Hochschule für Technik und Wirtschaft des Saarlandes
Fachbereich Grundlagen Informatik Sensortechnik (GIS)
Deutsches Forschungszentrum für Künstliche Intelligenz in Saarbrücken

OWLS2WSDL:

Semi-automatische Translation von OWL-S Diensten in WSDL und experimentelle Evaluierung

Vergleichende praktische Untersuchung zur Relation zwischen semantischen und konventionellen Webdiensten in OWL-S und WSDL

Masterarbeit

im Studiengang Kommunikationsinformatik

zur Erlangung des akademischen Grades:
Master of Science in Kommunikationsinformatik (M.Sc.)

vorgelegt von: Oliver Fourman
eingereicht am: 30.3.2007

Betreuer HTWdS: Prof. Dr. Reiner Güttler
Zweitkorrektor: Prof. Dr. Thomas Kretschmer

Betreuer DFKI: Dr. Matthias Klusch
Dipl.-Inf. Ingo Zinnikus

Abstract

Das Konzept der *Serviceorientierten Architektur* (SOA) unterstützt die Erstellung von Anwendungssystemen, die Modellierung von Geschäftsprozessen und die Erstellung von Business Workflows über Orchestrierung und Choreographie einzelner unabhängiger Dienstbausteine. Dafür notwendig ist die Interoperabilität der genutzten Basisdienste und die genaue Beschreibung ihrer Schnittstellen.
Das *Web Services Framework* ist die heute am verbreiteste Technologie um eine SOA zu implementieren. Die Zusammenfassung mehrerer lose gekoppelter Web Services ermöglicht es, Geschäftsprozesse innerhalb einer IT-Landschaft abzubilden.

Eine flexible Möglichkeit der Service-Komposition zur Beschreibung der Geschäftsprozesslogik und die Nutzung von *Web Services* innerhalb eines Verteilten Systems sind auch Hauptanforderungen, um Agentensysteme aufbauen zu können. Agentensysteme erwarten heute keineswegs Dienstbeschreibungen, deren Schnittstellen aufeinander abgestimmt sind, um Dienste auffinden und nutzen zu können. Dies ist ein Aufgabengebiet innerhalb des Forschungsbereichs Deduktion und Multiagentensysteme und der Künstlichen Intelligenz. Man versucht zu gegebenen Prozessbausteinen und Service-Schnittstellen innerhalb einer SOA möglichst ähnliche Schnittstellen zu finden, um Business-Prozessplanungen durchführen zu können, und um die Bildung von Ad-hoc-Service-Prozessketten und Service-Kompositionen (Ad-hoc-Workflows) zu unterstützen. Alternativ zu der Nutzung von Verzeichnisdiensten wie UDDI werden *Matchmaker* Technologien eingesetzt, um Web Services aufzufinden. Das *Semantic Web* ermöglicht die Definition von *Semantic Web Services* (z.B. in OWL-S), die das Auffinden durch den Einsatz von semantischen Matchmakern unterstützen. State of the Art Matchmaker kombinieren auch konventionelle Algorithmen aus dem Bereich des Information-Retrieval (IR) mit semantischem Matchmaking.

Die im Rahmen dieser Arbeit vorgestellte Translation *OWL-S nach WSDL* ermöglicht den Vergleich von zwei Matchmakern, die mit jeweils einer der Technologien arbeiten. Anhand der übersetzten Testkollektion OWLS-TC werden die beiden am DFKI entwickelten Matchmaker *OWLS-MX* und *WSDL Analyzer* verglichen.


OWL-S
Translation
to WSDL
OWLS2WSDL

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Beschreibung des Themas

Das Thema wurde im Forschungsbereich Deduktion und Multiagentensysteme (DMAS) am Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) in Saarbrücken bearbeitet.

Thema

OWLS2WSDL:
Semi-automatische Translation von OWL-S Diensten in WSDL und experimentelle Evaluierung
(Vergleichende praktische Untersuchung zur Relation zwischen semantischen und konventionellen Webdiensten in OWL-S und WSDL)

Ziele

1. Vergleichende Untersuchung der Möglichkeiten einer praktischen Generierung von OWL-S Diensten aus WSDL Diensten und, vor allem, umgekehrt, d.h. ihrer Konkretisierung in Form von WSDL-Groundings.

2. Semi-automatische Translation von OWL-S nach WSDL über ein zu erstellendes Tool namens owls2wsdl.

3. Vergleichende Analyse der Performanz einer ähnlichkeitsbasierten Dienstsuche mit WSDL-Analyzer (aus ATHENA Projekt) und OWLS-MX (aus SCALLOPS Projekt) über eine entsprechend aufgebaute Testkollektion von semantischen Webdiensten mit Konkretisierungen in WSDL.

Bemerkungen

Allgemein gefasst ist das Thema der Arbeit, den bisher in der Forschung wenig berücksichtigten Bereich des Groundings von OWL-S Diensten (semantisches Web) in WSDL-Beschreibungen (konventionelles Web) von konkreten Diensten (Software) zu bearbeiten. In einem ersten Schritt soll ein Tool entwickelt werden, mit dessen Hilfe aus einer OWL-S Beschreibung eines Dienstes eine adäquate Dienstbeschreibung in WSDL semi-automatisch generiert werden kann. Die Software soll (GPL) offen zugänglich gemacht werden. Für die umgekehrte Richtung gibt es z.B. ein Tool wsdl2owl-s (s.a. semwebcentral.org). Dabei liegt der Fokus auf einer vergleichenden Analyse möglicher Translationsarten mit Vor- und Nachteilen. Die vergleichende Untersuchung und Selektion einer geeigneten Translationsart von OWL-DL (für formale Semantik von Konzepten in OWL-S Dienstbeschreibungen) nach XSD für WSDL (über ein geeignetes konzeptionelles Datenmodell) ist dabei der theoretische Kern der Arbeit und Grundlage für das zu erstellende Tool owls2wsdl. Im Anschluss daran sollen in begrenztem Umfang verschiedene Evaluierungsschritte mit diesem Tool durchgeführt werden. Insbesondere soll abhängig von der Art der Generierung der Groundings von OWL-S Diensten in WSDL überprüft werden, inwiefern sich die Ergebnisse eines rein strukturellen ähnlichkeitsbasierten Vergleichs der generierten WSDL-Groundings durch den WSDL-Analyzers zu den Ergebnissen der hybrid semantischen Suche über der korrespondierenden Menge der semantischen Dienste durch den OWLS-MX Matchmaker verhalten. Beide Tools, OWLS-MX und WSDL-Analyzer werden im Source Code zur Verfügung gestellt.

Betreuer am DFKI

Dr. Matthias Klusch
Tel: +49-681-302-5297
Fax: +49-681-302-2235
http://www.dfki.de/~klusch/
E-Mail: klusch@dfki.de

Dipl.-Inform. Ingo Zinnikus
Tel: +49-681-302-5362
E-Mail: ingo.zinnikus@dfki.de

Kapitel 1

Einführung

Das Kapitel beschreibt das Umfeld, in dem das Thema dieser Abschlussarbeit angesiedelt ist. Vorgestellt werden die beiden Projekte ATHENA *und* SCALLOPS. *Beide sind im Forschungsbereich* Deduktion und Multiagentensysteme *am* DFKI *angesiedelt und beschäftigen sich mit der Interoperabilität von Diensten im* SOA Umfeld. *Unterschiedlich ist, dass* ATHENA, *obwohl die Projektbeschreibung sehr allgemein gehalten ist, in erster Linie mit konventionellen Service-Beschreibungen (WSDL) arbeitet und* SCALLOPS *im Bereich Orchestrierung von semantischen Servicebeschreibungen (OWL-S) angesiedelt ist. Die Translation von OWL-S nach WSDL verknüpft beide Projekte miteinander und ermöglicht einen Vergleich von genutzten Matchmakern.*

ATHENA - Advandced technologies for interoperability of heterogenous enterprise networks and their applications

Abbildung 1.1: ATHENA Banner

ATHENA-IP[1] ist eine im Rahmen des strategischen Ziels **Networked businesses and government**[2] von der Europäischen Kommission[3] geförderte Forschungsinitiative (IST[4] 2003-2004 Arbeitsprogramm FP6, Laufzeit 2004 - 2007) mit einem Budget von 26,6 Millionen Euro (14,4 Millionen durch die EU finanziert).

Ziel des Projekts ist die Verbesserung der Interoperabilität von Basistechnologien. Teilgebiete aus der Projektbeschreibung sind: (1) Anwendung und Services, (2) Forschung und Entwicklung, (3) Demonstration und Testen, (4) Schulungen sowie die (5) Evaluation von Technologien bzgl. ihrer gesellschaftlichen Auswirkungen. ATHENA verknüpft die Forschung und Entwicklung sehr stark

[1] http://www.athena-ip.org
[2] http://cordis.europa.eu/ist/so/business-govt/
[3] http://ec.europa.eu
[4] Information Society Technologies

mit dem *Community Building* Prozess (Bildung von Synergien), um zu gewährleisten, dass die Ergebnisse multidisziplinärer Forschungsvorhaben von Seiten der Industrie optimal und maßgeblich auf eine deutliche Akzeptanz bei den Endanwendern treffen.
"ATHENA beabsichtigt die umfassendste und systematischste europäische Forschungsinitiative für IT zu sein, mit dem Ziel die Grenzen der Interoperabilität zu brechen, den Austausch von Forschungsergebnissen innerhalb von Industriesektoren zu motivieren und eine neue netzwerkorientierte Geschäftskultur zu fördern." Aufbauend auf der visionären Aussage

> *"By 2010, enterprises will be able to seamlessly interoperate with others"*

ist das definierte Ziel von ATHENA-IP, Interoperabilität durch die Bereitstellung von Referenzarchitekturen, Methoden und Infrastrukturkomponenten zu ermöglichen. Forschung wird durch die Geschäftsanforderungen bestimmt, die durch einen breiten Bereich von Industriesektoren definiert sind. ATHENA soll als Lieferant für technische Erfindungen zur Unterstützung der Interoperabilität dienen.[5]
Das ATHENA Konsortium besteht aus 27 führenden Organisationen aus den Bereichen Forschung, Lehre, Industrie und weiteren Stakeholdern aus anderen Bereichen, um ein möglichst breites Spektrum an Zielvorgaben der Interoperabilität abdecken zu können.

SCALLOPS - Secure Agent-Based Pervasive Computing

"Das Forschungsprojekt SCALLOPS[6] (BMBF[7] Project 2004 - 2007) zielt auf die Bereitstellung von innovativen Methodiken, Techniken und Werkzeugen für die Entwicklung und intelligente Koordination von sicheren semantischen Webdiensten einer hochgradig vernetzten Informationsgesellschaft. Zentrale Forschungsthemen von SCALLOPS sind die Entdeckung, Verhandlung und Komposition von semantischen Webdiensten durch intelligente Agenten in pervasiven Umgebungen unter gleichzeitiger Wahrung von Integrität und Vertraulichkeit von Informationen."

Abbildung 1.2: SCALLOPS Logo

"SCALLOPS besitzt ein enormes ökonomisches Potential. Zum einen liefert es innovative Beiträge zur Überwindung einer der wichtigsten Barrieren gegen die Ausschöpfung des Marktpotentials des Webs: Entwicklung und intelligente Koordination sicherer, vertrauenswürdiger Dienste. Zum anderen unterstützt SCALLOPS den sich in Deutschland im Aufbau befindlichen Bereich der *Telematik* im Gesundheitswesen, deren Anwendungen höchste Anforderungen an die Vertraulichkeit und Integrität der Daten stellen." [8]

[5] http://www.dfki.de/web/forschung/projekte?pid=18
[6] http://www.dfki.de/scallops
[7] http://www.bmbf.de
[8] http://www.dfki.de/web/forschung/projekte?pid=192

1.1 Gliederung der Arbeit

Das Kapitel GRUNDLAGEN ab Seite 6 versucht, diese Arbeit thematisch innerhalb des Konzepts der *Serviceorientierten Architektur* (SOA) einzuordnen. Vorgestellt wird das *Web Service Framework* und die *Semantic Web Services* Technologie. Eine ausführliche Beschreibung der SOA würde den Rahmen dieser Ausarbeitung sprengen.

Das Kapitel ANALYSE DER PROBLEMSTELLUNG (Kapitel 3) ab Seite 24 beginnt mit der Einordnung der Aufgabenstellung. Es werden zentrale Problemstellungen bzgl. der Übersetzung von OWL-S Definitionen in WSDL Beschreibungen herausgearbeitet und der Begriff *Translation* erläutert. Dabei hilft die Vorstellung verwandter Arbeiten aus dem Bereich *Semantic Web Services*. Das Kapitel endet mit einem Pflichtenheft.

In dem Kapitel KONZEPTIONIERUNG DER TRANSLATION (Kapitel 5) werden die Anforderungen aus dem Kapitel ANALYSE ausgearbeitet und die Translation vorgestellt.
Das Kapitel VORGEHENSMODELL (Kapitel 4) auf Seite 36 hilft, den Entwicklungprozess der Translation und des Tools zu verstehen.

Die Beschreibung der IMPLEMENTIERUNG (Kapitel 6) ab Seite 51 stellt die implementierten Bausteine des Tools OWLS2WSDL vor und arbeitet die Teilaspekte der Translation heraus. Die erstellten Programmteile sind: der Parser, ein Persistenzlayer für die Datenhaltung, der Codegenerator und die grafische Benutzeroberfläche. Die Oberfläche dient der Verwaltung von Datentypen und Service-Informationen und unterstützt die semi-automatische Translation von OWL Klassen nach XML Schema. Abschließend werden in FUTURE WORK (Kapitel 6.9) mögliche Arbeitspakete im Anschluss an diese Arbeit vorgestellt, die den Leistungsumfang des implementierten Tools erweitern.

Das BENUTZERHANDBUCH ab Seite 67 beschreibt die Arbeitsweise des erstellten Tools. Zusätzlich wird die Vorgehensweise zur Übersetzung der OWLS-TC Testkollektion nach WSDL vorgestellt, um den Vergleich der beiden Matchmaker *OWLS-MX* und *WSDL Analyzer* zu ermöglichen. Anschließend werden in Kapitel 8 generierte WSDL Beschreibungen analysiert und die berechnete Ähnlichkeitswerte verglichen.

Die EXPERIMENTELLE EVALUIERUNG (Kapitel 8) prüft die implementierte Translation durch den Vergleich von Resultaten der beiden Matchmaker *OWLS-MX* und *WSDL Analyzer*. Als Datenbasis dient die übersetzte OWLS-TC Testkollektion.

Die ZUSAMMENFASSUNG (Kapitel 9) beschreibt noch einmal das Thema. Diskutiert wird der Unterschied zwischen semantischem und syntaktischem Matchmaking mit Bezug auf die Ergebnisse aus Kapitel 8.

1.2 Voraussetzungen

Architektur

Vorausgesetzt werden Kenntnisse aus dem Bereich anwendungsorientierter Middleware, um das Vorgehensmodell und den Aufbau einer *Service Orientierte Architektur* (SOA) zu verstehen. Das Architekturmodell SOA wird im Kapitel GRUNDLAGEN kurz vorgestellt. Hauptanwendung der SOA Infrastruktur ist die Modellierung und Implementierung von Geschäftsprozessen innerhalb eines Verteilten Systems. SOAs basieren auf kommunikationsorientierter Middleware wie dem Web Service Framework (XML, XSD, WSDL, SOAP, UDDI).

Von Vorteil sind auch Kenntnisse des Konzepts der *Modellgetriebenen Architektur* (MDA). Implementiert wurde die **Translation OWL-S nach WSDL**, d.h. Definitionen einer Beschreibungssprache (OWL-S) werden in Definitionen einer anderen Beschreibungssprache (WSDL) übersetzt. Teilaufgaben der Translation sind das **Parsen von OWL und OWL-S** und die **Quellcode Generierung von XML Schema und WSDL**. Der Begriff der *Translation* unterscheidet sich von dem Begriff der *Transformation* in der Art, dass das Ursprungsmodell und dessen Informationsgehalt möglichst gut interpretiert wird. Eine *Transformation* erzeugt eine genaue Abbildung des Ursprungsmodells. Weiterhin wurde ein **Metamodell** als Wissensbasis für die Generierung von WSDL und XML Schema eingeführt. Der Gebrauch des Begriffs *Metamodell* ist im Rahmen dieser Arbeit nur angelehnt an die Bedeutung des Begriffs im Kontext einer modellgetriebenen Architektur (MDA), bei der man zwischen struktureller (z.B. UML) und inhaltlicher Metamodellierung unterscheidet [AK03]. Die genutzten Metamodelle für **Service**, **Datentyp** und **Matchingprojekt** beinhalten die geparsten (OWL-S) bzw. geparsten und interpretierten (OWL) Informationen aus den Ursprungsmodellen inklusiver zusätzlicher neu gebildeter Metadaten. Als Beispiel für den Nutzen des Metamodells kann die Unterscheidung der OWL-S Spezifikation in OWL-S 1.0 und OWL-S 1.1 aufgeführt werden. Eingelesen und persistent gespeichert werden die Daten aus beiden Versionen plus die Information über die jeweilig genutzte Version.

