenen Diensten gesendet (5). Zu dieser Liste gehört der Dienst, der die Bedürfnisse des Nutzers am besten erfüllt (zur automatischen Auswahl siehe Abschnitt 4.2.3). Die Kommunikation zwischen dem Nutzer und dem Anbieter (6) kann jetzt stattfinden (zum automatischen Aufruf siehe Abschnitt 4.2.4).

An dem in der Abbildung 4.1 skizzierten Prozess sind vier Hauptkomponenten beteiligt: der Ontologie-Repository, das Dienstverzeichnis, der Dienstnutzeragent und der Dienstanbieteragent. Diese Hauptkomponenten werden im Abschnitt 4.4 beschrieben.

4.2 Anforderungen des Prototyps

Parallel zur SWS Anforderungen werden hier die Spezifikationen für das Prototyp definiert. Das Programm sollte generell in der Lage sein, ein Wagenreservierungsangebot zu publizieren (Abschnitt 4.2.1), zu suchen (Abschnitt 4.2.2), auszuwählen (4.2.3) und aufzurufen (Abschnitt 4.2.4). Diese Anforderungen werden im Folgenden genau erläutern

4.2.1 Eintragen eines semantisch beschriebenen Dienstes

Das Eintragen von semantisch beschriebenen Diensten ist der erste Schritt, der durchgeführt werden soll, damit nach passenden Diensten gesucht werden kann. Der Dienstanbieter muss seinen einzutragenden Dienst in WSMO semantisch beschreiben (5.3.1). Dabei soll die Beschreibung aus Fähigkeit und Interface (siehe Abschnitt 3.1.2.2.2) bestehen. Dann wird die Beschreibung durch den Dienstanbieteragent über eine Schnittstelle in den Dienstspeicher eingebracht und dort wird sie vom Mapper (4.3.4) weiterverarbeitet.

4.2.2 Automatisches Suchen

Die Suche nach semantisch beschriebenen Diensten erfolgt über eine in WSMO formulierte Suchanfrage (5.3.2). Die Anfrage muss aus einer Fähigkeit bestehen, aber nicht unbedingt aus einem Interface. Der Dienstnutzeragent sendet dann die formulierte Anfrage an das Dienstverzeichnis. Dort wird die Anfrage zuerst durch den Mapper in das interne Format transformiert und mit den im Dienstspeicher verfügbaren Diensten abgeglichen. Das Ergebnis ist eine Liste von passenden Diensten.

4.2.3 Automatische Auswahl

Aus der Liste der gefundenen Dienste wird ein Dienst, der die Wünsche des Nutzers am besten trifft, ausgewählt. Dabei spielt die nicht-funktionale Eigenschaft des Dienstes eine wichtige Rolle, da die Rangfolge der Dienste über sie bestimmt wird.

4.2.4 Automatischer Aufruf

Sobald der Dienst selektiert ist, kann er aufgerufen werden. Dazu müssen die Instanzdaten (Daten der Suchanfrage) der generierten WSMO-Ontologie in Instanzen des entsprechenden XML-Schemas (Daten der Dienste) abgebildet werden. Die XML-Anfrage kann vom Kommunikationsmodul (vom Dienstanbieteragent) aufgerufen werden. Der Dienstanbieteragent sendet dann die Instanzergebnisse an das Kommunikationsmodul (vom Dienstnutzeragent).

Im Kapitel 5 wird gezeigt, wie das Eintragen und die Suche eines semantisch beschriebenen Dienstes implementiert werden.

4.3 Architektur des Prototyps

Dieser Abschnitt stellt eine mögliche Architektur (Abbildung 4.2) des Prototyps vor. Diese Architektur enthält die geforderten Komponenten zur Realisierung des Prototyps. In den Abschnitten 4.3.1, 4.3.2, 4.3.3 und 4.3.4 werden jeweils Ontologie-Repository, Dienstanbieteragent, Dienstnutzeragent und Dienstverzeichnis beschrieben.

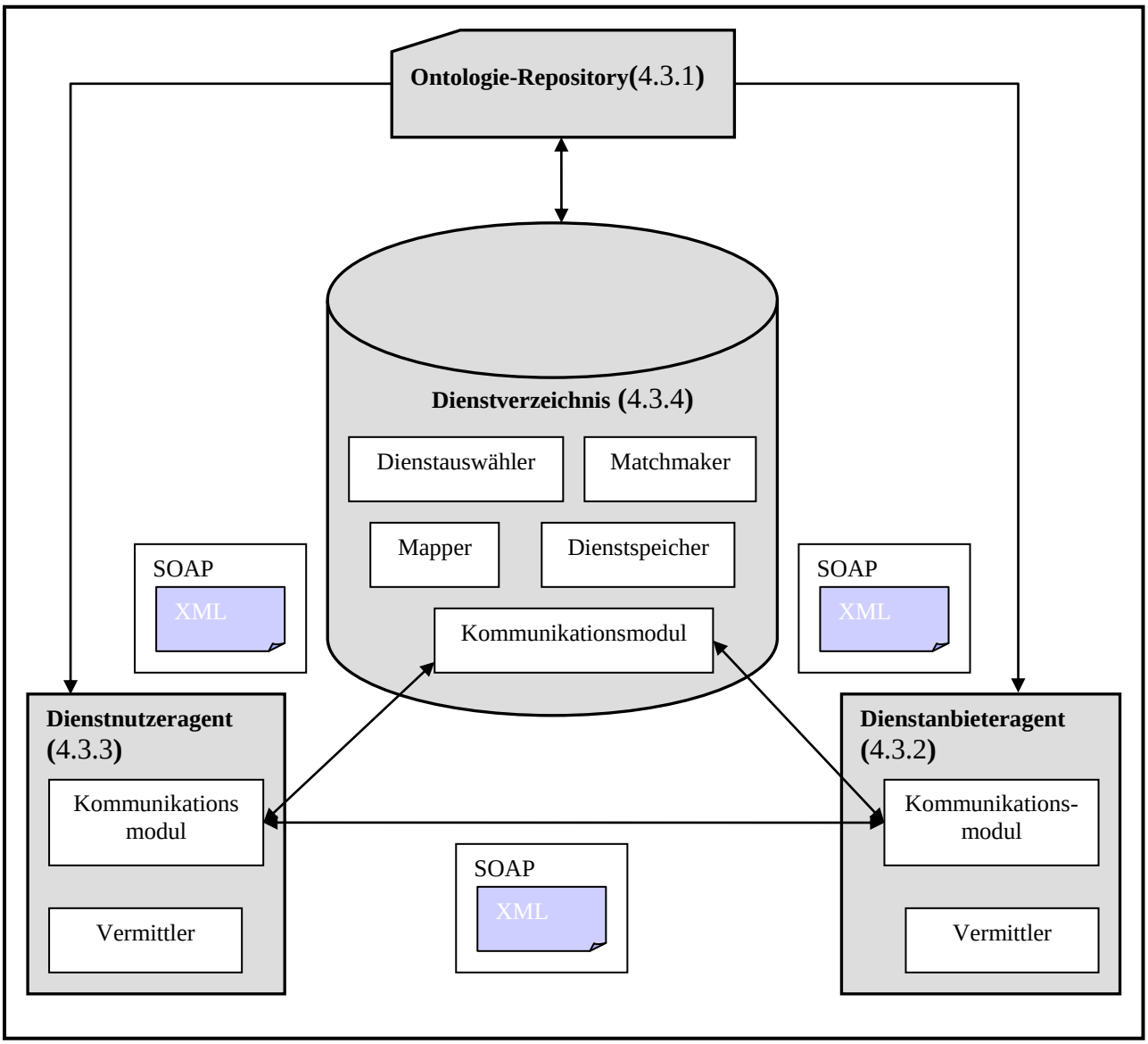

Abbildung 4.2: Architektur des Prototyps.

4.3.1 Ontologie-Repository

Der Ontologie-Repository stellt einen Standard von Metadaten dar, der von allen Benutzern des Netzwerks verwendet wird. Dem Dienstnutzeragent, dem Dienstanbieteragent und dem Dienstverzeichnis wird die gleiche semantische Beschreibung der Metadaten zur Verfügung gestellt. Somit sind die Agenten in der Lage, mit einer gemeinsamen Terminologie zu kommunizieren.

Um einen Ontologie-Repository zu entwickeln, gibt es bestimmte Verfahren. Eines von diesen Verfahren wird im Abschnitt 4.4.2 beschrieben.

4.3.2 Dienstanbieteragent

Nach der Modellierung des Dienstes hat der Dienstanbieteragent die Aufgabe, den Dienst im Dienstverzeichnis zu publizieren. Bevor der Dienst publiziert wird, soll er mit den gleichen Metadaten, die im Dienstverzeichnis gespeichert sind, annotiert werden.
Beim Publizieren sorgt das Kommunikationsmodul dafür, dass der Auftrag beim Dienstverzeichnis ankommt und dass die Antworten an den Dienstanbieteragent zurückkommen. Dabei ist das Kommunikationsprotokoll dasselbe wie bei Web Services (2.3), und das Kommunikationsformat ist XML. Außerdem ist das Kommunikationsmodul für den Dienstaufruf und die Weiterleitung der Ergebnisse an den Dienstnutzeragenten zuständig.
Um eine Inkompatibilität zwischen den Metadaten (von Dienstanbieteragent und Dienstverzeichnis) oder zwischen den verwendeten XML-Formaten zu vermeiden, wird der Vermittler verwendet.

4.3.3 Dienstnutzeragent

Der Dienstnutzeragent formuliert die Suchkriterien des Nutzers in einem XML-Format und annotiert sie dann mit den gleichen Metadaten, die sich im Dienstverzeichnis befinden. Anschließend sendet der Dienstnutzeragent diese semantisch annotierte Suchanfrage durch sein Kommunikationsmodul an das Dienstverzeichnis.
Im Fall einer Inkompatibilität der Metadaten oder der XML-Formate kann wieder ein Vermittler verwendet werden.

4.3.4 Dienstverzeichnis

Das Dienstverzeichnis enthält alle veröffentlichten Dienste und empfängt die vom Dienstnutzeragent gestellte Suchanfrage. Es hat folgende Aufgaben:

- Es speichert die semantisch beschriebenen Dienste, die vom Dienstanbieteragent kommen in der *Dienstspeicher*-Subkomponente.
- Es sorgt dafür, dass die Nachrichten, die vom Dienstanbieter- und vom Dienstnutzeragent kommen, dasselbe Format haben. Diese Aufgabe wird von der Subkomponente *Mapper* erledigt.
- Es vergleicht beide Beschreibungen (die Suchanfrage und den angebotenen Dienst) anhand der *Matchmaker*-Subkomponente.
- Es wählt den Dienst, der die Anforderungen des Nutzers am besten erfüllt anhand der *Dienstauswähler*-Subkomponente.

Die Subkomponenten des Dienstverzeichnisses werden im Folgenden beschrieben:

Dienstspeicher
Der Dienstspeicher ist ein Speicher, in dem die angebotenen Dienste gespeichert werden. Von hier werden die Dienste durch den Matchmaker geholt, um sie mit der Suchanfrage zu vergleichen.

Mapper
Das Dienstverzeichnis hat ein internes Format mit dem es arbeitet. Die beiden verschiedenen Beschreibungen, die es enthält, haben ein externes Format. Um sie zu vergleichen, muss jede Beschreibung auf das interne Format abgebildet werden.

Matchmaker
Um passende Dienste zu finden, die den Anforderungen des Nutzers entsprechen, wird die Suchanfrage mit dem angebotenen Dienst anhand des Matchingverfahrens verglichen. Dabei werden nur die Ein- und Ausgaben beider Beschreibungen berücksichtigt.
Man unterscheidet vier Typen von Matchings je nach der Ähnlichkeit zwischen der Suchanfrage und dem angebotenen Dienst [PKP+02]:

1. *exact*: Der Match ist exakt, wenn die Eingabe bzw. die Ausgabe bei beiden Beschreibungen ähnlich sind.

2. *plug in*: Die Dienstfunktionalität des Nutzers ist ein Teil der Funktionalität des Anbieters.

3. *subsume*: Die Dienstfunktionalität des Anbieters ist Teil der Funktionalität, die vom Nutzer beschriebenen ist d.h. der Anbieter liefert nicht genügend Informationen über den Dienst.

4. *fail*: Es gibt keine Ähnlichkeit zwischen der Suchanfrage und den angebotenen Dienstfunktionalitäten.

Dienstauswähler
Der Dienstauswähler bekommt vom Matchmaker die Liste mit den gefundenen Diensten. Daraus wird der Beste ausgewählt. Dann wird der selektierte Dienst dem Matchmaker zurückgesendet, der zum Schluss das Ganze (die passenden Dienste und den selektierten Dienst) an den Dienstnutzeragenten weiterleitet.

4.4 Entwicklung der Wagenreservierungsontologie

Dieser Abschnitt erläutert die Entwicklung der Wagenreservierungsontologie. Mit der Entwicklung dieser Ontologie haben die Agenten die Möglichkeit über eine gemeinsame Datenstruktur miteinander zu kommunizieren. Der Entwicklungsprozess besteht nach [GFC04] aus vier Abschnitten: der Spezifikation, der Konzeptualisierung, der Formalisierung und der Implementierung. Die ersten drei Entwicklungsschritte werden in diesem Abschnitt beschrieben und der letzte Schritt wird im Abschnitt 5.2 dargestellt.

4.4.1 Spezifikation der Wagenreservierungsontologie

Die Spezifikation legt das Ziel, den Verwendungszweck und die Wissensquellen der Ontologieentwicklung dar.

- ***Ziel der Ontologieentwicklung***
  Die hier zu entwickelte Ontologie soll das Publizieren, das Auffinden und den Aufruf eines Web Services unterstützen. Außerdem sollen die Agenten in der Lage sein, die verwendete

Terminologie verstehen zu können. Dazu soll eine gemeinsame Wortschatz bei allen (Dienstnutzeragent, -anbieteragent und -verzeichnis) vorhanden sein.

- ***Verwendungszweck***

Die Ontologie wird einerseits von den Agenten verwendet, da sie für die Automatisierung des Systems zuständig sind, und andererseits, um eine gemeinsame Terminologie zu haben.

- ***Wissensquellen***

Um die am häufigsten verwendeten Begriffe im Bereich einer Wagenreservierung zu finden, wurden folgende Quellen verwendet:

  - die Kenntnisse aus den verteilten Systemen. Das Prinzip des entfernten Methodenaufrufs (Anfrage-Antwort-Prinzip) wurde benutzt.
  - Allgemeine Informationen über einen Wagen und eine Reservierung.
  - Das Ontologiewörterbuch: Wordnet.
  - Mietwagen-Webseiten.

4.4.2 Konzeptualisierung der Wagenreservierungsontologie nach Methontology

Die Methontologie ist eine Entwicklungsmethode der Ontologie mit dem Ziel das Wissen zu organisieren und zu strukturieren. Sie besteht aus elf Entwicklungsstufen. Die Stufen 1 bis 8 können als Konzeptualisierungsstufen und die Stufen 9 bis 11 als Formalisierungsstufen betrachtet werden.

Stufe 1: Erstellung des Begriffsglossars

Diese erste Stufe hat die Aufgabe, alle relevanten Begriffe der Domain zu identifizieren (Konzepte, Konzeptattribute, Instanzen und Beziehungen zwischen den Konzepten). Bei der Beschreibung dieser Begriffe werden folgende Informationen betrachtet: der Name des Begriffs, die Synonyme des Begriffs, die Abkürzung, eine textuelle Beschreibung und der Typ des Begriffs. Dabei spielt die textuelle Beschreibung eine große Rolle, da sie eine allgemeine Bedeutung des Begriffs für den Ontologienutzer bereitstellt. Die Tabelle 4.1 und Tabelle 4.2 beschreiben die Wagenreservierungsontologie.

Name	Synonyme	Kurzwort	Beschreibung	Typ
Service	--	Serv	A service is a unit that provides something which someone need	Concept
Booking	Reservation	--	Is a Service which act to reserve a car	Concept
Car	auto	--	Is a means of transport	Concept
Economy car	--	ecoCar	Is a car of the economy class	Concept
Standard car	--	standardCar	Is a car of the standard class	Concept

Tabelle 4.1: Begriffsglossar der Wagensreservierungsontologie, Teil1.

Name	Synonyme	Kurzwort	Beschreibung	Typ
Provider	--	prov	A person or a company who provides a car for reservation	Concept
Tourist	--	--	someone who travels for pleasure	Concept
BookingDate	--	BDate	Period of Booking	Concept
BookingTime	--	BTime	Time period of booking	Concept
Destination	Terminus	--	the place designated as the end	Concept
City	--	--	City of the tourist	Concept
Country	--	--	Country of the tourist	Concept
cityIsInCountry	--	--	Specified to which country a city belong to	Relation
touristAge	--	--	Age of the tourist	Relation
Price	cost	--	the property of having material worth	Attribute
pickUpDate	--	--	Date of the pick up	Attribute
dropOffDate	--	--	Date of the drop off	Attribute
pickUpTime	--	--	Time of the pick up	Attribute
dropOffTime	--	--	Time of the drop off	Attribute
brand	Marque	--	a name given to a car	Attribute
color	--	--	Color of the car	Attribute
authorisation	--	--	official permission or approval	Attribute
firstname	--	FName	Firstname of the tourist	Attribute
lastname	--	LName	Lastname of the tourist	Attribute
title	--	--	Title of the tourist	Attribute
age	--	--	Age of the tourist	Attribute
address	--	--	Address of the tourist	Attribut
telefone	--	--	Phone of the tourist	Attribute
email	--	--	Email of the tourist	Attribute
providerName	--	--	Name of the Provider	Attribute

Tabelle 4.2: Begriffsglossar der Wagensreservierungsontologie, Teil2.

Diese beide Tabelle definiert die wichtigen Konzepte, Attribute und Beziehungen, die bei der Reservierung eines bestimmten Wagens notwendig sind.

Stufe 2: Erstellung der Konzepttaxonomie

Nach der Erstellung des Begriffsglossars soll diese zweite Stufe die identifizierten Konzepte in einer Konzepttaxonomie strukturieren, um die Rangordnung der Konzepte zu definieren. Bei der Erstellung der Konzepttaxonomie gibt es drei Ansätze: Bottom-up, Top-down und Middle-out.

1. Die *Bottom-up*-Strategie identifiziert zuerst die Konzepte, die am spezifischsten sind und dann generalisiert sie.

2. Die *Top-down*-Strategie geht in Gegenrichtung vor, d.h. von den abstraktesten zu den spezifischsten Konzepten.

3. Die *Middle-out*-Strategie identifiziert zuerst die Hauptkonzepte und dann spezifiziert oder generalisiert sie.

Bottom-up und Top-down sind als Konzeptualisierungsansatz nicht geeignet, da sie entweder sehr detailliert und nicht genug generalisiert oder sehr abstrakt und nicht ausreichend

spezifiziert sein können [UsG96]. Deswegen wird für dieses Beispiel die Middle-out-Strategie ausgewählt.

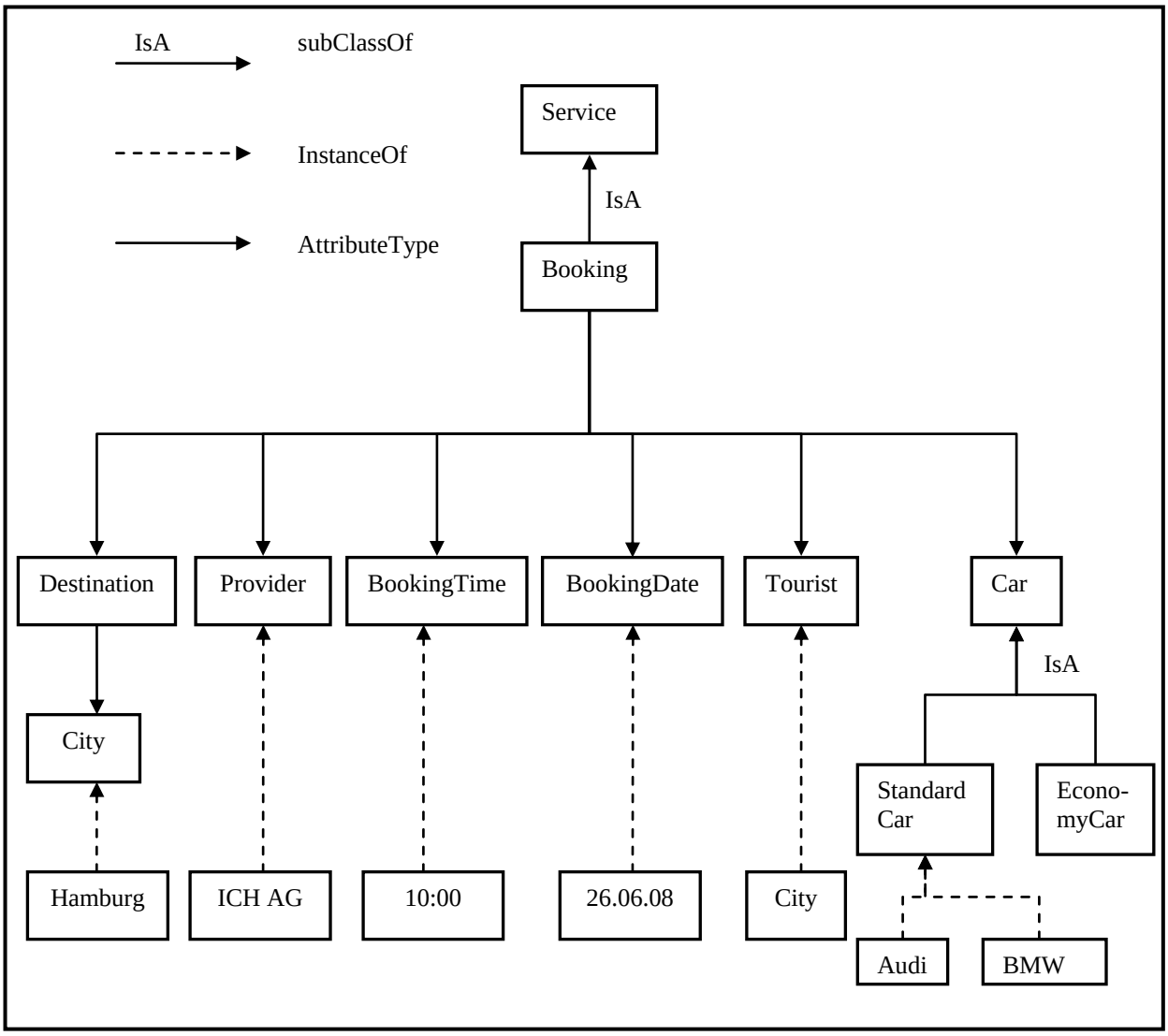

Abbildung 4.3: Taxonomie der Wagenreservierungsontologie.