Implementierung

Der Implementierungsteil setzt Grundkenntnisse aus den Bereichen der Objektorientierten Programmierung, persistente Speicherung, XML, XML Schema und Web Services (WSDL) voraus. Die Implementierung wurde mit Java durchgeführt, da die meisten Technologien aus dem Bereich *Semantic Web Services* auf Java basieren. Im Rahmen der Ausarbeitung wird auf den Java Quellcode verzichtet. Der Quellcode des implementierten Prototyps ist *Open Source* und wurde unter der *GNU General Public License* veröffentlicht.

Projektseite: `http://code.google.com/p/owls2wsdl/`

1.3 Konventionen

1.3.1 Begriffe und Wortwahl

Bei vielen englischen Bezeichnungen aus aktuellen Bereichen der Informatik ist es schwierig, ihre Bedeutung adäquat ins Deutsche zu übersetzen. In vielen Fällen nutzt man die deutsche Übersetzung und den entsprechenden Anglizismus auch gleichwertig. Folgende Tabelle zeigt einige Anglizismen, die in dieser Arbeit parallel zur Übersetzung benutzt werden.

ANGLIZISMUS	BEDEUTUNG, ÜBERSETZUNG
Inference (Engine)	Deduktion, Schlussfolgerung (Programmteil)
Matchmaking, Matchen	Untersuchung auf Ähnlichkeit
Matchmaking Candidate (Service)	Dienst, der gegen eine Referenz geprüft wird
Matchmaking Requirement (Service)	Referenzdienst
Message (WSDL, SOAP)	Nachricht (Protokoll)
Parsen	Auslesen von Daten
Pattern	Entwurfsmuster
Property (OWL)	Eigenschaft
Query (Service)	Anfrage (Referenzdienst)
Ranking	Auflistung von Ergebniswerten
Result (result set)	Resultat, Ergebnis (Menge)
Service Oriented Architecture (SOA)	Service- oder Dienstorientierte Architektur
Service	Dienst
Service Discovery	Dienstauffindung
Service Invocation	Dienstausführung
Translation	Übersetzung oder auch Translation
Web Service	Webdienst
Web Services	Technologiebereich

1.3.2 Typographische Konventionen

Diese Ausarbeitung wurde mit LaTeX erstellt.
Um Textstellen hervorzuheben, wurden folgende Hilfsmittel benutzt:

- Wichtige Bezeichnungen, Technologien und neu eingeführte Abkürzungen werden durch den Gebrauch von *emphasize* hervorgehoben.
- Englische Bezeichnungen werden *kursiv* dargestellt.
- Anglizismen werden nicht zwingend *kursiv* dargestellt.
- Quellcode wird in `monospace` bzw. `Typewriter` dargestellt.
- Bezeichnungen und Klassen (OWL, Java) werden serif hervorgehoben.
- Wichtige Begriffe in einem Kontext werden **fett** geschrieben.

Kapitel 2

Grundlagen

2.1 SOA

Der Begriff der *Serviceorientierten Architektur* (SOA), auch als *Dienstorientierte Architektur* bezeichnet, beschreibt ein Managementkonzept, das auf ein entsprechendes Systemarchitekturkonzept aufsetzt. Im Vergleich zu Middleware- und ERP-Systemen betont eine SOA die lose Kopplung von beteiligten Einheiten innerhalb eines Verteilten Systems.

Managementkonzept Das Managementkonzept strebt eine, an den Geschäftsprozessen ausgerichtete, Infrastruktur an, die schnell auf sich verändernde Anforderungen im Geschäftsumfeld reagieren kann.

Systemarchitekturkonzept Das Systemarchitekturkonzept sieht die Bereitstellung fachlicher Dienste und Funktionalitäten in Form von Services vor.

Eine SOA beinhaltet die Grundsätze *eindeutige Grenzen*, *autonome Diensteinheiten*, *eindeutig deklarierte Schnittstellen* und *Datenformate* sowie *einheitliche Dienstbeschreibungen* [VBE+].

Im Bereich von Verteilten Systemen wird das SOA Konzept immer wichtiger. Beobachtet man den aktuellen Markt (CeBit 2007) in den Bereichen betrieblicher EDV kann man die SOA bereits als Mainstream bezeichnen. Immer mehr IT-Dienstleister bieten SOA innerhalb ihres Produkt Portfolios an. “Accenture etwa steckt in den nächsten drei Jahren 450 Millionen Dollar in seine SOA-Bemühungen, HP will 500 Millionen Dollar investieren. IBM hat laut eigenen Angaben allein im Jahr 2006 eine Milliarde Dollar in die Entwicklung von SOA-Lösungen gepumpt.” ... “Mit den Gestaltungsoptionen steige der Bedarf an Strategie- und Prozessberatung. Nach Erhebungen von IDC gaben Kunden im Jahr 2006 weltweit bereits rund 8,6 Milliarden Dollar für IT-Dienstleistungen in SOA-Projekten aus. Bis 2010 soll die Summe auf 33,8 Milliarden Dollar steigen. Das Umsatzpotenzial für SOA-Produkte und -Services schätzt IBM-Manager Ray Harishankar auf 160 Milliarden Dollar im Jahr 2008. Sind diese Zahlen auch nur halbwegs realistisch, dann entwickelt sich SOA zumindest für einige Anbieter zum tragfähigen Geschäftsmodell.” (Computerwoche.de, Januar 2007)

2.1.1 Eigenschaften und Merkmale des Systemarchitekturkonzepts

- Ein Service bietet innerhalb der SOA eine Funktionalität, die über eine standardisierte Schnittstelle in Anspruch genommen werden kann. Innerhalb eines Softwareprojekts kann die Funktionalität ähnlich wie eine Softwarekomponente benutzt werden.
- Durch Aneinanderreihung und Komposition von Services lassen sich Geschäftsprozesse abbilden. Die Programmlogik ist über mehrere Dienstbausteine verteilt.
- Ein Service ist als *Black Box* realisiert. Bekannt ist lediglich die Schnittstellenbeschreibung, nicht die Implementierung.
- Services bieten ihre Funktionalität unabhängig von einer bestimmten Plattform an. Vereinbart wird lediglich das Kommunikationsprotokoll.
- Die Funktionalität eines Web Services kann über die Registrierung in dem Verzeichnisdienst UDDI beschrieben werden.

Merkmale des Systemarchitekturkonzepts

- Lose Kopplung der Komponenten
- Dynamisches Binden von Services zur Laufzeit
- Verzeichnisdienst zum Auffinden von Services zur Laufzeit
- Offene Standards zur Beschreibung der Service-Semantik
- Services sind komponierbar und orchestrierbar

2.1.2 State of the Art

Heute finden sich innerhalb einer SOA folgende Hauptkonzepte: *Web Services* (WSDL), *Business Process Modeling* (z.B. BPEL4WS[1]), *Business Automation*, Prozessplanung, Komposition und Koordination in Form einer Orchestrierung[2] oder Choreographie[3], Agentensysteme, *Service Negotiation*, *Service Discovering*, Software Generierung, *Semantic Web* und *Web 2.0*.

Man spricht dann von der Komposition von Diensten, wenn mehrere unterschiedliche Dienste dazu genutzt werden, eine bestimmte Aufgabenstellung zu erfüllen. Dazu werden in der Regel Geschäftsprozessketten gebildet, bei denen die Rückgabewerte einzelner oder mehrerer Dienste als Eingabeparameter weiterer Dienste genutzt werden. Der Vorgang wird als Kompositionsplanung (engl. *composition planning*) bezeichnet. So zusammengebaute Anwendungen werden als *Composite Applications* bezeichnet.

[1] http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
[2] Orchestrierung: Aufruf und Zusammenfassung von mehreren firmeneigenen Web Services. Häufig bereits bekannt in Zusammenhang mit EAI Middleware.
[3] Choreographie: Unternehmensübergreifender Aufruf von Web Services und Informationsfluss

2.1.3 Das Web Services Framework

Ein **Web Service** ist eine lose gekoppelte Softwarekomponente, die einem Klienten *(engl. service requestor)* über ein Intranet, Extranet oder das Internet eine bestimmte Funktionalität anbietet. Dabei genutzte Web-Standards sind HTTP, SMTP, XML, SOAP, WSDL und UDDI. Ein **Framework** kennzeichnet eine Sammlung von Dingen. Dazu zählen die Beschreibung der Architektur, die genutzten Technologien sowie Konzepte und Modelle.

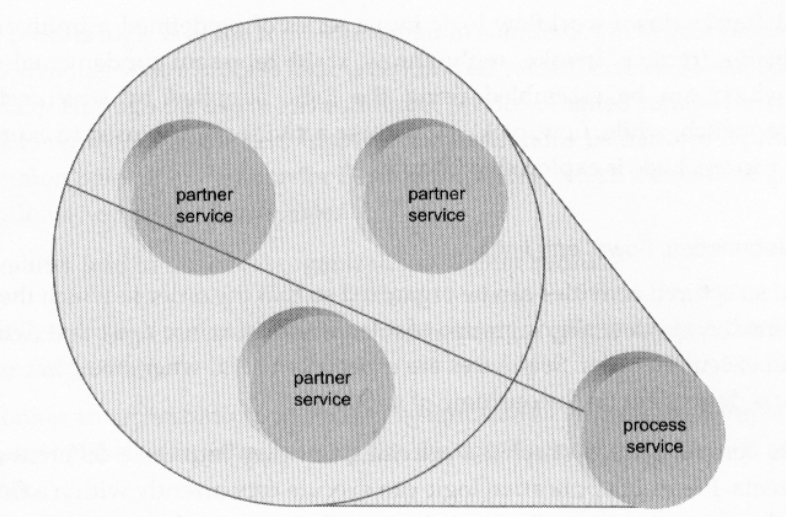


Abbildung 2.1: Komposition von Services [Erl05]

Das *Web Services Framework* wird nach [Erl05] durch folgende Eigenschaften charakterisiert:

- Es ist ein abstraktes, herstellerunabhängiges Framework, das durch Organisationen wie das W3C[4], OASIS[5] und WS-I[6] standardisiert und auch erweitert wird.
- Grundbausteine sind Web Services.
- Web Services können mit proprietären Technologien und Plattformen realisiert werden.
- Das Framework basiert auf Service Beschreibungen und dem Austausch von Nachrichten.
- Der Nachrichtenaustausch ist plattformneutral.
- WSDL ist das genutzte Kommunikationsagreement.
- Das genutzte Messaging Framework ist SOAP.
- UDDI ist Registry für Servicebeschreibungen und damit Discovery Architektur.
- Das Framework ermöglicht die Komposition von Service-Definitionen.
- Die zweite Generation von Web Services (WS-* Spezifikation) erweitert das Framework um ein grundlegendes Feature-Set.

[4] http://www.w3.org
[5] http://www.oasis-open.org
[6] http://www.ws-i.org

Aufbau einer SOA mit Web Services

- Durchführung einer service-orientierten Analyse.
- Modellierung wiederverwendbarer Web Services zur Beschreibung der Geschäftsprozesse.
- Komposition der Web Services im Sinne einer SOA (Orchestrierung).
- Design und Abbildung der Geschäftsprozesslogik (z.B. mit BPEL4WS).
- Evaluierung von SOA Plattformen wie J2EE oder .NET.
- Implementierung der Anwendung entsprechend der Web Service Definition (WSDL).

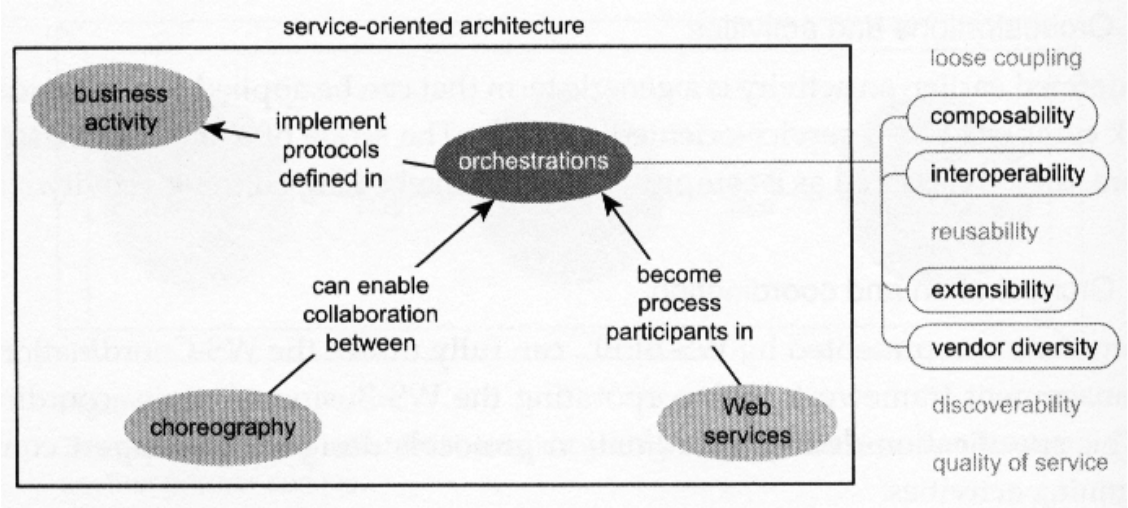


Abbildung 2.2: Orchestrierung innerhalb einer SOA [Erl05]

Tasks eines Agentensystems, das mit der SOA arbeitet:

- Lokalisierung von Web Services *(engl. discovery)*.
- Ausführung eines Dienstes durch einen Agenten *(engl. invocation)*.
- Zusammenarbeit mehrerer Services *(engl. interoperability)*.
- (Automatisierte) Zusammenstellung mehrerer Dienste *(engl. composition)*.
- Verifikation der Service-Beschreibung.
- Beobachtung der Ausführung *(engl. monitoring)*.

2.1.4 Syntaktische Beschreibung von Web Services (WSDL)

Die durch das W3C standardisierte Sprache zur Beschreibung von Web Services ist die *Web Services Description Language*[7] (WSDL). WSDL ist eine plattform-, programmiersprachen- und protokollunabhängige XML-Spezifikation zur Beschreibung von Web Services. Die Beschreibung enthält Informationen zu der Service-Schnittstelle (Signatur) und Informationen zu dem verwendeteten Austauschprotokoll, um mit dem Dienst kommunizieren zu können (z.B. SOAP mit Transportprotokoll HTTP). In der Beschreibung der Schnittstelle werden die Operationen eines Dienstes definiert. Eine Operation besteht jeweils aus einer Eingabe- und/oder Ausgabe-Nachricht *(request, response)*. WSDL Nachrichten bestehen wiederum aus Parametern, die entweder als Eingabeparameter oder als Rückgabeparameter dienen. Für die Beschreibung der Signatur wichtig sind die Parametertypen.
Nicht enthalten in einer konventionellen WSDL Beschreibung sind:

- Zusätzliche Informationen zu dem Dienst und dem Anwendungskontext (siehe UDDI).
- Informationen zu einem Quality-of-Service (QoS).
- Taxonomien oder Ontologien zur semantischen Einordnung des Dienstes.

Zur semantischen Einordnung von Web Services gibt es zur Zeit zwei Ansätze:
Mit WSDL-S[8] wurden Erweiterungen für WSDL definiert, mit denen man die Parameter der Schnittstelle und Bedingungen, die das Workflow (Vor- und Nachbedingungen) betreffen, mit Konzepten eines Domainenmodells assoziieren kann. Der zweite Ansatz ist die Definition einer unabhängigen semantischen Service Beschreibung in OWL-S (Kapitel 2.2). Die WSDL Beschreibung bleibt unverändert und wird über das WSDL-Grounding (Abschnitt 2.2.4) der OWL-S Definition referenziert.