In dieser Abbildung sind drei verschiedene Beziehungen zu identifizieren: subClassOf (bezeichnet die Generalisierung eines Konzeptes auf ein anderes Konzept), instanceOf (bezeichnet die Instanz eines Konzeptes) und AttributeType (Typ eines Konzeptattributs, das auch ein Konzept ist).

Stufe 3: Erstellung des binären Beziehungsdiagramms
Diese Stufe hat die Aufgabe, die Beziehung zwischen zwei Konzepten der Domain-Ontologie anhand eines Diagramms darzustellen.

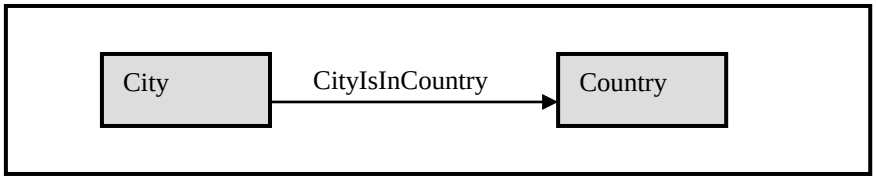

Abbildung 4.4: Binäres Beziehungsdiagramm.

Stufe 4: Erstellung des Konzeptwörterbuches
Diese Stufe soll alle identifizierten Konzepte der Domain-Ontologie mit deren Beziehungen und Attributen zusammenbringen (siehe Tabelle 4.3). Dies enthält an sich keine neuen Informationen, sondern organisiert die im Begriffsglossar enthaltenen Informationen (siehe Tabelle 4.1 und Tabelle 4.2) und klassifiziert sie nach deren Konzepten. Nach dieser Stufe werden in den Stufen fünf, sechs, sieben und acht die anderen Elemente des Begriffsglossars (Beziehungen, Attribute und Instanzen) detaillierter beschrieben.

Konzeptname	Klassenattribut	Instanzattribut	Beziehung
Service	--	--	--
Booking	pickUpDate dropOffDate pickUpTime dropOffTime providerName pickUpPlace dropOffPlace	Car Tourist	--
Car	--	Price Brand Color authorisation	isBooked
EconomyCar	--	Price Brand Color authorisation	--
StandardCar	--	Price Brand Color authorisation	--
Tourist	--	Firstname Lastname Title Age address Telefone Email	touristAge
Destination	pickUpPlace dropOffPlace	--	--
City	--	cityName Country	cityIsOnCountry
Country	--	countryName	--
BookingDate	pickUpDate dropOffDate	--	--
BookingTime	pickUpTime dropOffTime	--	
Provider	ProviderName	--	--

Tabelle 4.3: Konzeptwörterbuch.

Stufe 5: Detaillierte Definition der binären Ad-hoc-Beziehungen
Diese Stufe stellt eine detaillierte Beschreibung aller binären Beziehungen, die in dem Konzeptwörterbuch zu finden sind dar. Die Beziehungen werden nach dem Beziehungsnamen, der Kardinalität, dem Source- und dem Zielkonzept beschrieben.

Beziehungsname	Sourcekonzept	Sourcekardinalität	Zielkonzept
cityIsOnCountry	City	1	Country
Isbooked	Car	1	Tourist

Tabelle 4.4: Ad-hoc-binäre Beziehungen.

Stufe 6: Definition der Instanzattribute
Ein Instanzattribut ist ein Attribut, dessen Wert sich von jeder Konzeptinstanz unterscheidet. Die Tabelle 4.3 zeigt eine Liste von Instanzattribute. Diese werden in dieser Stufe in einer Tabelle nach dem Namen des zugehörigen Konzepts, dem Datentyp und der Kardinalität organisiert.

Instanzattribut Name	Konzeptname	Datentyp	Kardialität
car	Booking	Car	(1,1)
tourist		Tourist	(1,1)
price	Car	Decimal	(1,1)
brand		String	(1,1)
color		String	(1,1)
authorisation		Boolean	(0,1)
firstname	Tourist	String	(0,1)
lastname		String	(1,2)
title		String	(1,1)
age		integer	(1,1)
address		Address	(1,1)
telefone		String	(1,N)
eEmail		String	(0,N)
cityName	City	String	(1,1)
country		Country	(1,1)
countryName	Country	String	(1,1)

Tabelle 4.5: Instanzattribute.

Stufe 7: Definition der Klassenattribute
Im Gegensatz zu den Instanzattributen beschreiben die Klassenattribute die Konzepteigenschaften, deren Werte für die Instanz eines bestimmten Konzeptes immer gleich sind. Die Klassenattribute, die in der Tabelle 4.3 aufgelistet sind, werden hier detaillierter beschrieben. Dabei werden der zugehörige Konzeptname, der Datentyp und die Kardinalität berücksichtigt.

Klassenattribut Name	Konzeptname	Datentyp	Kardinalität
pickUpDate	Booking	BookingDate	(1,1)
dropOffDate			(1,1)
pickUpTime		BookingTime	(1,1)
dropOffDate			(1,1)
providerName		String	(1,1)
pickUpPlace		Destination	(1,1)
dropOffPlace			(1,1)
pickUpPlace	Destination	City	(1,1)
dropOffPlace			(1,1)
pickUpDate	BookingDate	String	(1,1)
dropOffDate		String	(1,1)
pickUpTime	BookingTime	String	(1,1)
dropOffTime		String	(1,1)
ProviderName	Provider	String	(1,1)

Tabelle 4.6: Klassenattribute.

Stufe 8: Definition der Konstanzen
Diese Stufe soll alle identifizierten Konstanzen des Konzeptwörterbuchs definieren. Aber diese Domain-Ontologie enthält keine Konstanzen, deswegen wird dieser Schritt übersprungen.

4.4.3 Formalisierung der Wagenreservierungsontologie nach Methontology

Sobald die Konzepte, die Taxonomie, die Attribute und die Beziehungen definiert sind, können formale Axiome und Regeln identifiziert und beschrieben werden, um die Beschränkungen innerhalb der Ontologie zu definieren bzw. zusätzliches Wissen abzuleiten. Dieses wird in den Stufen 9 und 10 dargestellt.

Stufe 9: Definition der formalen Axiome
Ein Axiom ist eine Aussage, die immer wahr ist. Es wird in dieser Stufe beschrieben. Bei der Beschreibung werden folgende Informationen berücksichtigt: der Axiomname, eine textuelle Beschreibung des Axioms, ein logischer Ausdruck in First-Order Logic, die Konzepte, die Attribute, die Beziehungen und die verwendeten Variablen.

Axiomname	City inside Country
Beschreibung	A city belongs to a country
Ausdruck	(?X)[City] (?Y)[Country] implies [cityIsInCountry](?X,?Y).
Konzepte	City, Country
Attribute	--
Ad hoc binäre Beziehung	cityIsInCountry
Variable	X,Y

Tabelle 4.7: Axiomdefinition.

Diese Tabelle zeigt die Definition eines formalen Axioms der Wagenreservierungsontologie. Dieses Axiom zeigt, dass eine Stadt zu einem Land gehört. Daran sind zwei Konzepte beteiligt: City und Country, die jeweils als X- und Y-Variablen bezeichnet werden.

Stufe 10: Definition von Regeln
Die Tabelle, welche die Regeln definiert, hat den gleichen Aufbau wie die Axiomstabelle. Demnach besteht sie aus folgenden Angaben: dem Regelnamen, einer textuellen Beschreibung, einem Ausdruck, der die Regel formal beschreibt, den Konzepten, Attributen und Beziehungen, auf die sich eine Regel bezieht und die im Ausdruck verwendeten Variablen.

Regelname	Valid Age
Beschreibung	integrity constraint for age
Ausdruck	(?X)[Tourist] and (?X, ?age)[touristAge] and ?age > 24
Konzepte	Tourist
Attribute	Age
Ad hoc binäre Beziehung	touristAge
Variable	X

Tabelle 4.8: Regeldefinition.

Diese Tabelle definiert die Gültigkeit des Alters eines Touristen mittels des Konzept Tourist, des Attribut Age, der Beziehung touristAge und der Variable X.

Stufe 11: Erstellung von Instanzen
Diese letzte Stufe ist keine Formalisierung der Ontologie, sondern unterstützt die Gültigkeitskontrolle der definierten Ontologie.

Instanzname	Konzeptname	Attribute	Werte
BergTourist	Tourist	Firstname	Markus
		Lastname	Berg
		Title	Mr.
		Age	26
		address	BergAddress
		Phone	040-444666
		email	markus@berg.de

Tabelle 4.9: Erstellung von Instanzen.

5 Implementierung

Dieses Kapitel dokumentiert, wie das im Kapitel 4 beschriebene Anwendungsbeispiel unter Nutzung der Werkzeuge, die im Abschnitt 5.1 vorzustellen sind, realisiert wird. Für diese Realisierung werden die Metadaten, das Dienstangebot und die Dienstanfrage anhand von WSMO semantisch beschrieben. Dazu wird erst die Implementierung der gemeinsamen Ontologie im Abschnitt 5.2 dargestellt. Dann wird im Abschnitt 5.3 die semantische Annotierung des Dienstangebots und der Dienstanfrage dargelegt. Die Ergebnisse der Implementierung werden im Abschnitt 5.4 vorgestellt.

5.1 Verwendete Technische Hilfsmittel

Die Werkzeuge, die im Rahmen der Implementierung verwendet wurden, sind open source. Dies soll die Erweiterbarkeit des Systems gewährleisten. Diese Werkzeuge sind:

- *WSMO Studio*[37] (Version 0.7.3): Dies ist ein Semantic-Web-Services- und eine Geschäftsprozessentwicklungsumgebung für WSMO.

- *WSMX* (Version 0.5) ist ein Framework, das auf dem konzeptionellen Modell von WSMO basiert und dazu dient, Dienstanbieter und –nutzer bei der Programmausführung zu verbinden. Siehe Abschnitt 3.1.2.4 für mehr Informationen.

5.2 Implementierung der Wagenreservierungsontologie

Bei der Realisierung des Anwendungsbeispiels (4.1) ist die Implementierung der Wagenreservierungsontologie der erste Schritt, der durchgeführt werden soll. Diese wird in diesem Abschnitt dargestellt. Dabei werden WSML (siehe Abschnitt 3.1.2.3) als Modellierungssprache und WSMO Studio (Abbildung 5.1) als Werkzeug verwendet.

[37] http://www.wsmostudio.org

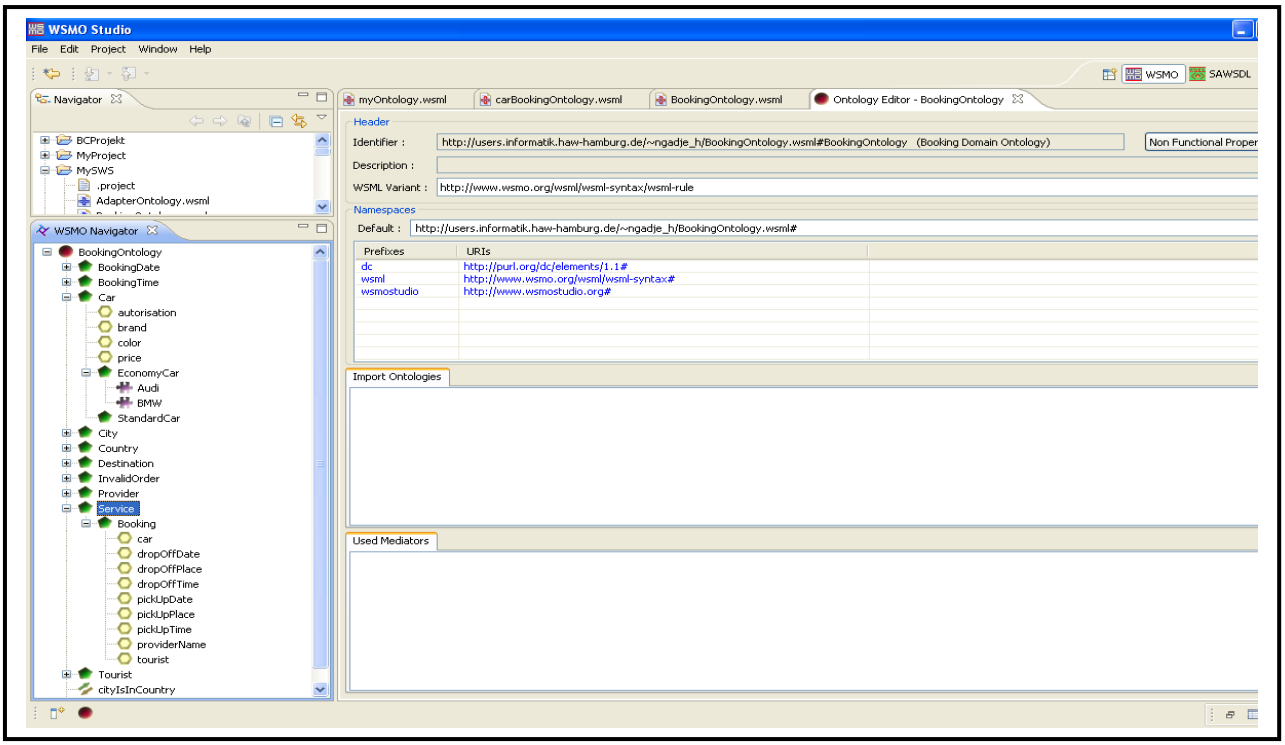

Abbildung 5.1: WSMO Studio.

Im Abschnitt 4.4 wurde die Wagenreservierungsontologie entwickelt. Dabei wurden die Konzepte, Beziehungen, Axiome, Regeln und Instanzen definiert. Diese werden in diesem Abschnitt implementiert.

Konzepte

Konzepte können mit Klassen in der objektorientierten Programmierung verglichen werden. Sie werden durch ihre Attribute definiert. Diese Attribute können anhand von Datentypen oder von anderen Konzepten definiert werden. Das Listing 5.1 zeigt die Definition von einigen Konzepten der Wagenreservierungsontologie (siehe Abschnitt 4.4.2).

```
concept Service
     nonFunctionalProperties
          dc#description hasValue "unit that provides something which someone need"
     endNonFunctionalProperties

concept Booking subConceptOf Service
     nonFunctionalProperties
          dc#description hasValue "Service which book a car for someone"
     endNonFunctionalProperties
     car impliesType Car
     tourist impliesType Tourist
     pickUpPlace impliesType Destination
     dropOffPlace impliesType Destination
     pickUpDate impliesType BookingDate
     dropOffDate impliesType BookingDate
     pickUpTime impliesType BookingTime
     dropOffTime impliesType BookingTime
     providerName impliesType Provider
```

Definition des Konzepts

Definition der Attribute

Datentyp

Listing 5.1: Konzepte der Wagenreservierungsontologie.

Beziehungen

Eine Beziehung erzeugt eine Abhängigkeit zwischen zwei Konzepten oder zwischen einem Konzept und dessen Attribut. Das Listing 5.2 stellt eine Beziehung zwischen dem Konzept City und dem Konzept Country her.

```
relation cityIsInCountry (impliesType City, impliesType Country)
   nonFunctionalProperties
       dc#description hasValue "a city belongs to an country"
   endNonFunctionalProperties
```

Bedeutung der Beziehung

Listing 5.2: Beziehung der Wagenreservierungsontologie.

Axiome

Ein Axiom kann mit Hilfe von Beziehungen implementiert werden. Das Beispiel des Listing 5.3 zeigt das Axiom cityIsInCountryDef, welches die Zugehörigkeit einer Stadt zu einem Land definiert. Dabei wird die Beziehung cityIsInCountry benutzt.

```
axiom cityIsInCountryDef
      definedBy
            (?city [country hasValue ?country] memberOf City)
                  implies
            cityIsInCountry(?city, ?country).
```

Definition der Axiom

Aufruf der Beziehung

Listing 5.3: Axiombeispiel.

Regel

Regeln werden mit Axiomen modelliert (Listing 5.4).

```
axiom validAge
      nfp
            dc#description hasValue "integrity constraint for age"
      endnfp
      definedBy
            !- ?x memberOf Tourist and touristAge(?x,?age)
      and ?age > 24.
```

Listing 5.4: Regelbeispiel.

Instanzen

Das Beispiel im Listing 5.5 zeigt einen Tourist und einen Wagen mit deren Eigenschaften.

```
instance BergTourist memberOf bo#Tourist
       bo#firstname hasValue "Markus"
       bo#lastname hasValue "Berg"
       bo#title hasValue "Mr"
       bo#age hasValue 26
       bo#address hasValue BergAddress
       bo#phone hasValue "+40-444666"
       bo#email hasValue "markus@berg.de"

instance Audi memberOf EconomyCar
     brand hasValue "Audi"
     color hasValue "Red"
     authorisation hasValue  boolean("true")
```

Instanz eines Touristen

Instanz eines Wagens

Listing 5.5: Einige Instanzen der Wagenreservierungsontologie.

5.3 Semantische Beschreibung der Dienste

Die implementierte Wagenreservierungsontologie soll in Verbindung mit dem Webdienst und der Dienstanfrage genutzt werden. In den Abschnitten (5.3.1) und (5.3.2) werden jeweils der Webdienst und die Dienstanfrage semantisch beschrieben.

5.3.1 Webdienst

Die semantische Beschreibung eines Webdienstes in WSMO wie im Abschnitt (3.1.2.2.2) beschrieben, besteht aus zwei Teilen: Fähigkeit und Interface.

Fähigkeit
Jeder Webdienst definiert nur eine Fähigkeit. Die im Listing 5.6 dargestellte Fähigkeit bietet zur Beschreibung der Funktionalitäten des Dienstes folgende Elemente an:

- Eine *Nachbedingung*: Sie repräsentiert das Angebot (Reservierung) an den Klienten. Dabei werden der Anfragesteller und der gewünschte Wagen berücksichtigt.
- Eine *Vorbedingung*: Sie prüft, ob eine Wagenreservierungsanfrage vorliegt.
- Eine *Voraussetzung*: Sie prüft, ob das Alter des Touristen im richtigen Bereich liegt (d.h. größer als 24 Jahre).

Diese Elemente werden durch ein Axiom definiert. Für die Definition der Voraussetzung wird die Regel „validAge“ (ist in der Ontologie vorhanden, siehe Abschnitt 5.2) verwendet.

```
precondition
       nonFunctionalProperties
              dc#description hasValue "The information about
              the car and about the person who wants to
              have a reservation
             must be given as a part of a booking request."
      endNonFunctionalProperties
      definedBy
            ?bookingRequest[
               car hasValue ?bookingItem,
               for hasValue ?bookingHolder
           ] memberOf bod#bookingReq.

assumption
      nonFunctionalProperties
          dc#description hasValue "The age provided by the
          requester must be valid."
      endNonFunctionalProperties
      definedBy
          bo#validAge.

postcondition
       definedBy
             ?booking[bo#car hasValue ?bookingItem,
              bo#tourist hasValue bookingHolder
             ] memberOf bo#Booking.
```

Vorbedingung

Voraussetzung

Nachbedingung

Listing 5.6: Webdienst-Fähigkeit.

Interface

Das Interface soll beschreiben, wie mit dem Webdienst zu kommunizieren ist. Dabei werden Choreography und Orchestration verwendet. Da die Orchestration die Zusammensetzung verschiedener Webdienste beschreibt und da das Anwendungsbeispiel keine Orchestration behandelt, wird auf sie hier verzichtet. Jedoch bezeichnet die Choreography die Kommunikation zwischen dem Nutzer und dem Dienst, somit hat der Dienstnutzer die Möglichkeit die Funktionalitäten des Dienstes zu nutzen. Diese Kommunikation wird mittels *stateSignature* und *transitionRule* beschrieben. Die stateSignature wird durch Ontologiekonzepte repräsentiert. Dennoch spezifiziert die transitionRule die Veränderungen des Zustands. Hier sind zwei Transitionen zu identifizieren:

1. Die erste prüft, ob eine Wagenreservierung schon vorliegt.

2. Bei der zweiten wird die Reservierung realisiert.

Das Listing 5.7 stellt die Choreographiebeschreibung der Wagenreservierung dar. Dabei wird die Wagenreservierungsontologie importiert (durch „*importsOntology*“), um die darin definierte Terminologie und die Instanzen verwenden zu können.

```
choreography carBookingWSBBookingOrderChoreography
    stateSignature carBookingWSBBookingOrderStateSignature

      importsOntology
                  {
                        bo#BookingOntology,
                        bod#BookingOntologyDef
                  }

      in bod#BookingReq withGrounding
            { _"http://users.informatik.haw-
            hamburg.de/~ngadje_h/bookingv1/carB.wsdl#wsdl.interfaceMessage
            Reference(carBSOAP/BookingOrder/in0)"}
      in bo#Tourist
      in bo#BookingDate
      in bo#BookingTime
      in bo#Destination
      in bo#Car
      out bod#BookingResp

    transitionRules carBookingWSBBookingTransitionRules

      forall {?request} with
        (?request memberOf bod#BookingReq)
      do
          add(_#1 memberOf bod#BookingResp)
      endForall
```

Abbildung der Konzepte auf WSDL-Daten

Eingabekonzepte

Ausgabekonzept

Erste Transition

Zweite Transition

Listing 5.7: Webdienst-Interface.