Syntaktische Matchmaker

Ein Matchmaker vergleicht Service-Beschreibungen miteinander und kann Aussagen über deren Ähnlichkeit treffen. Innerhalb eines Verteilten Systems sind solche Informationen sehr wichtig, um Aussagen darüber treffen zu können, inwieweit Dienste zueinander passen. Dabei vergleicht der Matchmaker die Beschreibung eines Referenzdienstes *(engl. requirement)* mit einer Menge von vorhandenen Dienstbeschreibungen *(engl. candidates)*. Die Analyse der Matchmakingwerte kann als Alternative zu der Nutzung eines Verzeichnisdienstes (z.B. UDDI) angesehen werden. Es geht darum, Services aufzufinden *(engl. discovering)* und miteinander zu komponieren *(engl. composition)* (Abbildung 3.1, Seite 24).
Es gilt: Selbst wenn Service-Beschreibungen nicht hundertprozentig kompatibel sind, können Matchmakingwerte hilfreich sein. Wird ein Service gefunden, bei dem nur ein Teil seiner Schnittstelle matcht, kann dieser Teil der Schnittstelle genutzt werden. Ein Beispiel ist das Fehlen eines

[7] http://www.w3.org/TR/wsdl
[8] http://www.w3.org/Submission/WSDL-S/

Parameters bei einer Suchanfrage, der die Ergebnismenge weiter einschränkt. Bei der Anwendungsentwicklung innerhalb einer SOA lassen sich anhand des Matchmakingwerts zu einem Referenzdienst die Kosten für die Entwicklung eines Adapters vorhersagen, der nötig wird, um den Dienst vollständig nutzen zu können. Ein am DFKI entwickelter syntaktischer Matchmaker ist der *WSDL Analyzer*. Er wird in Abschnitt 8.1.1 näher vorgestellt.

2.2 Semantic Web Services

2.2.1 Wissensrepräsentation über Ontologien

Was sind Ontologien? Ursprünglich kommt der Begriff der Ontologie aus der Philosophie und dort aus dem Gebiet der *Seinslehre*, einer philosophischen Disziplin, die auch Teil der Metaphysik ist. Dementsprechend beschreibt eine Ontologie einen *Seinszusammenhang*. Mit der Idee des *Semantic Web* hat der Begriff der Ontologie in den letzten Jahren einen Aufschwung erfahren und wird auch im Bereich der Informatik und Computerlinguistik benutzt. Die wohl bekannteste und gleichzeitig kürzeste Definition stammt von Tom Gruber:

> "An ontology is a specification of a conceptualization." Gruber, [Gru94]

Gruber bezeichnet Ontologien als *explizite formale Spezifikation einer gemeinsamen Begriffsbildung*. Die deutsche Übersetzung von *concept* sollte also als *Begriff* verstanden werden.

Eine Ontologie ist vergleichbar mit einem UML Klassendiagramm: Sie modelliert Begriffe (Klassenkonzepte), deren Eigenschaften sowie die Beziehungen zwischen den Begriffen. Häufig spricht man an dieser Stelle auch von *Taxonomien*. Eine *Taxonomie* beschreibt Klassen und deren Beziehungen als Hierarchie. Alesso und Smith [AS05] definieren den Zusammenhang zwischen einer Ontologie und einer Taxonomie wie folgt:

```
taxonomy = <{classes}, {relations}>
ontology = <taxonomy, inference rules>
```

Innerhalb der Informatik dienen Taxonomien der *Wissensrepräsentation* eines formal definierten Systems von Begriffen und Relationen. Zusätzlich enthalten Ontologien Inferenzregeln, die Schlussfolgerungen ermöglichen (engl. reasoning), und Integritätsregeln.[9]

Das Semantic Web

Die heutige Bedeutung des Begriffs *Semantic Web* wurde Ende der Neunziger im Rahmen einer Veröffentlichung für das W3C von Tim Berners-Lee[10] geprägt [BL98]. Gesucht wurden Möglichkeiten der Wissensrepräsentation im World Wide Web über maschinenlesbare Annotationen, die von Agentensystemen verstanden werden konnten.

> "Netz von Daten, die direkt und indirekt von Maschinen verarbeitet werden können."
> (Tim Berners-Lee)

[9] Zusammengefasst aus Wikipedia: Ontologie (Informatik)
[10] Berners-Lee, Direktor des World Wide Web Consortium, http://www.w3.org/People/Berners-Lee/card

Heute versteht man unter dem Begriff des *Semantic Web* ein auf RDF basierendes Framework, das es erlaubt, Daten innerhalb von Verteilten Systemen über Applikations-, Geschäfts- und Community Grenzen hinweg auszutauschen.[11] Ontologien beschreiben innerhalb des *Semantic Web* eine Menge von Begriffen und Beziehungen als Vokabular für den Informationsaustausch.

2.2.2 Bedeutung von Semantik innerhalb einer SOA

Für die Implementierung einer SOA ist es von Bedeutung, dass Services sich beliebig komponieren lassen, solange die Schnittstellenbeschreibung einer Service Beschreibung (z.B. WSDL) dies zulässt. Die Koordination der Kompositionen und die Validierung von Schnittstellen, d.h. die Prüfung der Interoperabilität, von Services und deren Schnittstellen bleibt der Anwendung bzw. dem Entwickler überlassen. Die semantische Beschreibung eines Dienstes ist Schlüssel, um mehrere Dienste automatisiert aufzufinden, mit ihnen zu kommunizieren und sie zu komplexeren Anwendungen, die einen Geschäftsprozess realisieren, komponieren zu können. Durch Integration des *Semantic Web* lassen sich nun Services und deren Daten semantisch einordnen (siehe Abschnitt 2.2). Wie wichtig die Integration der Ansätze des *Semantic Web* innerhalb der SOA sind zeigen aktuelle EU Projekte wie das europäischen Forschungsprojektes *Adaptive Services Grid*[12] (ASG), das versucht eine neuartige Softwareplattform zu entwickeln, die es ermöglichen soll, auf der Basis von semantischen Beschreibungen Servicekomponenten teilautomatisiert und fehlertolerant zu komplexen Serviceprozessen zusammenzubauen.

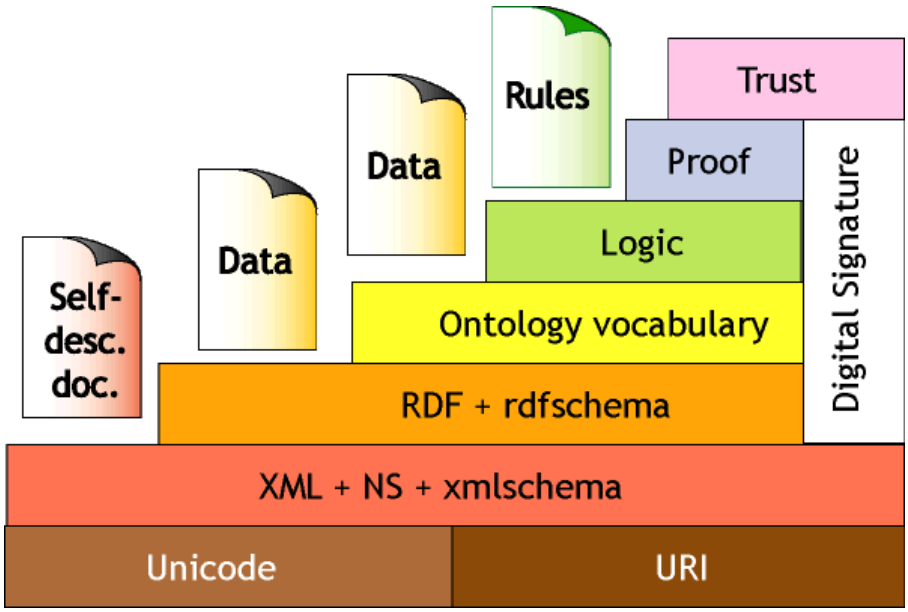


Abbildung 2.3: Semantic Web Layer Cake (Berners-Lee)

Die Abbildung 2.3 zeigt den *Semantic Web Layer Cake*, der die Technologien des *Semantic Web* einordnet und ihre Funktionalität beschreibt.

[11] http://www.w3.org/2001/sw/
[12] http://asg-platform.org

2.2.3 Die Web Ontology Language (OWL)

Die in dieser Arbeit genutzten Ontologien liegen in dem *Web Ontology Language* (OWL) Format[13] vor,; dies ist eine Internetspezifikation des W3C. OWL, als formale Beschreibungssprache ermöglicht es, Ontologien zu erstellen und über das Web zu verteilen (publizieren). Ziel ist es, Begriffe und deren Beziehungen in maschinenlesbarer Form so zu beschreiben, dass Software-Agenten die Bedeutung verarbeiten bzw. verstehen können. OWL basiert technisch auf der *RDF-Syntax*, einem Datenmodell, um Ressourcen und Beziehungen zwischen Ressourcen zu beschreiben und RDF Schema (RDF-S), einem Vokabular, um RDF-Ressourcen innerhalb einer Hierarchie anzuordnen. Zusätzlich zu RDF und RDF-S, wurden mit DAML-OIL und später OWL weitere logische Sprachkonstrukte eingeführt, die es erlauben, Ausdrücke ähnlich der Prädikatenlogik (für OWL-DL Beschreibungslogik) zu formulieren, um so das logische Schlussfolgern mit *Reasonern (Inference Engines)* zu ermöglichen. Historisch entstanden ist OWL neben RDF aus DAML+OIL. Dieser *Ontology Inference Layer* geht weit über die Ausdrucksmächtigkeit von RDF-S hinaus.

Sprachumfang von OWL

Eine OWL Definition besteht aus *Klassen*, *Eigenschaften*, *Restriktionen* und *Individuen*.

OWL Klasse OWL Klasse (`owl:class`) beschreibt ein Begriff innerhalb einer Ontologie. Über die Definition von Unterklassenbeziehungen (`rdfs:subClassOf`) lassen sich Klassen in einer Hierarchie anordnen. Ontologien erweitern Taxonomien (siehe Abschnitt 2.2.1) durch die Angabe von mehreren Eltern. Klassen können durch Änderung von Eigenschaften (*engl. properties*) bereits definierter Klassen, beispielsweise durch Restriktion (*engl. restrictions*) des Wertebereichs und Änderung von Kardinalitätsangaben neu gebildet werden. Weiterhin können Mengenbeziehungen genutzt werden, um anonyme Klassen zu definieren (siehe Abschnitt 5.4.1). Eine Instanz einer OWL Klasse wird Individual genannt.

Eigenschaften Eigenschaften (OWL Properties) beschreiben Beziehungen zwischen Klassen in Tripleform (RDF).

Domain (Subjekt), **Property** (Prädikat), **Range** (Objekt).

Für eine Klasse bzw. *Domain* wird eine Eigenschaft mit einem Wertebereich bzw. *Range* bestimmt. Es gibt zwei Typen von Eigenschaften:
Object Properties verknüpfen OWL Individuals miteinander, *Datatype Properties* verknüpfen Individuals mit Literalwerten (primitive/atomare XML Schema-Typen). Es können weitere Sub-Properties definiert werden, die ein Property spezialisieren.
Der OWL Sprachumfang umfasst für Eigenschaften folgende Axiome[14]:

- `ObjectProperty`, `DatatypeProperty`
- `domain`, `range`, `subPropertyOf` (RDF Schema)

[13] http://www.w3.org/2004/OWL/
[14] http://www.w3.org/TR/owl-ref/

- `equivalentProperty`, `inverseOf` (OWL)
- `FunctionalProperty`, `InverseFunctionalProperty` (globale Kardinalitäten, OWL)
- `SymmetricProperty`, `TransitiveProperty` (logische Charakteristika, OWL)

Restriktionen OWL unterscheidet drei Kategorien:

- Restriktionen bzgl. des Wertebereichs *(quantifier restrictions)*: Der Quantifier `someValuesFrom` verlangt mindestens eine oder mehrere Instanzen. Der Universal Quantifier `allValuesFrom` bedeutet, dass nur eine Instanz bzw. nur einen Literaltyp benutzt wird.
- Die Restriktion `hasValue` bezieht sich auf ein Individual oder einen Datenwert.
- Restriktionen bzgl. der Kardinalität: `cardinality`, `minCardinality`, `maxCardinality`

Individuen OWL Individuen repräsentieren Objekte (Instanzen) von Klassen. Auf der Grundlage ihrer Eigenschaften kann entschieden werden, ob eine Instanz zu einer Klasse gehört oder nicht. Dazu lassen sich Eigenschaften als *notwendig* oder *hinreichend* definieren. Eine notwendige Eigenschaft muss von einer Instanz erfüllt sein, damit sie zu einer Klasse gehört. Wurde für eine Klasse jedoch eine hinreichende Eigenschaft definiert, genügt es für eine Instanz, dieser Eigenschaft zu entsprechen, um einer Klasse eindeutig zugeordnet werden zu können.

Modellierung von OWL Klassen

Eine Klasse bzw. ein Begriff wird über seine Eigenschaften definiert. Eine Eigenschaft wird dabei über die Relation einer Klasse zu einer anderen Klassen beschrieben. Für Relationen zwischen Klassen können notwendige *(engl. necessary)* und hinreichend notwendige *(engl. necessary and sufficient)* Bedingungen festgelegt werden [KFNM04]. Notwendige Charakteristika einer Klasse werden wie Eigenschaften einer Superklasse behandelt. Für Instanzen der Klasse bedeutet dies, dass notwendige Bedingungen erfüllt sein müssen, um zu einer Klasse zu gehören. Als Beispiel kann die Pizza Ontologie[15] betrachtet werden. Die verschiedenen Pizzatypen (American, AmericanHot,...) definieren sich durch die Wahl ihrer Beläge. Eigenschaften, die eine Klasse hinreichend charakterisieren beschreiben eine Menge von Unterklassen. Hinreichend notwendige Eigenschaften beschreiben für Klassen eine Äquivalenz *(equivalent classes)*. Durch eine Äquivalenz wird die Klasse CheeseyPizza der Pizza Ontologie definiert. Die Klasse ist dabei äquivalent zu der Schnittmenge aus Pizza und der Restriktion (hasTopping, CheeseTopping).

Klassen, für die nur notwendige Eigenschaften definiert sind, werden als primitive oder unvollständige Klassen bezeichnet. Mit hinreichenden Bedingungen kann ausgedrückt werden, dass ein Objekt zu einer Klasse gehört, wenn es diese Anforderungen erfüllt. Eine Klasse, die mindestens

[15] http://www.co-ode.org/ontologies/pizza/

eine Kombination aus notwendiger und hinreichender Bedingung hat, wird als definierte oder vollständige Klasse bezeichnet [Sch05].

OWL Untersprachen

Die OWL Spezifikation unterscheidet drei Sprachebenen: *OWL-Lite, OWL-DL und OWL-Full*.

OWL-Lite OWL-Lite beinhaltet alle Grundstrukturen, um Klassen und Relationen zu definieren. Zum Standardsprachumfang gehören `subClassOf` (RDFS), `objectProperty`, `datatypeProperty`, `subproperty`, `domain` und `range`. Die Kardinalität kann mit Hilfe von `cardinality`, `minCardinality`, `maxCardinality` festgelegt werden. Allerdings erlaubt OWL-Lite nur die Werte 0 und 1. Mit `allValuesFrom` wird eine bestimmte Klasse oder ein Literal als Wertebereich festgelegt. `someValuesFrom` schränkt den Wertebereich eines Properties über die Angabe von Individuen ein.

OWL-DL OWL-DL erweitert OWL-Lite um folgende Konstrukte: `hasValue` schränkt den Wertebereich auf ein Individuum oder einen Literalwert ein. Kardinalitäten mit Werten größer 1 sind erlaubt. Es gelten die boolschen Kombinationen `complementOf`, `unionOf` und `intersectionOf`. Zum Sprachumfang gehören auch die Aufzählung mit `oneOf` und die Beschreibung von disjunkten Konzepte mit `disjointWith`.
OWL-DL ist entscheidbar.

OWL-Full OWL-Full nutzt den vollen Sprachumfang von OWL. Allerdings verliert man dadurch Garantien über die Vollständigkeit der Beschreibung, die von einem *Reasoner* erwartet wird (Inferenzen über die beschriebenen Ontologien). OWL-Full ist nicht entscheidbar.

Es gilt: $\textit{OWL-Lite} \subset \textit{OWL-DL} \subset \textit{PL1} \subset \textit{OWL-Full} \subset \textit{Prädikatenlogik}$

Logikhintergrund von OWL-DL

DL steht für *description logic* (Beschreibungslogik) und legt die Ausdrucksstärke der Ontologie fest. OWL-DL entspricht der Beschreibungslogik $\mathcal{SHOIN}(\mathcal{D})$ (syntaktische Variante).