5.3.2 Dienstanfrage

```
postcondition
            definedBy
                  ?booking[bo#car hasValue ?bookingItem,
                           bo#tourist hasValue bookingHolder
                  ] memberOf bo#Booking.

interface GoalInterface
      choreography GoalChoreography
            stateSignature GoalStateSignature
                  importsOntology
                  {
                        bo#BookingOntology,
                        bod#BookingOntologyDef
                  }
                  in bod#BookingReq
                  out bod#BookingResp
            transitionRules GoalTransitionRules
                  forall {?request} with
                        (?request memberOf bod#BookingReq)
                  do
                        add(_#1 memberOf bod#BookingResp)
                  endForall
```

Listing 5.8: Dienstanfrage.

Die Dienstanfrage (siehe Listing 5.8) spielt eine große Rolle bei der Auffindung und dem Aufruf eines Dienstes. Sie hat dieselben Bestandteile wie der Webdienst (5.3.1). Aber es wird hier nur die Nachbedingung in der Fähigkeit berücksichtigt, da die Dienste über diese gefunden werden.

5.4 Nutzung eines Webdienstes in WSMX

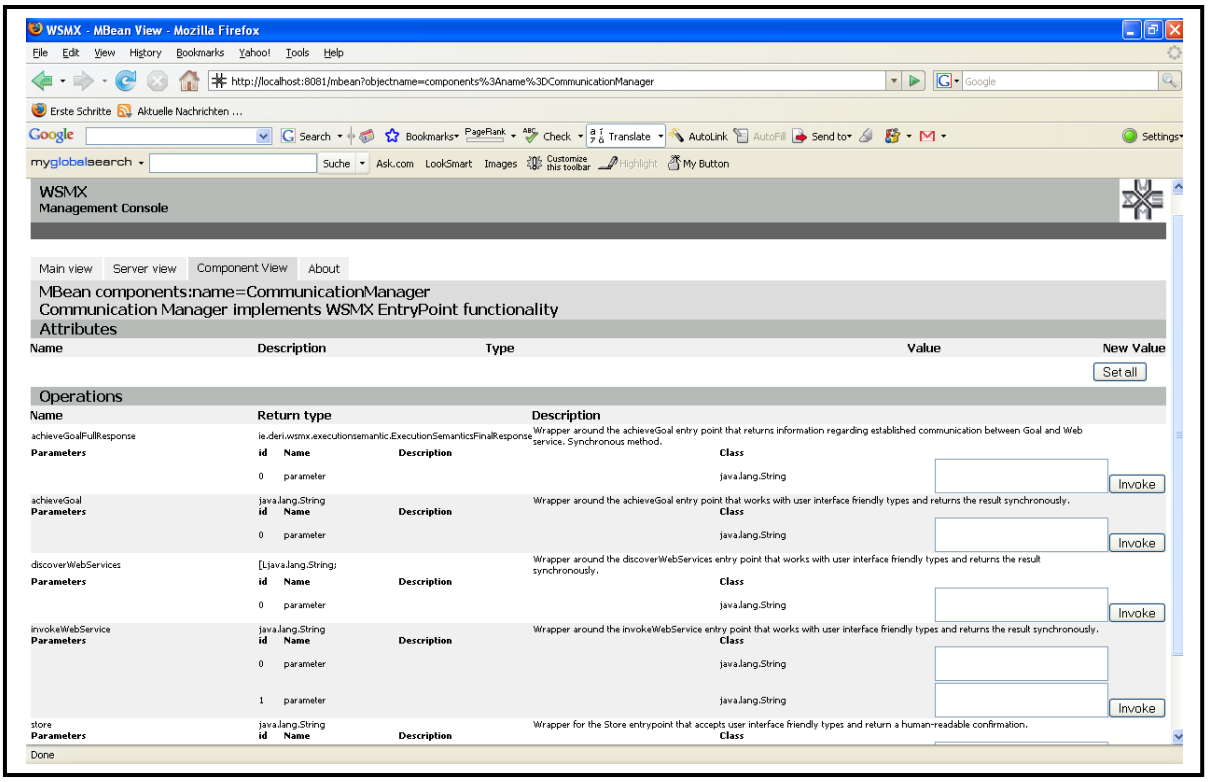

Abbildung 5.2: WSMX-Communication-Manager.

WSMX ist eine Referenzimplementierung von WSMO. Für die Implementierung des im Kapitel 4 vorgestellten Anwendungsbeispiels stellt WSMX eine *Communication-manager*-Komponente zur Verfügung (siehe Abbildung 5.2). Diese Komponente stellt schon eine API zur Verfügung, die es ermöglicht, einen Webdienst im Dienstverzeichnis zu publizieren, zu suchen und aufzurufen. Die Methoden dieser API können entweder über ein HTTP oder eine Web-Service-Schnittstelle aufgerufen werden.

5.4.1 Eintragen eines Webdienstes in WSMX

Wie im Abschnitt (4.2.1) erwähnt, muss der einzutragende Dienst zuerst in WSMO semantisch beschrieben werden. Dies wurde in Abschnitt (5.3.1) erledigt. Anschließend kann diese Beschreibung anhand der Methode „store“ der API publiziert werden. Aber davor sollen zusätzliche Axiome definiert werden, um die Geschäftsregeln eines Dienstes zu kodieren.
Ein Axiom, das bewertet, ob der Webdienst einen bestimmten Wagen an einem spezifischen Datum reservieren kann, kann z.B. erforderlich sein. Da mehrere Axiome in einer Nachbedingung der Fähigkeitsbeschreibung nicht definiert werden können, wurde eine externe Axiomontologie (begrenzt die spezifischen Fähigkeiten eines Dienstes) definiert. Die Listing 5.9, Listing 5.10 und Listing 5.11 stellen jeweils den zusätzlichen Axiomen folgenden Webdienste dar: carBookingWSA, carBookingWSB und carBookingWSC.

```
axiom isBookedDef
       definedBy
              ?bookingReq[bod#car hasValue ?car] memberOf   bod#BookingReq and
              (?car[bo#authorisation hasValue ?authorisation,
                     bo#brand hasValue ?brand,
                     bo#color hasValue ?color,
                     bo#price hasValue ?price
              ] memberOf bo#Car) and
              (?authorisation = true ) and
              (?brand = "BMW") and (?color = "Black")  and
              (?price = 800.0)
              implies
              bod#isBooked(?bookingReq).
```

Listing 5.9: carBookingWSA.

```
axiom isBookedDef
       definedBy
              ?bookingReq[bod#car hasValue ?car] memberOf   bod#BookingReq and
              (?car[bo#authorisation hasValue ?authorisation,
                     bo#brand hasValue ?brand,
                     bo#color hasValue ?color
                     bo#price hasValue ?price
              ] memberOf bo#Car) and
              (?authorisation = true ) and
              (?brand = "Audi") and (?color = "Red") and (?price = 800.0)
              implies
              bod#isBooked(?bookingReq).
```

Listing 5.10: carBookingWSB.

```
axiom isBookedDef
       definedBy
              ?bookingReq[bod#car hasValue ?car] memberOf   bod#BookingReq and
              (?car[bo#authorisation hasValue ?authorisation,
                     bo#brand hasValue ?brand,
                     bo#color hasValue ?color
                     bo#price hasValue ?price
              ] memberOf bo#Car) and
              (?authorisation = false ) and
              (?brand = "Audi") and (?color = "Red") and (?price = 900)
              implies
              bod#isBooked(?bookingReq).
```

Listing 5.11: carBookingWSC.

Der Unterschied zwischen diesen Diensten ist folgender:

1. Der Dienst carBookingWSA bietet einen schwarzen BMW (kostet 800 €) mit Fahrterlaubnis außerhalb des Reservierungsstandorts an.

2. Der Dienst carBookingWSB bietet einen roten Audi (kostet 800 €) mit Fahrterlaubnis außerhalb des Reservierungsstandorts an.

3. Der Dienst carBookingWSC bietet einen roten Audi (kostet 900 €) ohne Fahrterlaubnis außerhalb des Reservierungsstandorts an.

Bei der erfolgreichen Veröffentlichung sieht das System so aus:

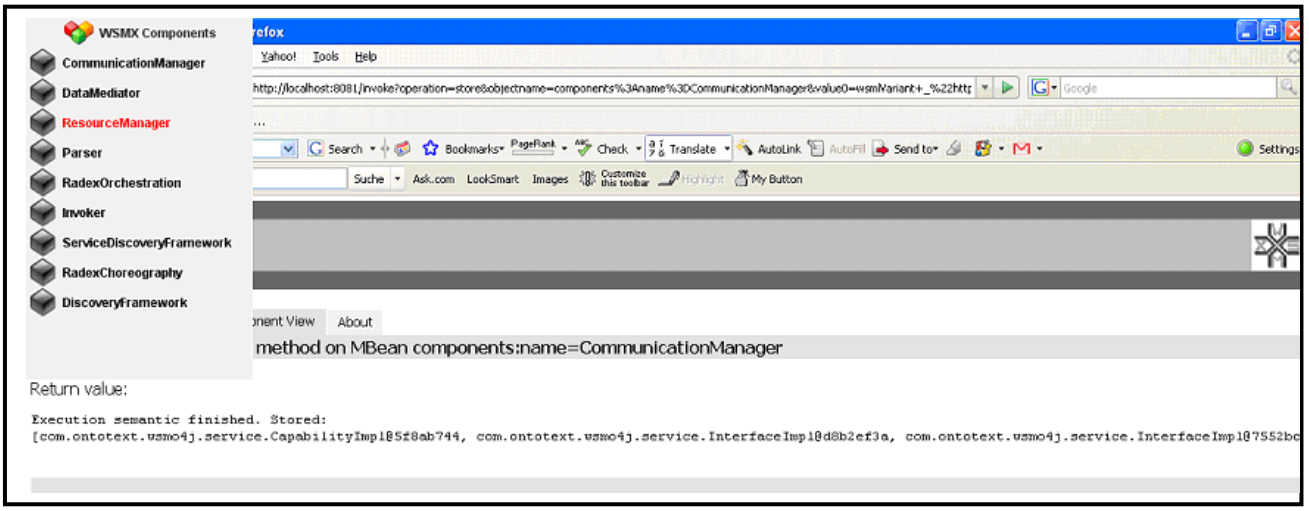

Abbildung 5.3: Das Publizieren eines Webdienstes.

5.4.2 Suche eines Webdienstes in WSMX

Nachdem die Webdienste publiziert sind, kann anhand der Dienstanfrage (siehe Abschnitt 5.3.2) und der Instanzdaten (Listing 5.12) nach dem passenden Dienst gesucht werden. Der Auffindungsprozess besteht aus zwei Phasen: zuerst werden die passenden Dienste ausfindig gemacht. Dann wird der Dienst, der die angegebenen Bedürfnisse des Dienstnutzers am besten trifft, aus der Liste der gefundenen Dienste ausgewählt.
Die Instanzdaten sollen alle Wünsche des Anfragestellers berücksichtigen. Das Listing 5.12 stellt zwei Instanzen dar. Die erste Instanz ist eine Reservierungsanfrage, die Daten wie Reservierungsdatum, -zeit, -standort, gewünschter Wagen und Kontaktdaten enthält. Der gewünschte Wagen wird in der zweiten Instanz spezifiziert.

```
instance bookingReq memberOf bod#BookingReq
       bod#bookingDate hasValue bo#bookingDate1
       bod#bookingTime hasValue bo#bookingTime1
       bod#bookingPlace hasValue bo# destination1
       bod#car hasValue Audi
       bod#for hasValue bo#BergTourist

instance Audi memberOf bo#EconomyCar
     color hasValue "Red"
     authorisation hasValue _boolean("true")
     brand hasValue "Audi"
```

Listing 5.12: Instanzdaten.

Für die Suche steht die Methode „achieveGoal“ zur Verfügung. Diese Methode nimmt als Eingabe den Dienstanfrage-Quellcode (Listing 5.8) in Verbindung mit den Daten aus Listing 5.12 an. Beim erfolgreichen Aufruf dieser Methode lassen sich die Ergebnisse der Suche auf dem „WSMX Monitor“ visualisieren.

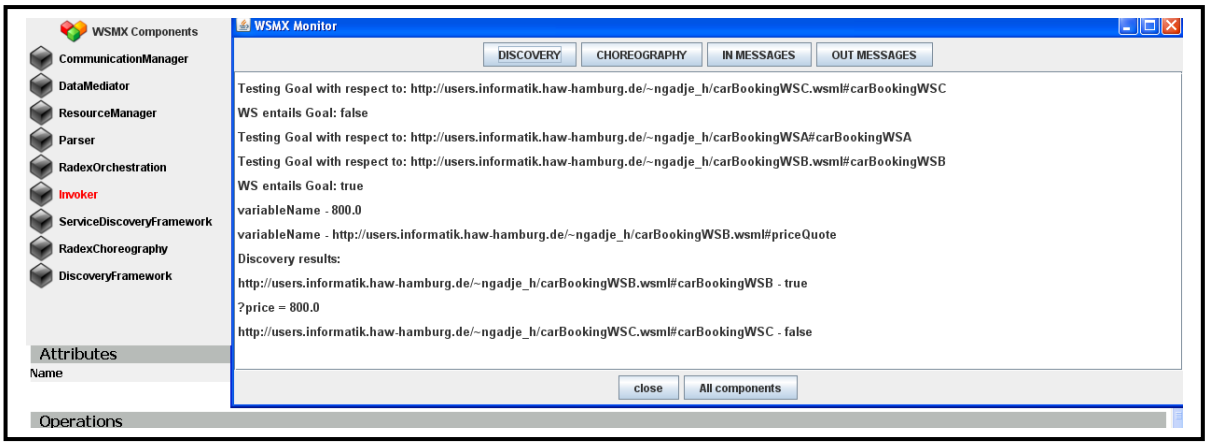

Abbildung 5.4: Suchergebnisse.

In der Abbildung ist eine Liste von drei verfügbaren Dienste dargestellt: carBookingWSA, carBookingWSB und carBookingWSC. Die Dienste carBookingWSB und carBookingWSC sind die Dienste, die die Anfrage unterstützen. Jedoch trifft der Dienst carBookingWSB die Funktionalitäten der Anfrage am besten, da er die Fahrt außerhalb des Reservierungsstandorts erlaubt, was bei carBookingWSC nicht der Fall ist. Der Dienst carBookingWSA wurde schon in der ersten Runde eliminiert, da er keinen roten Audi, sondern einen schwarzen BMW anbietet.

5.4.3 Aufruf eines Webdienstes in WSMX

Der Aufruf eines Webdienstes stellt die Kommunikation zwischen dem Dienstnutzer und dem Webdienst her. WSMX ermöglicht dieser Aufruf anhand der Methode „achieveGoal“. Dabei werden die Instanzdaten (Listing 5.12) der Anfrage berücksichtigt. Diese Daten werden durch den Grounding-Mechanismus auf Daten der WSDL-Beschreibung des Dienstes abgebildet (siehe Abbildung 5.5 und Abbildung 5.6).

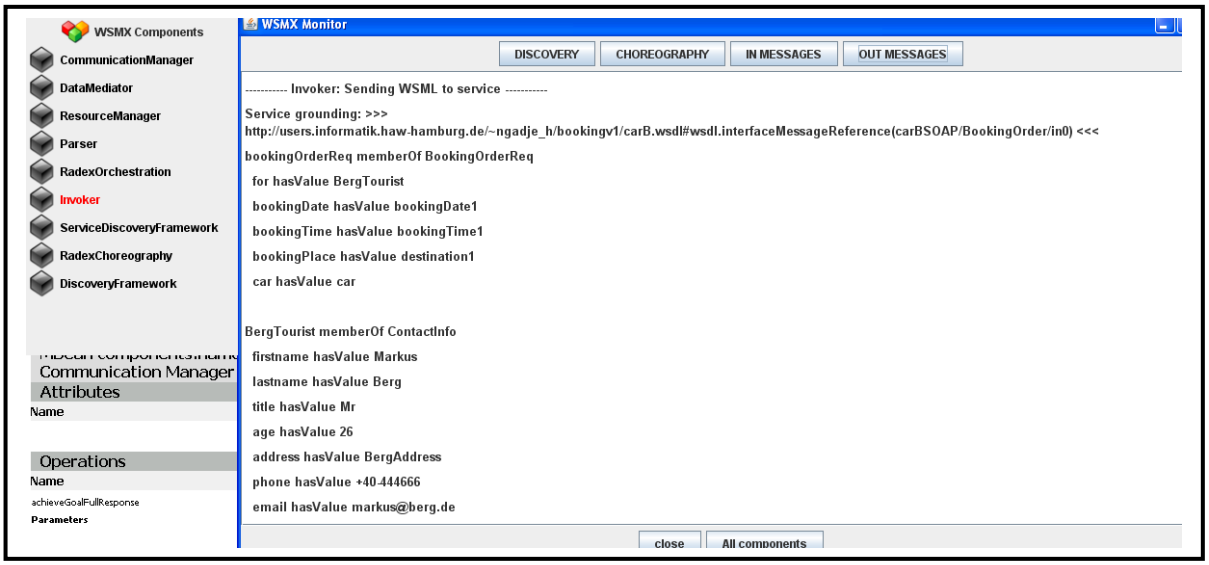

Abbildung 5.5 WSMX-Grounding, Teil1.

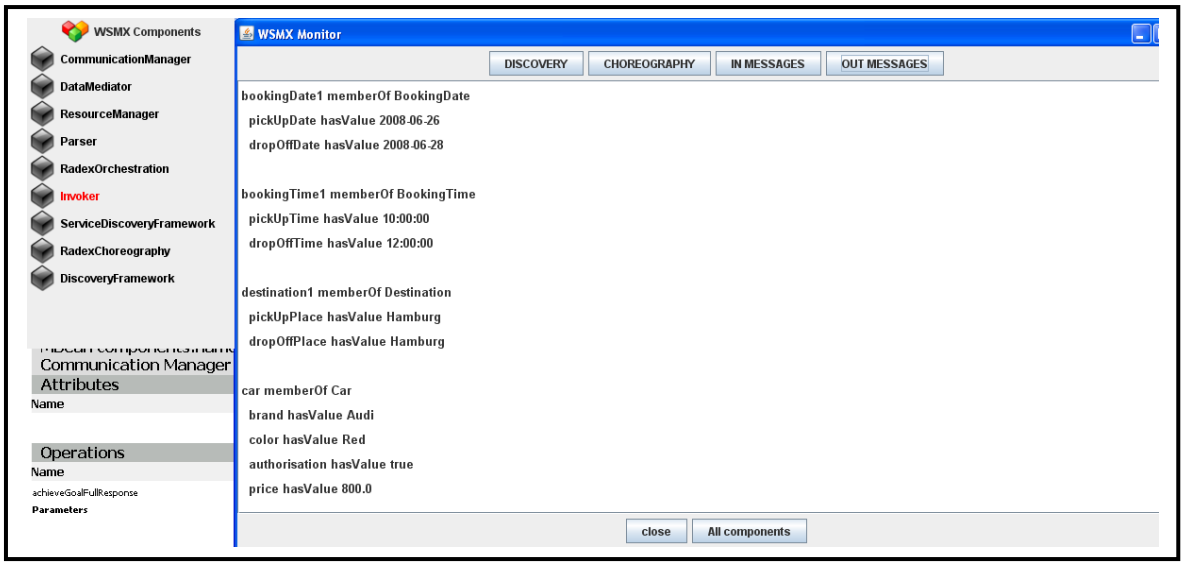

Abbildung 5.6 WSMX-Grounding, Teil2.

6 Bewertung

Dieses Kapitel bewertet die WSMO-Spezifikation anhand der während der Implementierung des Anwendungsbeispiels gemachten Erfahrungen. Dabei werden die Vorteile einer semantischen Beschreibung von Web Services auf Basis von WSMO aber auch einige Mängel oder mögliche Probleme evaluiert. Zu diesem Zweck gliedert sich das Kapitel in drei Bereiche. Zuerst wird das konzeptionelle Modell von WSMO behandelt, dann wird die WSMO-Modellierungssprache (WSML) betrachtet und zum Schluss wird die praktische Einsetzbarkeit von WSMO bewertet.

6.1 Struktur von WSMO

Nach der Analyse der verschiedenen Ansätze von SWS im Abschnitt 3.2 wurde WSMO wegen seiner Vorteile als die beste Technologie für eine semantische Beschreibung ausgewählt. Seine Struktur definiert eine klare Trennung der WSMO-Hauptelemente (3.1.2.2), d.h. jedes Element wird unabhängig von anderen Elementen spezifiziert. Dieses steht im Einklang mit der offenen und verteilten Natur des Webs.
Die funktionalen und nicht-funktionalen Eigenschaften werden bei der Formulierung von der Anfrage und dem Webdienst getrennt. Diese Trennung ist auf Grundsätze der Softwaretechnik abgestimmt.

6.2 WSMO-Modellierungssprache

Der weitere große Vorteil von WSMO im Vergleich zu anderen Ansätzen zur Beschreibung von Semantic Web Services wie OWL-S ist die formale Sprache WSML (3.1.2.3), welche auf verschiedenen logischen Formalismen (DL, PL1 und LP) basiert. Sie ermöglicht somit eine ausdrucksstärkere Beschreibung der Dienste.