Charakteristika einer Beschreibungslogik (OWL-DL) nach [DOS03]:

- Teilmenge der Prädikatenlogik erster Stufe (PL1), meist entscheidbar
- Ausdrucksstärke orientiert an praktischer Verwendbarkeit
- Definition eines Vokabulars, bestehend aus *Konzepten* einer Domaine
- Schlussfolgerungen möglich (man gewinnt aus vorhandenem Wissen neues Wissen)

Beschreibungslogiken „Beschreibungslogiken ist ein Sammelbegriff für verschiedene logikbasierte Sprachen, die mit dem Schwerpunkten Wissensrepräsentation und Konzeptualisierung eines Gegenstandsbereichs entwickelt wurden.“
(Prof. Dr. Benno Stein, Bauhaus Universität Weimar)

Anwendung der Beschreibungslogik OWL-DL

Für OWL-DL gelten nach [AH04] bzw. W3C[16] folgende Bedingungen:

1. Eine Ressource ist entweder eine Klasse oder ein Individuum oder ein Datenwert.
2. Explizite Typangaben erforderlich.
3. OWL-DL verbietet die Verwendung von:
   - `owl:inverseOf`
   - `owl:FunctionalProperty`
   - `owl:InverseFunctionalProperty`
   - `owl:SymmetricProperty`
4. Kardinalitätsangaben sind in transitiven Properties nicht erlaubt.
5. Eingeschränkte Verwendung anonymer Klassen.

OWL-DL gibt vor, welche Information aus einer Ontologie extrahiert werden kann. Dieses Wissen spielt bei der Entwicklung eines Parsers, der Informationen durch Lesen von Ontologien und Schlussfolgern *(engl. reasoning)* sammelt, eine entscheidende Rolle. Will man ein Programm entwickeln, das Informationen einer Ontologie verarbeitet, muss demnach angegeben werden, mit welcher Sprachspezifikation man arbeiten will.

Jena[17] erwartet Angaben der Form *createOntologyModel(OntModelSpec spec)*, um ein internes Modell einer Ontologie aufbauen zu können. Für OWL-DL gibt es folgende Möglichkeiten, um das interne Datenmodell zu spezifizieren[18] (Auszug aus der API):

OWL_DL_MEM A specification for OWL DL models that are stored in memory and do no additional entailment reasoning

OWL_DL_MEM_RDFS_INF A specification for OWL DL models that are stored in memory and use the RDFS inferencer for additional entailments

OWL_DL_MEM_RULE_INF A specification for OWL DL models that are stored in memory and use the OWL rules inference engine for additional entailments

OWL_DL_MEM_TRANS_INF A specification for OWL DL models that are stored in memory and use the transitive inferencer for additional entailments

Zusätzlich zu den Möglichkeiten, die Jena bietet, können speziell für OWL-DL entwickelte Reasoner eingesetzt werden, die auch zusammen mit Jena arbeiten. Für die Implementierung des Parsers wurde der Open-Source OWL-DL Reasoner Pellet[19] benutzt.

[16] `http://www.w3.org/TR/owl-ref/#OWLDL`
[17] Bibliothek für Java: `http://jena.sourceforge.net`
[18] Jena API OntModelSpec
[19] `http://pellet.owldl.com`

2.2.4 Web Ontology Language for Services (OWL-S)

Semantic Web Services erweitern konventionell definierte Web Services um eine semantische Beschreibung, indem semantische Informationen (Wissen) des *Semantic Web* in die, durch Web Services beschriebene, Business Logik integriert wird. Die semantische Zusatzinformation hat gegenüber rein syntaktisch beschriebenen Services einige Vorteile in Bezug auf Grundoperationen der Web Services Technologie (siehe Tabelle 2.1). Die Suche und das Auffinden *(engl. discovery)*, die Selektion, die Komposition *(engl. composition)* sowie die Integration und Ausführung *(engl. invocation)* von Web-Services. [GT04, MPM+04, AS05]

Task	Description
Discovery	Locating Web Services (conventionally using UDDI)
Invocation	Excecution of service by an agent or other service
Interoperation	Breaking down interoperability barriers and automatic insertion of message parameter translations..
Composition	New services through automatic selection, composition and interoperation of existing services..
Verification	Verify service properties..
Excecution Monitoring	Tracking the execution of composite tasks and identifying failure cases of different excecution traces..

Tabelle 2.1: Maschinen-interpretierbare Tasks [AS05]

Die *Web Ontology Language for Services*[20] (OWL-S) ist eine Sprache um *Web Services* semantisch zu beschreiben. Ihr Vorgänger ist die DAML-S Spezifikation, die im Rahmen des DAML-Programms[21] der DARPA erstellt worden ist. Das *OWL-S Modell* wird mittels OWL semantisch beschrieben. Referenziert werden die nötigen Ontologien über den Namespace und über Imports. Was die Kontrolle des Nachrichtenflusses zwischen Prozessen betrifft, orientiert sich OWL-S an der *Semantic Web Rule Language* (SWRL) [AMM+04].

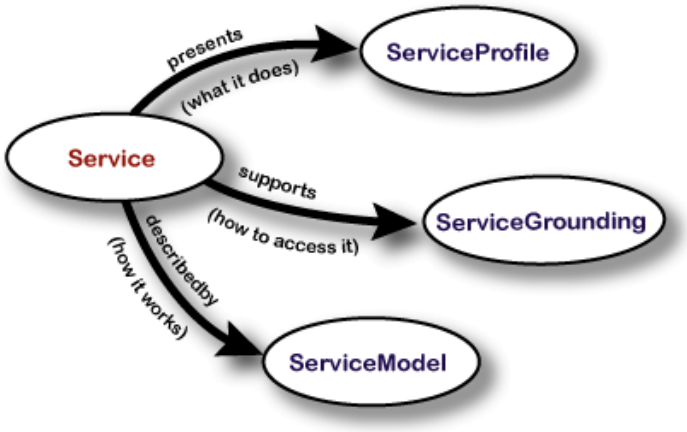


Abbildung 2.4: Toplevel der Service Ontologie *(upper ontology)*

[20] http://www.w3.org/Submission/OWL-S/
[21] http://www.daml.org

Die OWL-S Service Definition besteht aus drei Teilen (siehe Abb. 2.4): Das *ServiceProfile* beschreibt, was der Dienst macht *(presents)*. Hier wird durch Definition der Ein- und Ausgabe des Prozesses dessen Signatur spezifiziert. Das *ServiceModel* beschreibt wie der Dienst arbeitet *(describedBy)*. Im *ServiceGrounding* wird durch Angabe von Kommunikationsprotokoll und Serialisierungstechnik beschrieben, wie auf einen Web Service zugegriffen werden kann *(supports)*.

```
<owl:Ontology rdf:about="#">
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Service.owl"/>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Profile.owl"/>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl"/>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Grounding.owl"/>
</owl:Ontology>
```

Listing 2.1: OWL-S Service Beschreibung *(conceptual areas)*

Das Service Profile

Das *Service Profile* dient der Entdeckung des Dienstes. Innerhalb der semantischen Beschreibung beinhaltet das *Service Profile* funktionale und nicht-funktionale Eigenschaften. Zu den nicht-funktionalen Parametern gehören Informationen für den menschlichen Benutzer wie serviceName, textDescription und contactInformation. Weiterhin dienen Parameter wie ServiceCatergory zur Einordnung des Web Services. Zu den funktionalen Eigenschaften des *ServiceProfile* gehört

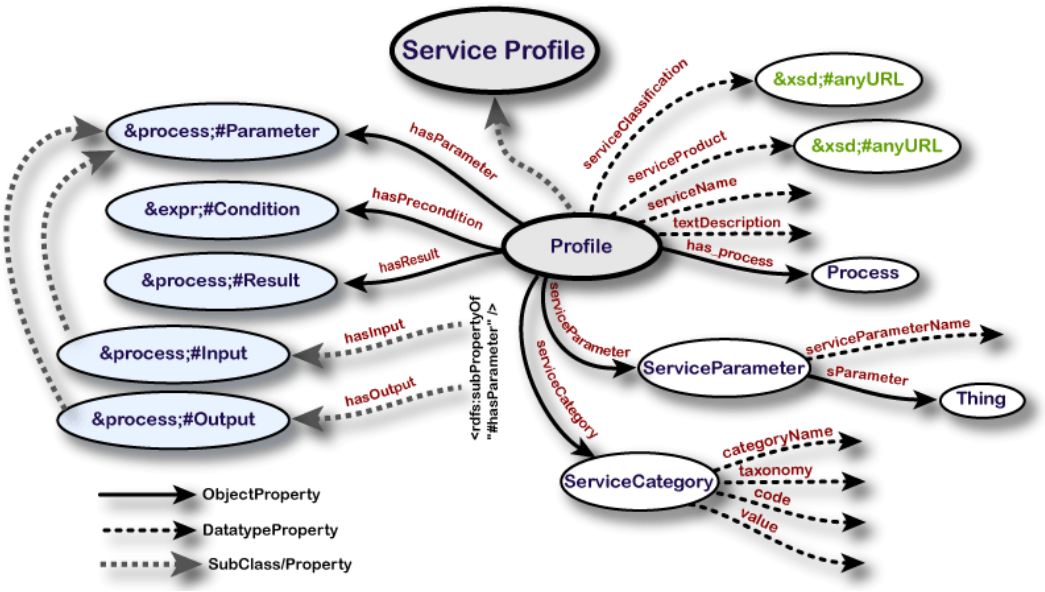


Abbildung 2.5: Profile Ontologie

die Definition der Parameter (hasInput, hasOutput), der Vorbedingungen *(engl. preconditions)* und der Ergebnisse *(engl. results)*. Dies wird von dem Prozessmodell geerbt. Die Typdeklaration der Parameter geschieht im Prozessmodell über Referenzierung von OWL Klassen. Durch Importieren der entsprechenden Ontologien steht der Servicebeschreibung so eine Wissensbasis zur Verfügung. Das *ServiceProfile* spezifiziert nicht, wie der Service ausgeführt werden kann.

Die Prozessbeschreibung

Das *ProcessModel* beschreibt ähnlich eines Protokolls, wie der Service funktioniert und wie er ausgeführt werden kann *(engl. service orchestration and invocation)*. Mit Hilfe einer entsprechenden Ontologie werden Service Signatur sowie Vor- und Nachbedingungen beschrieben. Eine zusätzliche Ontologie (control ontology) ermöglicht die Beschreibung des Status eines Prozesses (initial activation, execution, completion). Das *ProcessModel* unterstützt die Komposition von Web Services und das Monitoring der Interaktion.
OWL-S definiert drei unterschiedliche Prozessarten: ***atomic, simple und composite***.

```
<owl:Class rdf:ID="Process">
  <rdfs:comment> The most general class of processes </rdfs:comment>
  <rdfs:subClassOf rdf:resource="&service;#ServiceModel"/>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AtomicProcess"/>
    <owl:Class rdf:about="#SimpleProcess"/>
    <owl:Class rdf:about="#CompositeProcess"/>
  </owl:unionOf>
</owl:Class>
```

Listing 2.2: OWL-S 1.1, Ausschnitt der Process.owl

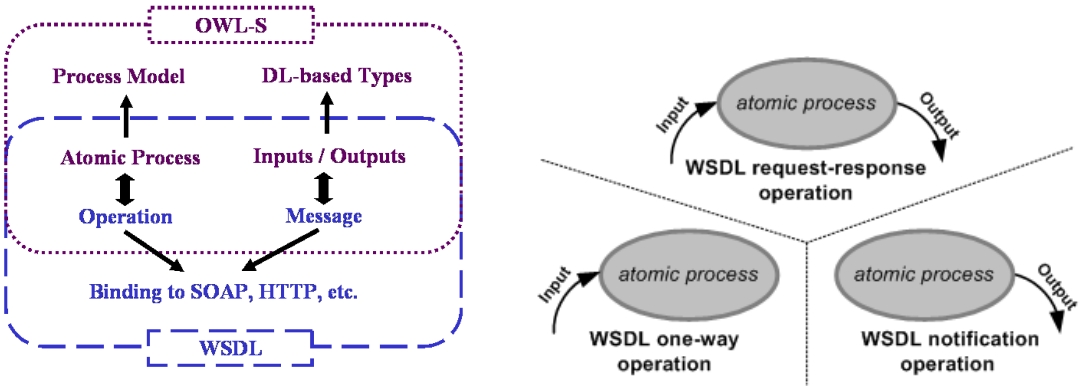
AtomicProcess Ein atomarer Prozess *(AtomicProcess)* innerhalb der Prozessbeschreibung ist ein direkt aufrufbarer bzw. ausführbarer Prozess *(engl. directly invokable)*, der nur eine spezielle Aufgabe erledigt. Es existieren keine weiteren Sub-Prozesse. Die Ein- und Ausgabeparameter eines atomaren Prozesses werden durch OWL Klassen typisiert.

SimpleProcess Bei einem *SimpleProcess* handelt es sich auch um einen Ausführungsschritt, allerdings beschreibt die Definition eine abstrakte Sicht. Der Prozess ist nicht direkt aufrufbar.

CompositeProcess Eine Prozess-Komposition *(CompositeProcess)* setzt sich aus mehreren Sub-Prozessen zusammen, die selbst wieder atomar, simple oder kombiniert sein können. Zusätzlich werden Kontrollkonstrukte wie *sequence, split, join, any order oder choice* und Kontrollstrukturen wie *Sequence* und *If-Then-Else* genutzt, um eine Ablaufsteuerung *(engl. control flow)* zu implementieren. Neben der Ablaufssteuerung beschreibt eine solche Spezifikation den Datenfluss *(data flow)*, d.h. welche Prozess Ein- und Ausgaben an andere Sub-Prozesse *(tasks)* geleitet werden.

Das Grounding

Das *Grounding* bestimmt, wie ein Service aufgerufen wird. Will man einen WSDL basierten Web-Service über einen atomaren Prozess nutzen, wird die passende WSDL Operation, die WSDL Messages sowie der WSDL PortType und das WSDL Binding referenziert. Parameter eines atomaren Prozesses entsprechen dabei den *WSDL Message Parts*. Die Typen der *WSDL Message Parts* werden innerhalb einer WSDL Definition direkt in der <type> Section über XML Schema beschrieben (bzw. importiert).



(a) WSDL Grounding (W3C Submission) (b) WSDL Mapping der Prozesstypen [AS05]

Abbildung 2.6: WSDL Grounding

Vergleich von OWL-S und WSDL

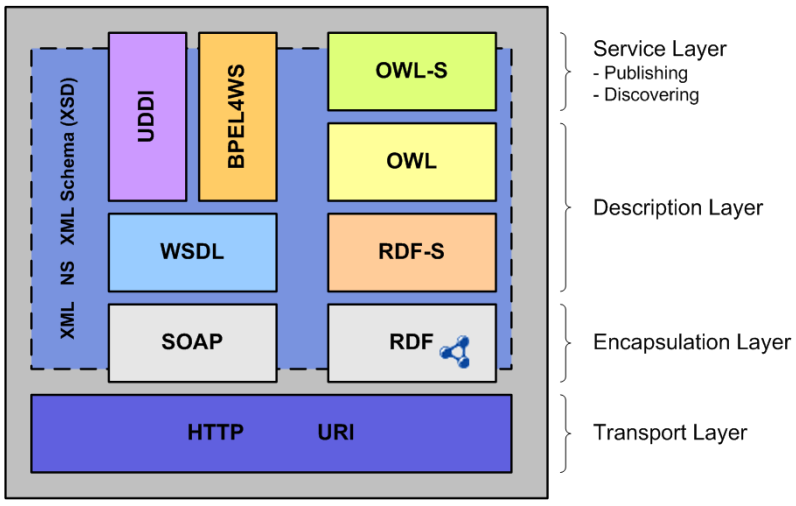


Abbildung 2.7: Vergleich der Protokollstacks von WSDL und OWL-S

Abbildung 2.7 zeigt, dass OWL-S Beschreibung der Menge aus WSDL und BPEL4WS entspricht. Die Möglichkeit des Matchmakings von OWL-S entspricht dem Zugriff auf ein UDDI Verzeichniss. Die Schichten der einzelnen Ebenen entsprechen sich nicht direkt, sondern sind durch ihre Funktionalität vergleichbar.

Die OWL-S Spezifikation entspricht folgenden Web Service Technologien [SPH04]:

- Das *ServiceProfile* entspricht den Einträgen einer UDDI Registry (UDDI API).
- Das *ProcessModel* mit der Möglichkeit der Komposition und der Ablaufssteuerung entspricht dem Geschäftsprozessmodell der *Business Process Execution Language* (BPEL4WS).
- Das *Grounding* mappt OWL-S Prozesse z.B. nach WSDL oder Java.