Trotz der Vorteile von WSMO ergeben sich einige Mängel oder Probleme im Modellierungsumfeld. Die Semantik von WSML stellt ein Problem dar. WSML wurde speziell für WSMO-Beschreibungen entwickelt und sollte eine Semantik für alle WSMO-Hauptelemente bereitstellen. Dies geschieht aber im Moment nur für das Hauptelement *Ontologies*. Die Capability, welche in der Beschreibung der Anfrage sowie des Webdienstes eine wichtige Rolle spielt, enthält keine Semantikdefinition. Dies führt dazu, dass die Semantik von WSML unvollständig ist.

Es hat sich bei der Realisierung gezeigt, dass es wünschenswert ist, eine zusätzliche Funktionalität in WSML zu haben, welche die Verarbeitung von Zeichenketten und dezimalen Zahlen unterstützt. Die aktuelle WSML-Sprache gibt nicht die Möglichkeit, wie in objektorientierten Sprachen (z.B. Java) anhand einer temporären Variable zwei Attribute vom Typ „String“ und vom Typ „Integer“ zusammenzusetzen (oder einen Substring auszulesen) bzw. zu vergleichen (<, >, =). Dieses könnte in WSMO durch ein Axiom geschehen, was den größeren Aufwand bedeuten würde, mehr Codezeilen schreiben zu müssen.

6.3 Praktische Einsetzbarkeit

Neben den im Abschnitt 6.2 erwähnten Mängeln von WSML stellte die Komplexität von WSML eine signifikante Schwierigkeit bei der Implementierung des Anwendungsbeispiels dar. Neben WSML-Rule und WSML-Flight (die zwei WSML-Varianten, die im Rahmen der Implementierung verwendet wurden) sind zusätzlich Kenntnisse in WSDL und SOAP gefordert, um eine vollständige WSMO-Beschreibung zu erstellen. So war eine Einarbeitung notwendig. Leider war das WSMO-Studio-Werkzeug nicht von so großem Vorteil, da es über keinen Debugger verfügt, der bei der Verifikation der erstellten semantischen Beschreibung helfen könnte. Wie WSMO-Studio hat WSMX keinen Debugger. Somit war die ständige Suche nach der Fehlerquelle eine zeitaufwendige Angelegenheit.

Der Grounding-Mechanismus, welcher die semantischen Beschreibungen in WSDL repräsentiert, ist ein wesentlicher Aspekt beim Aufruf eines Dienstes. Das Grounding soll Details über den Nachrichtenaustausch zwischen dem Nutzer und dem Webdienst zeigen. Aber diese Funktionalität wird leider nicht von WSMX unterstützt. Im Moment werden nur die Instanzdaten der Anfrage auf Daten der WSDL-Beschreibung abgebildet (Abbildung 5.5 und Abbildung 5.6).

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Diese Arbeit hat die verschiedenen Ansätze zur Implementierung von Semantic Web Services aufgezeigt und bewertet. Außerdem wurde anhand von WSMO ein Anwendungsbeispiel, nämlich ein Wagenreservierungssystem implementiert.

Zunächst wurden die beiden Ausgangstechnologien beschrieben, die die Grundlage SWS bilden, nämlich Semantic Web und Web Services. Dabei wurden die verschiedenen Technologien und Konzepte vorgestellt. Dann wurde eine Reihe konkreter Ansätze von SWS (WSDL-S, SWSF, OWL-S und WSMO) präsentiert und verglichen. Aus diesem Vergleich dieser verschiedenen Ansätze, welcher den ersten Schwerpunkt dieser Arbeit bildete, wurde folgende Schlussfolgerung gezogen:

- Die Modellierungssprache von SWSF hat die beste Ausducksmächtigkeit. Der Grund ist darin zu finden, dass die beide Teile von SWSF, nämlich SWSO und SWSL, eine Erweiterung von OWL-S bzw. von WSMO sind. Aber es existiert noch kein Werkzeug für SWSF zur Unterstützung bei der Implementierung.

- WSDL-S ist eine Technologie, die WS-Standards unterstützt, daher wird die Interoperabilität zu bestehenden Systemen gesichert. Sie definiert keine eigene Sprache, kann aber andere Sprachen wie OWL oder RDF verwenden. Leider definiert sie nur eine nicht-funktionale Eigenschaft zur Beschreibung seiner Dienste.

- OWL-S ist ein weiterentwickelter Ansatz, welcher eine detaillierte Service-Ontologie bietet. Dieser Ansatz unterscheidet nicht zwischen dem Angebot des Anbieters und der Anfrage des Nutzers und verwendet zu seiner Implementierung mehrere Komponenten, die zusammenarbeiten müssen.

- WSMO hat nicht nur ein starkes konzeptionelles Modell, welches das Dienstangebot von der Dienstanfrage deutlich trennt, sondern auch eine ausdruckstarke formale Sprache und ein einheitliches Implementierungswerkzeug. Sie bietet daher die beste Möglichkeit, einen

Dienst semantisch zu beschreiben. Deswegen wurde es für die Implementierung von SWS ausgewählt.

Als zweiter Schwerpunkt dieser Arbeit wurden mit Hilfe von WSMO das Design und die Implementierung eines Wagenreservierungssystems realisiert. Beim Design wurden die notwendigen Komponenten und Anforderungen des Systems definiert. Die Wagenreservierungsontologie wurde entwickelt und mit dem Werkzeug WSMO-Studio implementiert. Diese Software ermöglichte auch die semantische Beschreibung von Dienstangeboten (WSMO-Web-Services) und -anfragen (WSMO-Goals).

Ein weiteres Vorhaben dieser Arbeit war es, die Implementierung des Wagenreservierungssystems mit WSMO zu evaluieren. Dabei wurden folgende Bewertungspunkte betrachtet: die Struktur, die Modellierungssprache und die praktische Einsetzbarkeit von WSMO.

- Die Struktur von WSMO stellt eine deutliche Trennung von der Suchanfrage (Goals) und dem Dienstangebot (Web Services) dar.

- Die Semantik der ausdruckstarken Sprache WSML ist unvollständig, da WSML die Semantik nur für eine seiner Hauptelemente definiert, nämlich das Ontologie-Element. Außerdem mangelt es noch an einer zusätzlichen Funktionalität in WSML, und zwar die Verarbeitung von Zeichenketten und dezimalen Zahlen.

- Bei der praktischen Einsetzbarkeit wurde festgestellt, dass es hilfreich wäre, einen Debugger im Werkzeug zu haben. Dann wäre die Fehlersuche keine so zeitaufwendige Angelegenheit. Des Weiteren zeigt das WSMX-Werkzeug keine Details über den Nachrichtenaustausch zwischen Dienstnutzer und Dienstanbieter.

7.2 Ausblick

In der Arbeit wurden die Technologien von Semantic Web Services vorgestellt und bewertet. Bei der Bewertung der Ausdrucksmächtigkeit der verschiedenen Ansätze besetzte SWSF die erste Stelle. Anlass für eine weiterführende Arbeit wäre die Entwicklung eines Werkzeugs zur Unterstützung von SWSF, da im Moment noch keines existiert.

Einige Verbesserungen in Bezug auf das in dieser Arbeit vorgeschlagene Design für das Wagenreservierungssystem sind denkbar. Im Design verläuft die Kommunikation zwischen den verschiedenen Komponenten in verschiedenen XML-Formaten. Dies führt zu einigen Problemen:

- Das Dienstverzeichnis soll die Nachrichten, die vom Dienstanbieteragent und Dienstnutzeragent gesendet werden, jedes Mal in das interne Format (WSML) umwandeln.

- Der Dienstanbieteragent und Dienstnutzeragent müssen jedes Mal die enthaltenen Nachrichten in das interne Format (XML) zurückwandeln.

- Die Um- und Rückwandlung führt zu lange Verarbeitungszeiten.

Eine Lösung für dieses Design wäre es, bei allen drei Komponenten das gleiche XML-Format zu benutzen, nämlich WSML. Die Unternehmen dazu zu bringen, das gleiche XML-Format für ihre Webdienst-Transaktionen zu benutzen, ist leider eine schwierige Angelegenheit, da eine solche Umstellung einen Anstieg von Kosten und Know-how mit sich bringt.

WSMO ermöglicht eine automatische Dienstsuche. Die aktuellen WS-Standards unterstützen das zwar nicht, aber sie ermöglichen sichere Datentransaktion und sicheres Datenmanagement während der Geschäftverhandlung zwischen Kunden und Verkäufer. Dies ist bei WSMO nicht der Fall. WSMO ermöglicht zwar die beste Modellierung, Analyse und Implementierung von Diensten, aber nicht die Verwaltung und die Sicherheit bei Online-Verhandlungen.

Anhang

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://users.informatik.haw-
hamburg.de/~ngadje_h/BookingOntology#"
,
     dc _"http://purl.org/dc/elements/1.1#",
     wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
     wsmostudio _"http://www.wsmostudio.org#" }

ontology BookingOntology
     nonFunctionalProperties
          dc#date hasValue _date(2008,05,23)
          wsmostudio#version hasValue "0.7.3"
          dc#contributor hasValue "Hulda Ngadjeu"
          dc#title hasValue "Booking Domain Ontology"
          dc#language hasValue "en-US"
     endNonFunctionalProperties

axiom cityIsInCountryDef
     definedBy
          ?city[country hasValue ?country] memberOf City
      implies
            cityIsInCountry(?city,?country).

axiom validAge
     nonFunctionalProperties
          dc#description hasValue "integrity constraint for age"
     endNonFunctionalProperties
     definedBy
          !- ?x memberOf Tourist
  and touristAge(?x,?age)
  and ?age > 24.


concept Service
     nonFunctionalProperties
          dc#description hasValue "unit that provides something which someone
need"
     endNonFunctionalProperties
```

Listing 0.1: Wagenreservierungsontologie, Teil 1.

```
concept Booking subConceptOf Service
     nonFunctionalProperties
          dc#description hasValue "Service which book a car for someone"
     endNonFunctionalProperties
     car impliesType Car
     tourist impliesType Tourist
     pickUpPlace impliesType Destination
     dropOffPlace impliesType Destination
     pickUpDate impliesType BookingDate
     dropOffDate impliesType BookingDate
     pickUpTime impliesType BookingTime
     dropOffTime impliesType BookingTime
     providerName impliesType Provider

concept Car
     nonFunctionalProperties
          dc#description hasValue "means of transport"
     endNonFunctionalProperties
     price impliesType _decimal
     brand impliesType _string
     color impliesType _string
     authorisation impliesType _boolean

concept Tourist
     nonFunctionalProperties
          dc#description hasValue "A person who wish to book a car"
     endNonFunctionalProperties
     firstname impliesType _string
     lastname impliesType _string
     title impliesType _string
     age impliesType _integer
     telephone impliesType _string
     email impliesType _string

concept Destination
     nonFunctionalProperties
          dc#description hasValue "the place designated as the end City of
the tourist"
     endNonFunctionalProperties
     pickUpPlace impliesType City
     dropOffPlace impliesType City

concept City
     nonFunctionalProperties
          dc#description hasValue {"City of the tourist", "concept of a
city"}
     endNonFunctionalProperties
     name impliesType _string
     country impliesType Country
```

Listing 0.2: Wagenreservierungsontologie, Teil 2.