Die Definition eines atomaren Prozesses entspricht einer WSDL Operation. Es gelten die Beziehungen in Abbildung 2.6 (b). Die jeweiligen Ein- und Ausgaben eines Prozesses entsprechen WSDL Messages und werden über das *Grounding* auch auf passende WSDL Messages gemappt. WSDL kennt weiterhin die *solicit-response* Operation, bei der ein Dienst zuerst eine Nachricht an den Client sendet und danach von dem Client eine Bestätigung erwartet.

2.2.5 Interoperabilität von Semantischen Web Services

Interoperabilität fordert, benötigte Dienste unkompliziert, einfach und möglichst zur Laufzeit des Systems auffinden zu können. Semantische Web Services ermöglichen [SPAS03]:

- die semantische Beschreibung eines Web Service Tasks innerhalb einer Geschäftslogik (z.B. Verkauf eines Buches, Verifikation einer Kreditkarte).
- das automatisierte Auffinden *(engl. discovery)* anhand der explizit definierten Ausschreibung *(engl. advertisement)* und der Beschreibung des Dienstes.
- Beziehungen zu anderen Diensten und Regeln innerhalb der Geschäftsprozesslogik.
- die semantische Beschreibung der auszutauschenden Daten.
- die Definition eines Prozessflusses durch die Einführung von Vor- und Nachbedingungen. *(engl. preconditions and effects/results)*.
- die Komposition von Web Services zu komplexeren Web Services.

Das Auffinden von Servicedefinitionen wird mit Hilfe von Matchmakern durchgeführt. Ein wichtiges Anwendungsgebiet ist der Bereich Agentensysteme.

Aktuelle OWL-S Matchmaking Technologien:

- CMU[22] OWL-S/UDDI Matchmaker[23]
- Hybrid Semantic Web Service Matchmaker OWLS-MX[24]
- TUB OWL-S Matcher OWLSM[25]

[22] Intelligent Software Agents Lab, Carnegie Mellon University
[23] http://www.daml.ri.cmu.edu/matchmaker/
[24] DMAS, DFKI Saarbrücken, http://projects.semwebcentral.org/projects/owls-mx/
[25] http://owlsm.projects.semwebcentral.org

2.2.6 Matchmaking von Semantischen Web Services

Einsatz von Matchmakern

Matchmaker ermöglichen das Auffinden von Web Services. Die Idee bei der Verwendung von semantischen Matchmakern ist, aufgrund der semantischen Information zusätzliche Beziehungen zwischen Web Services zu erkennen, die nicht in einer genutzten UDDI Registry auftauchen. Matchmaking Algorithmen im *Semantic Web* Bereich können mit Hilfe von *Reasonern* passende Dienste auffinden, indem sie durch Schlussfolgern (Subsumieren) semantische Ähnlichkeit feststellen können.[SPH04]

Arbeitsweise

Matchmaker werden zur Bestimmung von Ähnlichkeitswerten eingesetzt. Matchmakingwerte sind in der Regel Ähnlichkeitswerte, die in Prozent ausgedrückt werden. Bei semantischen Web Services wird darüber hinaus eine semantische Ähnlichkeit bestimmt.

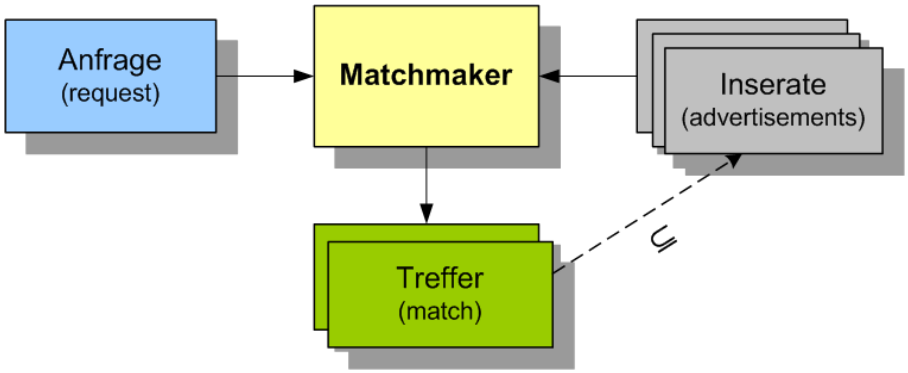


Abbildung 2.8: Arbeitsweise eines Matchmakers [KK06]

Verglichen werden die Fähigkeiten *(engl. capabilities)* eines Dienstes:

- Eingaben *(inputs)*
- Vorbedingungen *(preconditions)*
- Ausgaben *(outputs)*
- Nachbedingungen *(results)*

Die Signatur besteht lediglich aus den Ein- und Ausgaben.

Architektur eines Matchmakers

Die Architektur eines semantischen Matchmakers (z.B. OWLS-MX, Abschnitt 8.1.2) besteht aus einer Datenbank für Ontologien, der Datenbank für die Inserate, einem *Reasoner* und einer oder mehreren *Matching Engines* (Syntaktische und Semantische Matching Engine) [KK06, BF06].

2.2.7 OWL-S innerhalb einer modellgetriebenen Architekur (MDA)

Hauptvorteile einer *modellgetriebenen Architekur*, *(engl. model driven architecture)* (MDA) sind die Portabilität, die Interoperabilität und Wiederverwendungsmöglichkeit von Software durch eine architekturbedingte Trennung von Spezifikation und Implementierung. Die Geschäftslogik wird von der Implementierungsplattform separiert. Bei einer MDA konzentriert sich der Entwickler auf die Erstellung von Modellen anstatt auf die Implementierung von Quellcode.

Die *Semantic Web Services Technologie* ähnelt dieser Arbeitsweise. Sie ermöglicht die unabhängige Entwicklung des semantischen Dienstes (OWL-S). Über einen *top down* Ansatz lässt sich der Web Service über das WSDL Grounding konkretisieren. Das Grounding verknüpft OWL Konzepte mit WSDL Beschreibungen. Durch die Trennung der Geschäftsprozesslogik ist beispielsweise die Translation OWL-S2BPEL möglich. Allerdings handelt es sich bei OWL-S nicht um ein abstraktes plattformunabhängiges Metamodell wie beispielsweise PIM4SOA. OWL-S Definitionen erweitern in der Regel existierende Service Beschreibungen *(bottom up)* um das ServiceProfile und ServiceModel (Abschnitt 2.2.4). Eine OWL-S Service Definition muss WSDL Grounding besitzen, um eine Service Implementierung innerhalb einer SOA nutzen zu können.
Bemerkung: In der Regel liegt für jeden semantischen Web Service auch ein entsprechender konventioneller Web Service vor, der die Implementierung der Funktionalität beschreibt. In der OWLS-MX Test Collection fehlt der Bezug zu den Service Beschreibungen (WSDL Grounding) und damit auch zu der Service Implementierung. Im Rahmen dieser Arbeit werden die fehlenden WSDL Beschreibungen anhand der OWL-S Service Signatur generiert und über ein Re-Engineering der fehlende Groundingteil rekonstruiert (siehe Kapitel 3).

Aus der MDA Architektur folgt die Aufteilung der Tätigkeitsfelder *Implementierung der Anwendung*, *Implementierung der Web Service Schnittstelle*, *semantische Beschreibung des Dienstes* und die *semantische Einordnung* der, für dessen Schnittstelle benötigten, Begriffe. Innerhalb des Entwicklungsprozesses werden demnach die Rollen *Software-Entwickler*, *Web Service Entwickler*, *SOA Spezialist* und *Domain Expert* (Computerlinguist) benötigt (siehe Abb. 5.2). Sobald semantische Dienste definiert worden sind, lassen sie sich innerhalb einer SOA auf semantischer Modellebene ansprechen, ausführen und kombinieren. *Composition Planning Tools* unterstützen die Orchestrierung von Semantic Web Services. Ein erster Vorschlag einer Service Komposition inklusive Toolchain (WSDL2OWL-S Converter, Workflow Editor) wurde 2004 von den Entwicklern der OWL-S API[26] vorgestellt [SPH04]. Ebenfalls mit OWL-S arbeitet der am DFKI entwickelte *OWL-S service composition planner OWLS-XPlan*[27] [KG06].
Ergänzung: MDA wird durch die Object Management Group[28] (OMG) standardisiert und basiert auf bzw. steht in Zusammenhang mit folgenden weiteren Standards der OMG: *Unified Modeling Language* (UML), *Meta-Object Facility* (MOF), *XML Meta-Data Interchange* (XMI) und *Common Warehouse Metamodel* (CWM) [GT04, TG05].

[26] http://code.google.com/p/owl-s/ ehem. http://www.mindswap.org/2004/owl-s/api/
[27] http://projects.semwebcentral.org/projects/owls-xplan/
[28] http://www.omg.org

Kapitel 3

Analyse der Problemstellung

Aufgrund der zahlreichen Spezifikationen, Tools und APIs in den Bereichen SOA, Web Services und Semantic Web Services ist es wichtig, das Technologie-Umfeld und verwandte Arbeiten vorzustellen, um das Thema richtig einordnen zu können. Ziel dieses Kapitels ist die Analyse der Problemstellung und die Erstellung eines Pflichtenhefts.

3.1 Einordnung des Themas

Die Bezeichnung des zu erstellenden Tools **OWLS2WSDL** deutet es bereits an: Ziele der Arbeit sind die Konzeptionierung und die Implementierung einer Translation von OWL-S nach WSDL. Die Abbildung 3.1 zeigt die Anwendung der beiden Definitionsarten innerhalb einer SOA.

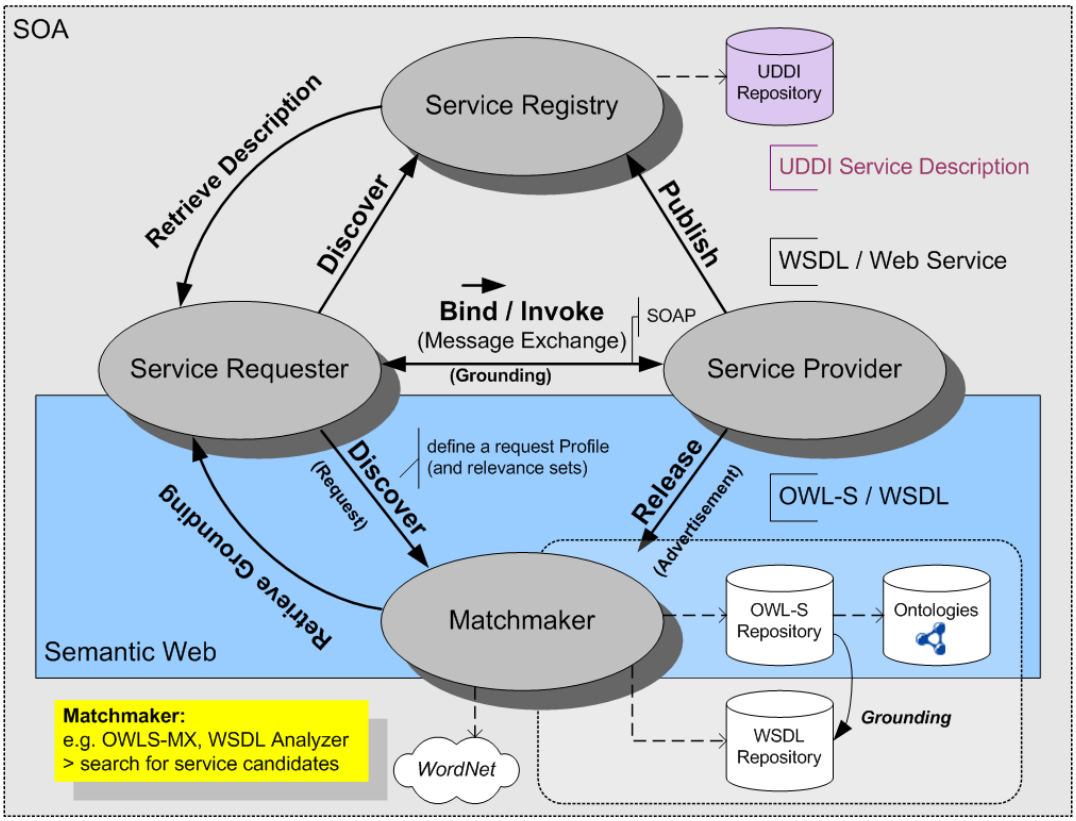
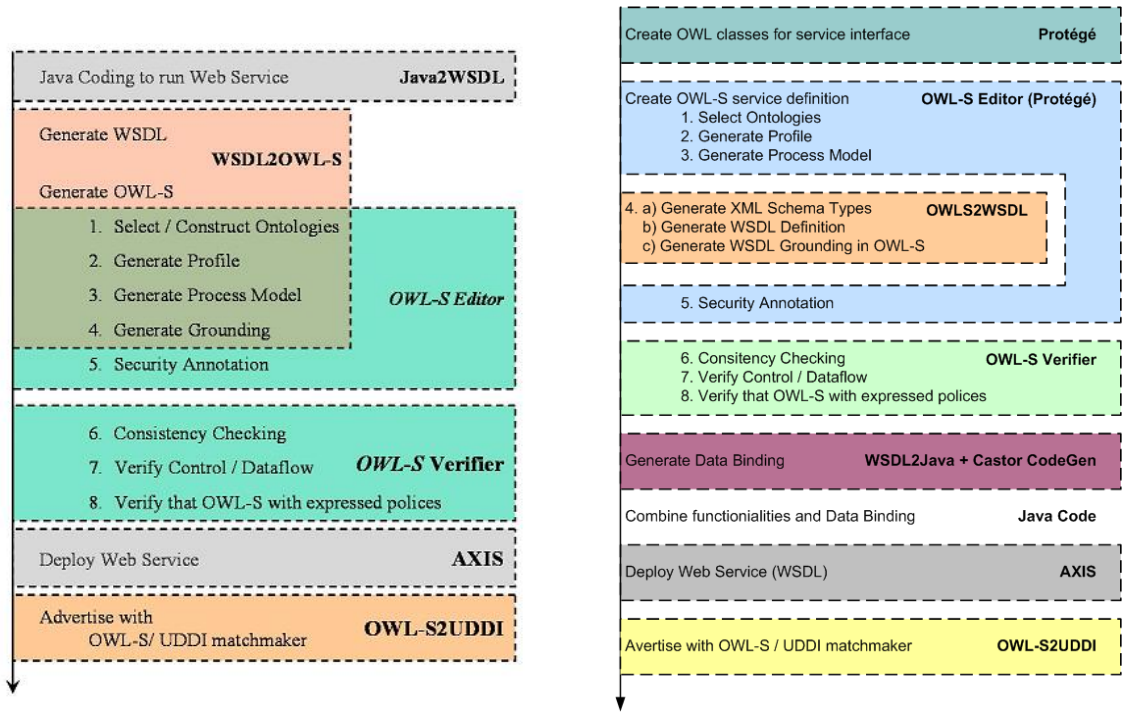


Abbildung 3.1: SOA: Web Services und Semantic Web Services

Für die generierten WSDL Definitionen soll das Matching Verhalten überprüft werden und mit dem Matching Verhalten der ursprünglichen OWL-S Definitionen verglichen werden. Wie die Abbildung 3.1 zeigt, dienen Matchmaker neben UDDI dem Auffinden von Web Services. Ein Matchmaker findet für eine Anfrage *(request)* einen Treffer bzw. eine Treffermenge *(match)* aus einer Menge von Inseraten *(advertisements)*. Die Anfrage besteht dabei selbst aus einer Referenzbeschreibung. Semantische Matchmaker betrachten die semantische Ähnlichkeit der Parameter von Service Schnittstellen innerhalb der Inseratmenge (siehe [KK06]).



(a) WSDL2OWL-S (b) OWL-S, top down Ansatz

Abbildung 3.2: Realisierung eines OWL-S Dienstes

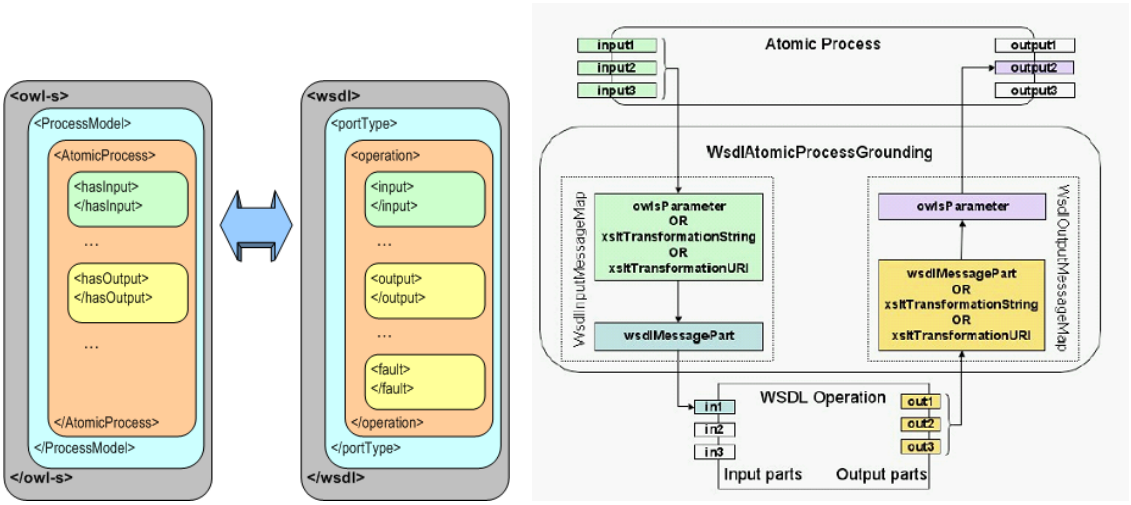
Anhand der Abbildung 3.2 (b) wird die Realisierung eines semantischen Web Services beginnend mit der Erstellung einer OWL-S Definition beschrieben. Die Translation OWL-S nach WSDL wird in die Entwicklung eingeordnet. Die Grafik wurde der Übersicht 3.2 (a) nachempfunden, die den umgekehrten Weg beschreibt: die Realisierung eines in OWL-S beschriebenen Dienstes, der auf einer vorhandenen Implementierung in Java basiert. Dazu genutzt wurde die Konvertierung von WSDL2OWL-S und Tools des *Intelligent Software Agents Lab*[1] (iSAL).

[1] http://www.cs.cmu.edu/~softagents/, Carnegie Mellon Universität

3.2 Mapping von OWL nach W3C Schema

Wie in der Abbildung 3.2 gezeigt wird, gehört zu der Translation von OWL-S nach WSDL die Generierung von XML Schema. Eine OWL Definition (Ontologie) beinhaltet dabei auch OWL Individuen. Das XML Schema beinhaltet Typinformationen, um die Parameter der Service-Schnittstelle beschreiben zu können. Bei der WSDL Schnittstelle kann es sich um primitive, simple und komplexe Datentypen handeln. Parameter-Typen eines atomaren OWL Prozesses sind entweder primitive Typen (XML Schema) oder OWL Klassen. Das WSDL Grounding als Bestandteil einer OWL-S Definition mappt die im OWL-S Prozessmodell verwendeten OWL Konzepte mit den Schema Typen der WSDL Beschreibung (wsdl:part) [PSSN03].

Im Hinblick auf die Translation OWL-S nach WSDL bedeutet dies, dass das generierte Schema möglichst genau dem ursprünglichen OWL Klassenmodell entsprechen sollte, um bei Anfrage eines WSDL Services und Empfang einer SOAP Response Nachricht ein vollständig definiertes OWL Individual (Instanz einer OWL Klasse) bilden zu können.



(a) Mapping von OWL-S und WSDL

(b) WSDLAtomicProcessGrounding

Abbildung 3.3: WSDL Grounding einer atomaren Prozessbeschreibung. [Syc06]

Handelt es sich bei einem Parametertyp um eine referenzierte OWL Klasse, muss zusätzlich zu dem Mapping des OWL Parametertyps auf den Typ des WSDL Message-Part Elements, eine entsprechende Transformationsanweisung (`grounding:xsltTransformationString`) angegeben werden, um mit der Service-Schnittstelle kommunizieren zu können.

Beispiel: Für eine komplexe WSDL Response Nachricht (SOAP, XML-Instanz) müssen Daten extrahiert werden, um ein OWL Individual zu bilden (RDF Graph), das den Output eines atomaren Prozesses darstellt.

Data Binding und XML Schema

Das Mappen von XML Schema Instanzen auf ein Objektmodell wird als *Data Binding* bezeichnet. Ein *Data Binding Framework* für Java ermöglicht es, anhand einer XML Schema Definition entsprechende Java-Klassen zu generieren und die Objekte mit Werten aus der XML Instanz zu belegen. Für die Realisation der Translation OWL2XSD ist die Abbildungsfolge OWL-Modell auf Java-Klassenmodell und Java-Klassenmodell nach XML Schema von Interesse.

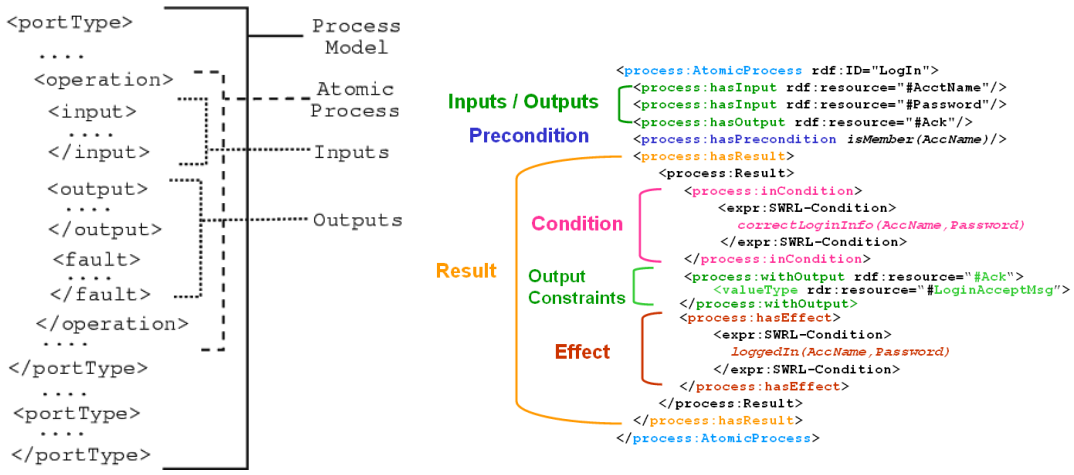
3.3 Mapping der OWL-S Prozess Beschreibung nach WSDL

Anwendungsfälle der Translation OWL-S nach WSDL im Rahmen dieser Arbeit sind:

1. Mit Hilfe der generierten WSDL Beschreibungen soll ein Vergleich der beiden Matchmaker *OWLS-MX* und *WSDL Analyzer* ermöglicht werden.

2. Die Generierung von WSDL Groundings für OWL-S Definitionen der OWLS-TC.

Für die Generierung von WSDL aus OWL-S muss die Signatur eines atomaren OWL-S Prozesses betrachtet werden. Das Mapping von OWL-S Prozessen und WSDL Operationen wird in den Abbildungen 2.6, 3.3 und 3.4 vorgestellt. Die Übersetzung des Workflows *(service capabilities: precondition, result)* (Beispiel Abb. 3.4 b) und Vorstellung OWLS2BPEL in Abschnitt 3.4.3) ist nicht Teil dieser Arbeit.



(a) Mapping der Prozess Signatur

(b) Beispiel: Mapping aller Fähigkeiten (OWL-S 1.2)

Abbildung 3.4: Mapping der OWL-S Prozess-Signatur [Pol05].

Erstellung von OWL Instanzen anhand einer SOAP Message

Ein OWL-S Anwendungsbeispiel ist der ZipCodeFinder Service (siehe auch Anhang B.1). Anhand des Beispiels kann die Arbeitsweise des OWL-S WSDL Groundings erläutert werden. Das Beispiel beschreibt den klassischen Anwendungsfall von OWL-S: Die Rückgabewerte eines bereits existierenden Web Services (WSDL) werden genutzt, um die Rückgabeparameter (OWL

Individuen) eines atomaren Prozesses zu instanzieren. Anhand der Einschränkungen, die man bei dieser *bottom up* Arbeitsweise bzgl. der Vollständigkeit der OWL Klasseninstanzen erkennt, lassen sich Anforderungen an die Translation von OWL-S nach WSDL definieren.

Insgesamt sind 4 Definitionen erforderlich:

1. **OWL:** http://www.daml.org/2001/10/html/zipcode-ont#ZipCode
2. **OWL-S:** http://www.mindswap.org/2004/owl-s/1.1/ZipCodeFinder.owl
3. **WSDL:** http://www.tilisoft.com/ws/LocInfo/ZipCode.asmx?WSDL
4. **XSD:** http://www.tilisoft.com/ws/LocInfo/DataObjects/ZipCodeData.xsd

Matchingverhalten

Je mehr syntaktische und semantische Information des OWL Konzeptmodells nach XML Schema (WSDL) übersetzt werden, desto ähnlicher sollte das Matchmakingverhalten des *WSDL Analyzers* dem des *OWLS-MX* sein. Aufgrund der Mächtigkeit der verwendeten Beschreibungslogik (OWL-DL) ist jedoch zu erwarten, dass ein semantisches bzw. hybrid-semantisches Matchmaking, das die Verwendung eines *Reasoners* impliziert immer präziser sein wird als ein rein syntaktisches Matchmaking.

3.4 Verwandte Arbeiten

3.4.1 OWL-S Groundings

Service Grounding in WSDL

Bereits im Oktober 2003 hat das DAML-S Projekt in [MBL+03] für OWL-S 1.0 das WSDL Grounding vorgestellt, das bis heute die meist genutzte Funktionalität darstellt.
Die de fakto Implementierung von OWL-S (Version 1.0 und 1.1) ist die OWL-S API für Java von Evren Sirin [EP04]. Sie unterstützt das Grounding für WSDL und UPnP. Die integrierte *Execution Engine* kann einen atomar definierten Prozess, den AtomicProcess, sowie komponierte Prozesse, eine CompositeProcess Definitionen, ausführen. Die API wurde anfangs von dem mindswap[2] Projekt[3] der Universität Maryland gehostet. Die aktuelle Version findet man seit August 2006 bei Google Code[4].

Service Grounding in Java

Im Rahmen seiner Diplomarbeit stellt Michael Dänzer[5] eine Erweiterung der OWL-S API um ein Java Grounding vor [Dän05].

[2] Maryland Information and Network Dynamics Lab Semantic Web Agents Project
[3] http://www.mindswap.org
[4] http://code.google.com/p/owl-s/
[5] http://www.ifi.unizh.ch/ddis/people/michaeldaenzer/

Bridging the Gap Between Abstract and Concrete Services

Eine wichtige Aufgabe während der Ausführung eines *Semantic Web Service* (OWL-S) ist das **bidirektionale Mapping** der semantisch definierten Serviceparameter auf die im WSDL-Grounding festgelegten XML Schema Typen der genutzten Service-Beschreibung (WSDL). Das im Grounding benutzte Konzept der XSL Transformation wird durch den Vorschlag von [BL04, PNL05] durch ein alternatives Konzept des Mappings auf einer semantischen Basis ersetzt. Grund für diesen Ansatz war der Fakt, dass ein OWL-Modell immer mehrere Serialisierungsmöglichkeiten in RDF hat, und für jede Abbildung ein eigenes XSL Script geschrieben werden muss.

Der Artikel [BL04] zeigt, dass die Transformation von XML Schema nach OWL ("from a lower to a higher level description") eindeutig ist. Der umgekehrte Weg, die Transformation von OWL nach XML Schema, ist aufgrund der Tatsache, dass es für jedes umfangreichere OWL-Modell mehrere - im schlimmsten Fall exponentiell viele - Serialisierungsmöglichkeiten gibt, problematisch. Aussage des Artikels ist, dass dieses Serialisierungsverhalten zu einer weniger breiten Akzeptanz von OWL-S im Anwendungsbereich führt.

Für die aktuelle Arbeit ist die Erkenntnis, dass es für ein gegebenes OWL-Modell mehrere Serialisierungsmöglichkeiten gibt, ebenfalls von Bedeutung. Eine Veranschaulichung der Problematik ist die Tatsache, dass eine OWL-Instanz z.B. nicht vollständig konkretisiert sein muss, um gebildet werden zu können. Unter konkretisieren wird die Nutzung von Eigenschaften der OWL-Klasse im Individual verstanden. In der Tat instanzieren viele Anwendungsbeispiele OWL-Klassen mit nur einer Eigenschaft (z.B. ZipCodeFinder Service der OWL-S API auf Seite 134). Eine entsprechende XSL Transformation ist einfach zu implementieren, da nur ein Wert in der SOAP Nachricht gefunden und transformiert werden muss.

Weiterhin interessant in Bezug auf diese Arbeit sind die Einschränkungen des vorgestellten Konzepts: Akzeptiert werden nur strikt aufgebaute Typhierarchien. Zyklen innerhalb der Typ Definition sind nicht erlaubt. Die für OWLS2WSDL erstellte Translation von OWL nach XML Schema soll Zyklen erlauben. Allerdings führen XML Schema Definitionen mit Zyklen generell zu Problemen bei der Verarbeitung (Parsen und Auswertung der XML Schema Definition).

3.4.2 Übersetzung von OWL nach Java

Mit OntoJava[6] und Jastor[7] findet man Ansätze [KPBP04], um OWL Beschreibungen (Ontologien) in Java Klassenmodelle zu übersetzen. Abgebildet werden OWL Klassenhierarchien und ein Großteil der OWL Sprachelemente.

[6] http://www.aifb.uni-karlsruhe.de/WBS/aeb/ontojava/
[7] http://jastor.sourceforge.net/

3.4.3 OWLS2BPEL

In [YDY+07] wird ein Prototyp für die Transformation von OWL-S nach BPEL4WS vorgestellt. Die Stärke von OWL-S liegt bei der Service-Komposition. Bei der Abbildung eines Workflows zeigen sich jedoch Einschränkungen bei der Fehlerbehandlung *(engl. fault/error handling)* und bei dem *Event Handling*. Diese Funktionen werden von der *Business Process Execution Language* (BPEL4WS) unterstützt. Um die BPEL4WS Funktionalität auch für die in OWL-S definierten Service-Kompositionen nutzen zu können, wurde an der George Mason Universität in Virginia das Konvertierungstool OWLS2BPEL implementiert. Die Ergebnisse einer Konvertierung können mit einer *BPEL Engine* ausgeführt werden.

Vergleich OWLS2BPEL und OWLS2WSDL

OWLS2BPEL generiert anhand von OWL-S Definitionen entsprechende BPEL und WSDL Beschreibungen. Benötigte komplexe Typbeschreibungen liegen in entsprechenden XML Schema Definitionen bereits vor. Fokus von OWLS2WSDL ist die Übersetzung der Signatur eines atomaren Prozesses (Service-Schnittstelle) einschließlich der Übersetzung seiner Parametertypen. Dies beinhaltet die Generierung von XML Schema Typen für die referenzierten OWL Konzepte (OWL2XSD).

3.4.4 METEOR-S Web Service Annotation Framework (MWSAF)

Das *METEOR-S Web Service Annotation Framework* (MWSAF) wurde von dem *METEOR-S Projekt*[8] an der Universität von Georgia entwickelt und unterstützt den Anwender bei der semantischen Annotierung von WSDL Beschreibungen (WSDL-S).
Semantische Informationen mit betreffenden Funktionen sind [POSV04]:

- *Data / Information Semantics (development, description, annotation)*
- *Functional / Operational Semantics (publication, discovery)*
- *QoS Semantics (composition)*
- *Execution Semantics (execution)*

Ziel des METEOR-S Projekts ist es, Technologien aus dem Bereich *Semantic Web Services* so zu erweitern, dass deren Anwendung dynamischer und skalierbarer wird. Die Spezifikation [POSV04] beschreibt die Erweiterung von WSDL und UDDI um semantische Informationen. Unter der Bezeichnung WSDL-S[9] hat das METEOR-S Projekt den Entwurf ihrer Spezifikation von semantisch annotierten Web Services bei dem W3 Consortium eingereicht. WSDL-S wird dort als Alternative zur OWL-S Spezifikation vorgestellt.