```
concept Country
     nonFunctionalProperties
          dc#description hasValue {"country the tourist", "concept of a
country"}
     endNonFunctionalProperties
     name impliesType _string

concept Provider
     nonFunctionalProperties
          dc#description hasValue "person or a company who provide a car
for reservation"
     endNonFunctionalProperties
     providerName impliesType _string

concept BookingDate
     nonFunctionalProperties
          dc#description hasValue "Period of Booking"
     endNonFunctionalProperties
     pickUpDate impliesType _string
     dropOffDate impliesType _string

concept BookingTime
     nonFunctionalProperties
          dc#description hasValue "Time period of booking"
     endNonFunctionalProperties
     pickUpTime impliesType _string
     dropOffTime impliesType _string


concept EconomyCar subConceptOf Car

concept StandardCar subConceptOf Car

relation cityIsInCountry( ofType City,  ofType Country)
     nonFunctionalProperties
          dc#description hasValue "Relation that holds between a city and
the country it belongs to"
     endNonFunctionalProperties

instance BergTourist memberOf bo#Tourist
      bo#firstname hasValue "Markus"
      bo#lastname hasValue "Berg"
      bo#title hasValue "Mr"
      bo#age hasValue 26
      bo#phone hasValue "+40-444666"
      bo#email hasValue "markus@berg.de"

instance bookingDate1 memberOf bo#BookingDate
      bo#pickUpDate hasValue "2008-06-26"
      bo#dropOffDate hasValue "2008-06-28"
```

Listing 0.3: Wagenreservierungsontologie, Teil 3.

```
instance bookingTime1 memberOf bo#BookingTime
      bo#pickUpTime hasValue "10:00:00"
      bo#dropOffTime hasValue "12:00:00"

instance bookingPlace1 memberOf bo#BookingPlace
      bo#pickUpPlace hasValue Hamburg
      bo#dropOffPlace hasValue Hamburg

instance Hamburg memberOf bo#City
      bo#name hasValue "Hamburg"
      bo#country hasValue bo#Germany
```

Listing 0.4: Wagenreservierungsontologie, Teil 4.

Literaturverzeichnis

[ABB+04] S. Arroyo, C. Bussler, J. de Brujin, R. Lara, M. Moran, M. Stollberg, M. Zaremba, L. Vasiliu, M. Paolucci, K. Sycara, D. Martin, L. Cabral, J. Domingue. *Semantic web services tutorial.* Technical report, International Semantic Web Conference 2004.

[AFJ+05] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth und K. Verma. *Web Service Semantics –WSDL-S.* W3C Member Submission Version 1.0, 2005. http://www.w3.org/Submission/WSDL-S/, März. 2008.

[BBB+05] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, D. L. McGuinness, S. McIlraith, G. Newton, D. De Roure, M. Skall, J. Su, S. Tabet, und H. Yoshida. *Semantic web services framework (SWSF).* W3C Member Submission, 2005. http://www.w3.org/Submission/2005/07/, Apr. 2008.

[BBD+05] J. d. Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, U. Keller, M. Kifer, B. König-Ries, J. Kopecky, R. Lara, H. Lausen, E. Oren, A. Polleres, D. Roman, J. Scicluna und M. Stollberg. *Web Service Modeling Ontology (WSMO).* W3C Member Submission, 2005. http://www.w3.org/Submission/ 2005/06/, Nov. 2007.

[BCF+05] C. Bussler, E. Cimpian, D. Fensel, J.M. Gomez, A. Haller, T. Haselwanter, M. Kerrigan, A. Mocan, E. Oren, B. Sapkota, I. Toma, J. Viskova, T. Vitvar, Ma. Zaremba und Mi. Zaremba. *Web Service Execution Environment (WSMX).* W3C Member Submission, 2005.

[BCH+01] T. Berners-Lee, D. Connolly, S. Hawke, I. Herman, E. Prud'Hommeaux und R. Swick. *W3C Semantic Web Activity.* http://www.w3.org/2001/sw, Jul. 2008.

[BFK+05] J. de Bussler, D. Fensel, U.Keller, M. Kifer, H. Lausen, R. Krummenacher, A. Polleres und L. Predoiu. *Web Service Modeling Language (WSML).* W3C Member Submission, 2005.

[BHL01] T. Berners-Lee, James Hendler und Ora Lasilla. *The Semantic Web.* Scientific American, 2001.

[BMR+05] C. Bussler, M. Moran, D. Roman, M. Stollberg und M. Zaremba. *Semantic Web Services for Autonomic Computing: A Conceptual Model, Language, and Execution Environment.* Tutorial at the 2nd IEEE International Conference on Autonomic Computing 2005.

[CaS06] J. Cardoso und Amit P.Sheth. *Semantic Web Services, Processes and Applications.* Springer, 2006.

[CMO+05] E. Cimpian, M. Moran, E. Oren, T. Vitvar und M. Zaremba. *WSMX Delivery: Overview and scope of WSMX*. WSMX Working Draft, 2005. http://www.wsmo.org/TR/d13/d13.0/v0.2/index.pdf, Jun. 2008.

[DJM+05] W. Dostal, M. Jeckle, L. Melzer und B. Zengler. *Service-orientierte Architekturen mit Web Services: Konzepte-Standard-Praxis.* Spektrum, 2005.

[DLN+96] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. *Reasoning in description logics*. In G. Brewka, editor, Principles of Knowledge Representation, S. 191-236. CSLI-Publications, 1996.

[DoJ04] W. Dostal und M. Jeckle. Semantik, Odem einer Service-orientierten Architektur. In Java Spektrum, 2004. http://www.jeckle.de/semanticWebServices/intro.html, Jul. 2008.

[FLP+07] D. Fensel, H. Lausen, A. Polleres, M. Stollberg, D. Roman, J. Bruijn und J. Domingue. *Enabling Semantic Web Services.* Springer-Verlag Berlin Heidelberg, 2007

[Fre03] J. Frey. *Logische Programmierung: PROLOG.* 2003. http://www.ps.uni-sb.de/courses/seminar-ws03/LogischeProgrammierung.pdf, Mai 2008.

[Fre04] J. Freistätter. *Wissensmanagement als Basis für wissensbasiertes Management: Einführung in die technischen Grundlagen von Wissensbasierten Systemen und die Möglichkeiten der Wissensrepräsentation.* http://www.bmlv.gv.at/pdf_pool/-publikationen/09_vu4_01_wbm.pdf, Apr. 2008.

[GFC04] A. Gómez-Pérez, M. Fernández-lópez und O. Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web.* Springer, 2004.

[GoJ90] R. Golecki und J. Jungmann. *Einführung in die Aussagenlogik. http://lbs.hh.schule.de/oberstufe/ki-al.pdf, Apr. 2008.*

[Gru93] T.R. Gruber. *A translation approach to portable ontology specifications.* Knowledge Acquisition Vol. 5, Nr. 2(1993), S.199-220.

[HaL04] T. Hauser und U. M. Löwer. *Web Services: Die Standards*. Galileo Computing, 2004.

[HFB+00] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer und E. Motta. *The Ontology Inference Layer: OIL*. Technical Report IR-479, Vrije University of Amsterdam, Faculty of Sciences, 2000.

[LaS99] O. Lassila und R. R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification.* W3C Recommendation, 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222, Mär. 2008.

[Mel06] I. Melzer. *Service-orientierte Architekturen mit Web Services: Konzepte-Standard-Praxis.* Spektrum Akademischer Verlag, 2006.

[MBH+04] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. Mcllraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan und K. Sycara. *OWL-S: Semantic Markup for Web services.* 2004. http://www.daml.-.org/services/owl-s/1.1/overview/, Mär.2008.

[MSZ01] S. A. McIlraith, T. Cao Son, und H. Zeng. *Semantic Web Services.* IEEE Intelligent Systems, Special Issue on the Semantic Web, 16(2):46-53, 2001.

[MZM+04] M. Moran, M. Zaremba, A. Mocan und C. Bussler. *Using WSMX to bind requester & provider at runtime when executing semantic web services*. In Proceedings of the Workshop on WSMO Implementations, 2004.

[NFF+91] R. Neches, R. E. Fikes, T. Finin, T. R. Gruber, R. Patil, T. Senator und W. R. Swartout. *Enabling technology for knowledge sharing*. AI Magazine Vol. 12, Nr. 3(1991), S. 36-56.

[PKP+02] M. Paolucci, T. Kawamura, T.R. Payne und K. Sycara. *Semantic Matching of Web Services Capabilities*. In Proceedings of the First International Semantic Web Conference, Sardinia, Italy 2002.

[RLK05] D. Roman, H. Lausen und U. Keller. *Web Service Modeling Ontology (WSMO)*. WSMO Final Draft, 2005. http://www.wsmo.org/TR/d2/v1.1/, Apr. 2008.

[Sch06] Dr.-Ing. Rainer Schönbein. *Wissensrepräsentation mittels Ontologien.* 2006. http://www.iitb.fraunhofer.de/servlet/is/8649/visIT_02_06.pdf, Feb 2008.

[SGA07] R. Studer, S. Grimm, A. Abecker. *Semantic Web Services: Concepts, Technologies and Applications*. Springer, 2007.

[Sow00] J.F. Sowa. *Knowledge Representation: Logical, Philosophical and Computational Foundations.* Brooks/Cole Publishing, 2000.

[StH05] M. Stollberg und A. Haller. *Semantic Web Services Tutorial.* DERI-Digital Enterprise Research Institute. 3rd International Conference on Web Services, 2005.

[UsG96] M. Uschold und M. Gruninger. *Ontologies: Principles, Methods and Applications.* Knowledge Engineering Review, 11(2), S. 93-155, 1996.

[Von02] H. Vonhoegen. *Einstieg in XML.* Galileo Computing, 2002.

[WBF+04] D. Wang, T. Bayer, T. Frotcher und M. Teufel. *Java Web Services mit Axis*. Entwickler. Press, 2004.

[WoJ95] M. Wooldridge und N.R. Jennings. *Intelligent Agents: Theory and Practice.* Knowledge Engineering Review 10(2), S. 115-152, 1995.

[Woo02] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002.

[ZMH+05a] M. Zaremba, M. Moran, T. Haselwanter, H. Lee und S. Han. *WSMX Architecture*. WSMX Working Draft v0.3, 2005. http://www.wsmo.org/TR/d13/d13.4/v0.3/20051012/20051012_d13_4.pdf, Jun. 2008.

[ZMH+05b] M. Zaremba, M. Moran und T. Haselwanter. *WSMX Architecture*, WSMX Final Draft v0.2, 2005. http://www.wsmo.org/TR/d13/d13.4/v0.2/, Jun. 2008.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 10. Januar 2009 _____

Ort, Datum

_____

Unterschrift