[8] http://lsdis.cs.uga.edu/projects/meteor-s/ und http://swp.semanticweb.org
[9] http://www.w3.org/Submission/WSDL-S/

Maßgeblich beeinflusst von den Forschungsergebnissen innerhalb der WSDL-S Spezifikation wurde die Entwicklung der *Semantic Annotations for WSDL* (SAWSDL). SAWSDL ist eine einfache Erweiterung zu WSDL, die zwei Basistypen für eine Annotierung zur Verfügung stellt: die *model reference annotation* assoziiert die Service Schnittstelle (port types, operations, inputs, outputs) und XML Schema Elemente mit Konzepten aus dem Semantic Web mit dem Ziel, die Suche nach Web Services und deren dynamische Konfiguration zu verbessern. Die *schema mapping annotations* dienen dazu, Daten bzw. Instanzen eines Datenmodells zu transformieren (z.B. WSDL auf ein ontologiebasiertes Modell).
Voraussetzung für die Transformation von Datenmodellen ist ein vorhandenes Mapping, das WSDL Konzepte mit den Konzepten aus der Ontologie assoziiert. Das MWSAF beinhaltet für diesen Zweck einen Matchmaker, der zu einer gegebenen WSDL Beschreibung ein möglichst gutes Mapping innerhalb einer ebenfalls gegebenen Ontologie sucht (siehe Abb. 3.5 und [POSV04]). Über eine grafische Benutzerschnittstelle kann der Benutzer dann semi-automatisch den Web Service annotieren. Durch den Einsatz des Matchmakers werden Ähnlichkeitswerte zwischen den gegebenen Ontologien und WSDL Beschreibungen bestimmt.

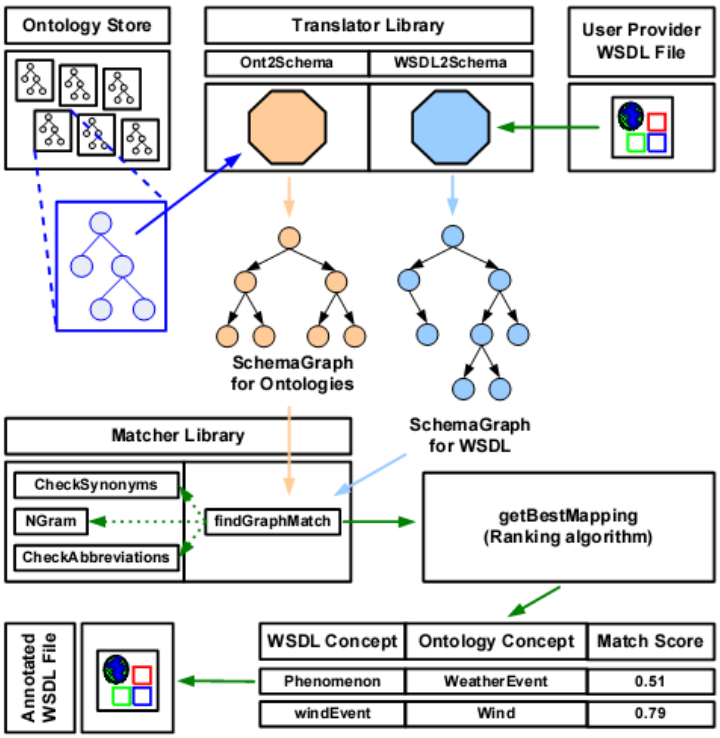


Abbildung 3.5: MWSAF Architecture (METEOR-S) [POSV04]

Vergleich MWSAF und OWLS2WSDL

Es können Parallelen zu dieser Arbeit festgestellt werden. Das MWSAF setzt einen Matchmaker ein, der die Struktur eines Ontologie Graphen mit WSDL vergleicht. Ziel ist es, gefundene XML

Schema Typen und Elemente zu annotieren. Für OWLS2WSDL kann die Annotation bei Generierung der XML Schema Typen direkt vorgenommen werden. Ein WSDL Matchmaker wird für die Evaluation der Translation benutzt (siehe Kapitel 8). Bestandteil der Translation OWL-S nach WSDL (OWLS2WSDL) ist das vollständige Mapping der Service Schnittstelle und eine möglichst gute Abbildung des referenzierten OWL Klassen- bzw. Konzeptmodells.

Hypothese: *Je vollständiger die Abbildung von OWL nach XML Schema (XSD) ist, desto besser ist das Ergebnis eines Matchmakings von WSDL Beschreibungen.*

Eine konzeptionelle Entsprechung zu dieser Arbeit ist die Verwendung eines Zwischenmodells. Um XML Schema Definitionen und OWL Definitionen matchen zu können, konvertiert das MWSAF beide Modelle zu einem einheitlichen Repräsentationsformat, das als *SchemaGraph* bezeichnet wird (Tools sind WSDL2Schema und Ontology2Schema, siehe Abb. 3.5). Die Translation OWLS2WSDL arbeitet ebenfalls mit einem Zwischenmodell (Wissensbasis), das während des Parsevorgangs von OWL-S und OWL erzeugt wird und zum Generieren von XML Schema und WSDL genutzt wird. Es beinhaltet alle Informationen der semantischen Service Definition und eine erste Interpretation der genutzten Parametertypen (Auswertung der OWL Definition).

Das FUSION Projekt

Das FUSION Projekt, ein IST Forschungsprojekt der Europäischen Kommission (6th Framework Programme), das sich mit der Fusion von Geschäftsprozess-Beschreibungen innerhalb einer SOA beschäftigt, untersucht und vergleicht in [FKG+06] Technologien im Bereich von Semantischen Web Services (OWL-S, WSDL-S, WSMO).

3.5 Pflichtenheft

Hintergrund

Die im Forschungsbereich (DFKI) *Deduktion und Multiagentensysteme* angesiedelten Projekte **ATHENA** und **SCALLOPS** nutzen Matchmaker, um für Service Definitionen Ähnlichkeitswerte zu ermitteln. Das ATHENA Projekt nutzt den *WSDL Analyzer* (WA), das SCALLOPS Projekt nutzt den *OWLS-MX*.

Für den OWLS-MX wurde eine umfangreiche Testkollektion (OWLS-TC) zusammengestellt, die bereits Referenzwerte für das hybrid-semantische Matching enthält. Mit dieser Arbeit soll es möglich gemacht werden, die Service-Beschreibungen auch mit dem *WSDL Analyzer* zu analysieren. Die ermittelten Ähnlichkeitswerte sollen mit den Ergebnissen des OWLS-MX verglichen werden können. Die OWL-S Definitionen der OWLS-TC besitzen kein WSDL Grounding. Es existiert auch keine WSDL Beschreibung, anhand der ein WSDL Grounding automatisch generiert werden könnte.

Vorhabensbeschreibung

Ziel ist es, die gegebene Testkollektion OWLS-TC so zu einer WSDL-TC zu übersetzten, dass der *WSDL Analyzer* zur Bestimmung von Ähnlichkeitswerten eingesetzt werden kann. Ein Matchmaking Projekt soll folgendermaßen interpretiert werden können:

- Der *Query Service* (OWLS-MX) entspricht einem *Requirement* (WA).
- Das *Relevance Set* (OWLS-MX) entspricht den *Candidates* (WA).

Da das WSDL Grounding der OWL-S Definitionen fehlt, müssen anhand der OWL-S Prozess-Beschreibung WSDL Beschreibungen generiert werden. Betrachtet wird nur die Schnittstelle eines atomaren Prozesses (Signatur). Die zu jedem Parameter gehörenden OWL Konzepte müssen ebenfalls übersetzt werden, um innerhalb einer WSDL Operation die Typen der Message Parts generieren zu können.

Für alle OWL-S Definitionen sollen automatisch mögliche WSDL Service-Beschreibungen generiert werden. Zusätzlich soll der Benutzer semi-automatisch die Generierung der XML Schema Definition beeinflussen können. Dazu zählen die Konfiguration des XML Schema Generators und die manuelle Zuordnung von primitiven Datentypen zu OWL-Klassen (Mapping), für die kein Datentyp in der OWL Definition ermittelt werden kann. Eine OWL-Klasse ohne Eigenschaften wird zu einem SimpleType übersetzt.

Die Ergebnisse des Matchmakings der generierten WSDL Beschreibungen sollen im Rahmen einer Evaluierung der Translation mit den Ergebnissen des *OWLS-MX* verglichen werden.

Programmfunktionen

Muss-Kriterien

Die Aufgabenstellung OWLS2WSDL fokusiert die Translation der Prozess-Signatur und beinhaltet damit die Translation der durch die Parameter referenzierten OWL Klassen nach XML Schema (OWL2XSD). OWL2XSD arbeitet mit OWL-DL. Möglichst alle Ontologien, die mit Protégé erstellt wurden, sollen verarbeitet werden können.

Funktionspunkte, OWLS2WSDL Tool und experimentelle Evaluierung:

F10 Parsen von Datentypen und Service Informationen aus OWL und OWL-S
- /F11/ Datentypen
- /F12/ Service Beschreibungen
- /F13/ Abhängigkeiten (Parametertypen)

F20 Persistente Speicherung der geparsten Daten
- /F21/ Datentypen (Wissensbasis)
- /F22/ Service-Beschreibungen
- /F23/ Projekte

F30 Konzeptionierung und Implementierung von OWL2XSD

F40 Konzeptionierung und Implementierung von OWLS2WSDL

F50 XML Schema Generierung
- /F51/ Untersuchung der Konfigurationsmöglichkeiten
- /F52/ Implementierung XML Schema Generator

F60 Generierung von WSDL

F70 Erstellung einer grafischen Oberfläche (GUI)
- /F71/ Übersicht der Datentypen (Wissensbasis)
- /F72/ Übersicht der Service-Beschreibungen
- /F73/ Semi-automatische Bearbeitung der Wissensbasis
- /F74/ Konfiguration XML Schema Generierung
- /F75/ Frontend für XML Schema Generator
- /F76/ Frontend für WSDL Builder

F80 Evaluation der Translationsergebnisse
- /F81/ Validierung der generierten WSDL Beschreibungen
- /F82/ Test mit dem *WSDL Analyzer*

F90 Übersetzung der OWLS-TC und Vergleich der Matchmakingwerte

Kann-Kriterien

1: Die generierten WSDL Beschreibungen werden innerhalb dieser Arbeit als Eingabe für den *WSDL Analyzer* genutzt, der Ähnlichkeitswerte ermittelt. Hat man jedoch die Beschreibung eines Web Service in WSDL vorliegen, lässt sich mittels WSDL2Java (*Axis*[10]) auch das Grundgerüst einer Service-Implementierung erstellen (Message Stubs, Klassen für jeden komplexen Datentyp, usw.). Diese Funktionalität könnte auch in das zu erstellende Tool eingebaut werden.

2.a: Anhand der generierten WSDL Beschreibungen soll ein *Re-Engineering* der ursprünglichen OWL-S Definitionen mit WSDL Grounding Teil ermöglicht werden. Dafür notwendig ist das automatisierte Mappen von XML Schema Typen auf OWL-Klassen. 2.b: Eine weitere Aufgabe besteht darin, automatisch Transformationsregeln generieren zu lassen, die eine Instanzierung von OWL-Klassen (OWL Individuals) aus den Daten des enstprechenden WSDL Services (SOAP Messages) ermöglichen. Diese Transformationsregeln sind Bestandteil des WSDL Groundings.

Abgrenzungs-Kriterien

1: OWLS2WSDL betrachtet nur die Signatur von **atomaren Prozessen** innerhalb des OWL-S Prozessmodells. Ein CompositeProcess soll in der ersten Version der Translation nicht weiter untersucht werden. Die Service Beschreibungen der OWLS-TC bestehen bis auf eine Ausnahme nur aus jeweils einem atomaren Prozess.
Ausnahme bildet die Servicedefinition CheckHospitalAvailability_service.owls, die eine Prozess-Komposition beinhaltet.

2: Eine Translation der Vor- und Nachbedingungen zur Definition von Workflows ist nicht Teil dieser Arbeit. Die Tools OWLS2BPEL[11] (Seite 30) und PBEL4WS2OWLS[12] übersetzen die OWL-S Logik zum Erstellen von Workflows [SYWZ05].

[10] http://ws.apache.org/axis/
[11] http://www.laits.gmu.edu:8099/OWLS2BPEL/
[12] http://bpel4ws2owls.sourceforge.net

Kapitel 4

Vorgehensmodell

Bei Aufgabenstellungen, bei denen es nicht abzusehen ist, wie lange die Bearbeitung dauert und wie umfangreich und zeitintensiv eine Implementierung werden kann, ist der Gebrauch eines Vorgehensmodells hilfreich. Es koordiniert die Arbeitsschritte, die nötig sind, um die Aufgabe zu lösen, und ist Grundlage für die Bildung und Gewichtung von Arbeitspaketen. Weiterhin lassen sich durch die Betrachtung des Gesamtbilds Vorhersagen zu der Bearbeitungszeit von einzelnen Arbeitsschritten machen (z.B. bei weiteren Anpassungen).

Aufgabenstellungen dieser Arbeit:
Die Konzeptionierung und die Implementierung der Translation von OWL-S nach WSDL und die praktische bzw. experimentelle Evaluierung der Translation durch Analyse der Matchmaking-Ergebnisse einer ähnlichkeitsbasierten Dienstsuche mit dem *WSDL Analyzer*. Die Resultate sollen mit den entsprechenden Ergebnissen des *OWLS-MX* verglichen werden. Die ständige Validierung, Evaluation der generierten WSDL Beschreibungen und die darauf folgenden Anpassungen der Translation führen zu einem Vorgehensmodell, das an das *Clean Room Vorgehensmodell* angelehnt ist. Kennzeichnend für dieses Vorgehensmodell ist, dass mehrere Iterationen nötig sind, um die Aufgabenstellung zu erfüllen bzw. um sich einem brauchbaren Ergebnis anzunähern.
Die wichtigsten Arbeitspakete sind:

- Interpretation von Ontologien und Übersetzung von OWL-Klassen, Eigenschaften, Vererbungshierarchien und anonymen Klassen, die OWL-Logik ausdrücken.
- Generierung von XML Schema Typen, die bezüglich der ursprünglichen OWL-Klassendefinition möglichst vollständig sind. D.h. die generierten WSDL Beschreibungen sollen die ursprünglichen OWL-S Definition mit deren Parametern und Parametertypen möglichst gut interpretieren.
- Generierung von WSDL Beschreibungen, die alle Merkmale der WSDL Spezifikation erfüllen und validieren *(XML Spy, Castor CodeGen, Axis)*.
- Generierung von WSDL Beschreibungen, die mit dem *WSDL Analyzer* verarbeitet werden können. Betrachtet werden die beiden Programmteile WA Parser und WA Matcher.
- Matchmakingwerte des *WSDL Analyzers* sollen den *OWLS-MX* Werten entsprechen bzw. sich den OWLS-MX Werten annähern (Evaluation).

Zu Beginn der Arbeit konnten nur wenige Aussagen zu dem genauen Funktionsumfang des Tools sowie zu dem Format der zu generierenden XML Schema Definitionen gemacht werden. Der Rahmen bildete die Zielbestimmung: mit Hilfe der Translation OWLS2WSDL WSDL Beschreibungen zu generieren, die mit dem *WSDL Analyzer* analysiert werden können und möglichst aussagekräftig sind.

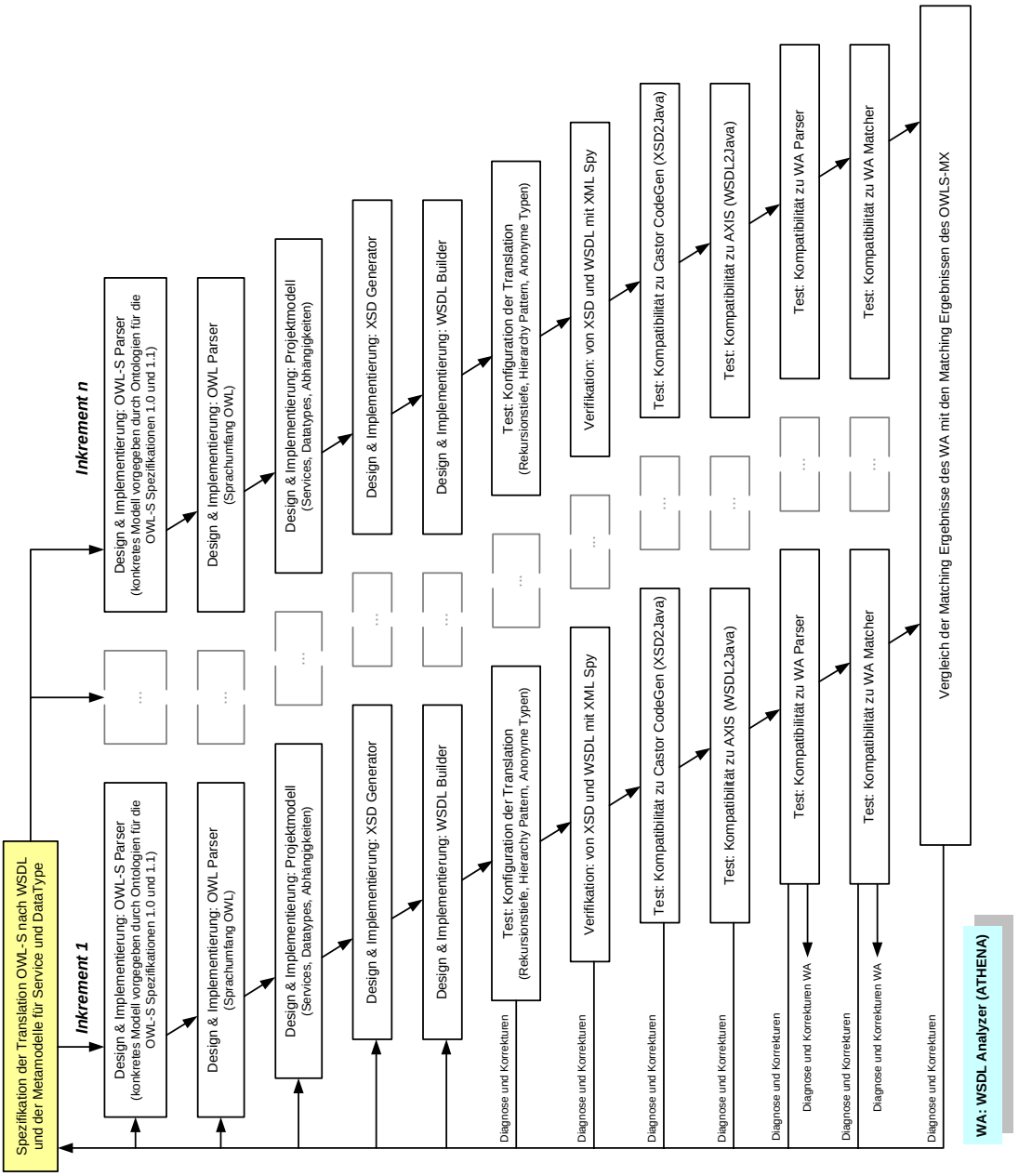


Abbildung 4.1: Genutztes Vorgehensmodell

Das Vorgehensmodell berücksichtigt die Entwicklung des OWLS2WSDL Tools und die Erweiterung des *WSDL Analyzers* (inkl. Bug Fixing).

Kapitel 5

Konzeptionierung der Translation (Design)

Das Kapitel beschreibt ein mögliches Konzept für die Translation OWL-S nach WSDL. Neben der Übersetzung der Service-Signatur ist die Übersetzung der Parametertypen, also die Translation OWL nach XML Schema, von großer Bedeutung.

5.1 Design Direktiven

Obwohl die Entwicklung der Translation *OWL-S nach WSDL* unabhängig von einem bestimmten Projektkontext ist, empfehlen sich folgende Design Direktiven:

1. Beachtung vorhandener Mapping Direktiven bzgl. des WSDL Grounding (siehe Abb. 2.6).
2. Wohl überlegte Nutzung eines OWL Reasoners beim Parsen von OWL-DL.
3. Einsatz einer plattform- und spezifikationsunabhängigen Wissensbasis für ein Projekt.
4. Trennung von Parsen und Code-Generierung (Abb. 5.1).
5. Trennung der Basisfunktionalität von der Benutzeroberfläche (MVC).

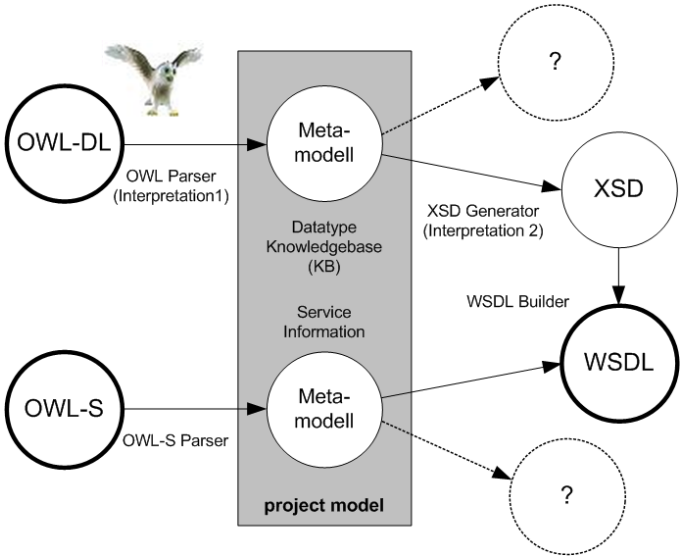


Abbildung 5.1: Design. Trennung von Parsen und Code Generierung.

5.2 OWL-S Use Cases

Mit Hilfe eines klassischen Use Case Diagramms lässt sich die Translation *OWL-S nach WSDL* und die verwandte Translation *OWL-S nach BPEL* innerhalb des Entwicklungs-Paradigma einordnen. Die Abbildung 5.2 zeigt die Rollen, die für die Implementierung von Anwendungen und Web Services innerhalb einer SOA mit *Semantic Web* Support benötigt werden. Dazu zählen (1) Entwickler der Web Services, (2) Anwendungsentwickler (für Provider und Consumer Programme), (3) SOA Experten, die Geschäftsprozesse durch Orchestrierung von Services innerhalb der SOA abbilden und (4) Entwickler von Ontologien, die alle Dienste und Daten semantisch einordnen (Domain-Experten). Der Service Consumer nutzt Matchmaker (für OWL-S oder WSDL) und/oder das UDDI Verzeichnis, um die Dienste aufzufinden, die er nutzen möchte.

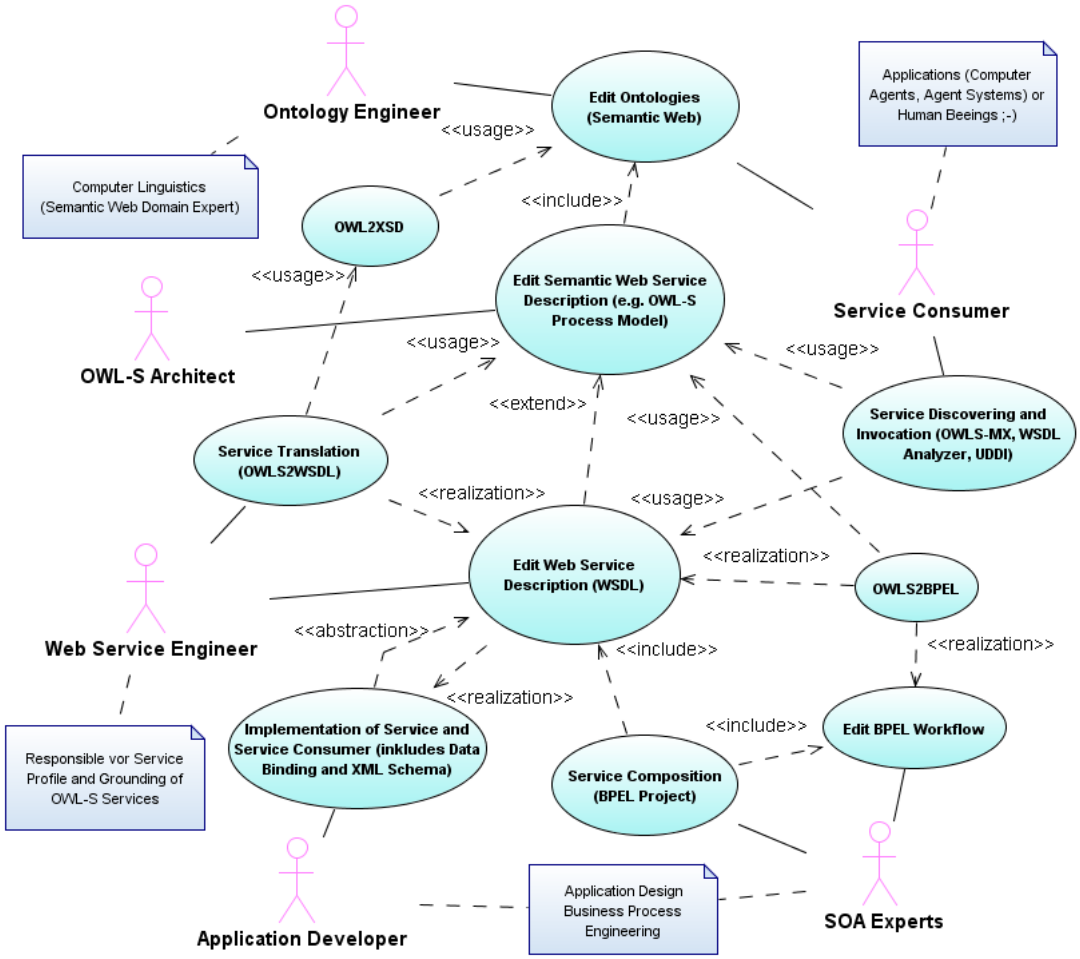


Abbildung 5.2: Entwicklung von Anwendungen und Web Services, die OWL-S nutzen

Use Case dieser Arbeit: "Übersetze die OWL-S Definitionen, die für die OWLS-MX Testkollektion erstellt worden sind nach WSDL und nutze den *WSDL Analyzer*, um Ähnlichkeitswerte zwischen WSDL Beschreibungen zu bestimmen, die mit den Ähnlichkeitswerten des *OWLS-MX* für die entsprechenden OWL-S Definitionen vergleichbar sind."

5.3 Übersicht

Anhand der ermittelten Anforderungen aus Kapitel 3 ergeben sich folgende Translationsschritte:

1. Parsen der im Prozessmodell semantisch definierten Service-Schnittstelle (Signatur).
2. Untersuchung von Abhängigkeiten. Ermittlung der Parameter Typen/Konzepte.
3. Interpretation von Parametertypen (OWL, semantische Beschreibungsebene).
4. Speicherung aller Service-Informationen in einer Service-Kollektion und Speicherung von Datentyp-Informationen in einer Wissensbasis (KB).
5. Generierung von WSDL Beschreibungen (strukturelle Beschreibungsebene).
6. Generierung von Java-Quellcode, der die Kommunikationsschnittstelle und das *Data Binding* einer möglichen Service Implementierung realisiert (Implementierungsebene).

Die Translationsschritte beschreiben einen Protokollstack (Abb. 5.3).

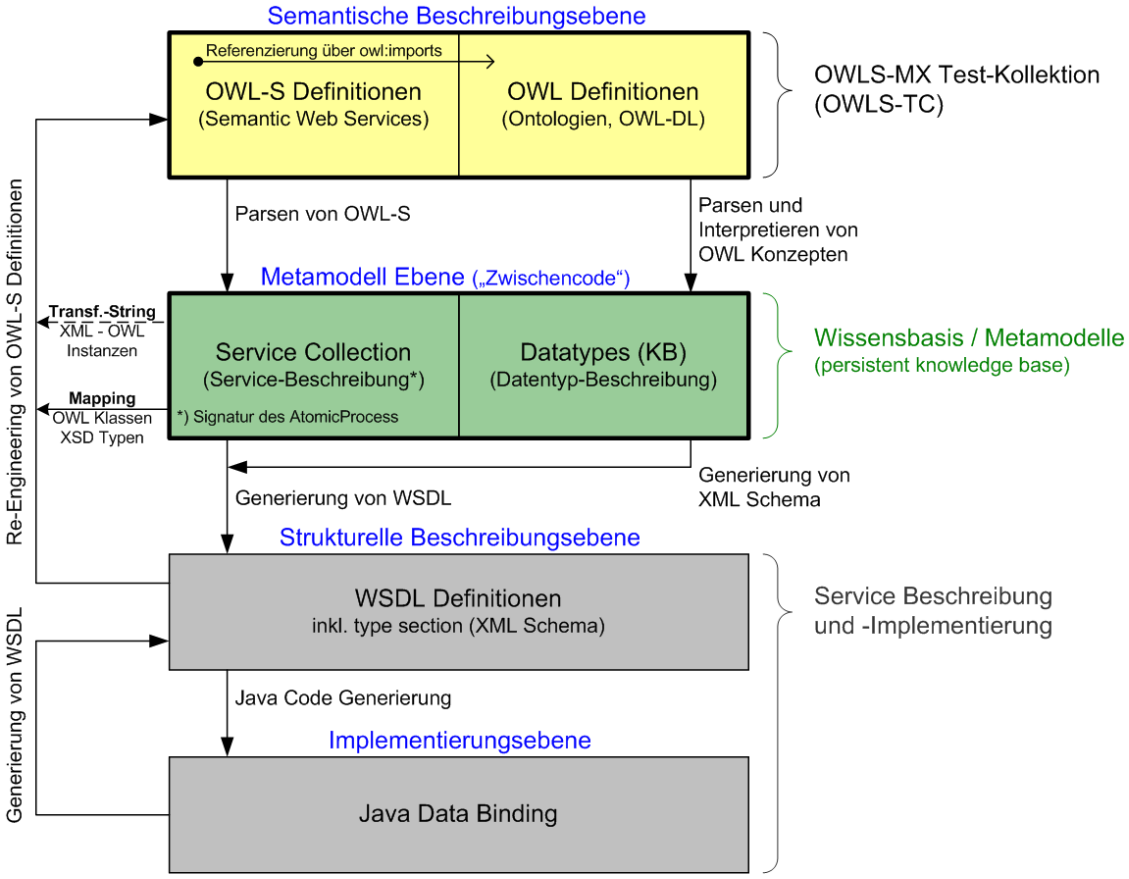


Abbildung 5.3: Translations-Stack OWL-S nach WSDL

Die unabhängige Wissensbasis wird eingesetzt, um die gesammelte Informationen flexibel speichern und verarbeiten zu können. Dies ermöglicht die Verarbeitung von Daten aus unterschiedlichen OWL-S Versionen (1.0, 1.1) und trennt die Interpretation der abstrakten OWL-Definition

von der Generierung der XML Schema Definition (siehe Abb. 5.3). Die Interpretation der semantischen Beschreibung kann so ständig verbessert werden, ohne dass sie direkten Einfluss auf die Generierung von XML Schema hat.

5.4 Translation von OWL nach XML Schema (OWL2XSD)

Grundproblem ist das Mapping von OWL-Konzepten (früher OIL) auf XML Schema Typen. In [KFHH] werden Mapping-Vorschläge für die Richtung OWL nach XML Schema diskutiert. Grund für den Ansatz sind folgende Vorteile von XML Schema:

- XML Schema Definitionen sind XML Dokumente.
- Die XML Schema Spezifikation stellt ein breites Spektrum an Datentypen zur Verfügung.
- Mit XML Schema können komplexe Strukturen aufgebaut werden.
- Aufbau unabhängiger "Wortschätze" mit Hilfe des Namespace Mechanismus.
- Die XML Schema Spezifikation unterstützt das Konzept der Ableitung über eine Restriktierung oder Erweiterung eines bereits definierten Typs.

Heute ist man eher an dem umgekehrten Weg interessiert: Für diejenigen XML Schema Typen, die in WSDL Beschreibungen genutzt werden, sollen OWL Definitionen erstellt werden. Eine, um die semantische Einordnung in das *Semantic Web* erweiterte, Typdefinitionen erlaubt es den Nutzern des Web Service (z.B. Agenten), die Daten der Schnittstelle zu verstehen bzw. semantisch einzuordnen. XML Schema bietet eine gute Basis, um Ontologien neu zu entwickeln oder zu erweitern. In [FZT04] wird ein Mapping-Konzept für ein in XML Schema definiertes Datenmodell und OWL vorgestellt.

5.4.1 Interpretation des OWL Klassenkonzepts (OWL Parser)

XML Schema Instanzen (XML Dokumente) basieren auf einer Baumstruktur. OWL Instanzen werden hingegen durch ein Graphenmodell ausgedrückt (den RDF Graphen), dessen Knoten für Konzepte und eine Kante für die Beziehung zwischen zwei Konzepten steht. Beziehungen werden als Eigenschaften *(properties)* interpretiert. Ein Knoten der Kante steht dabei für das zu beschreibende Konzept *(Domain)*, der andere Knoten ist Wert der Eigenschaft *(Range)*.
In der OWL-Terminologie wird der Begriff der *Klasse* (`owl:class`) benutzt, um ein Konzept zu beschreiben, aber auch immer dann eingesetzt, wenn eine bereits definierte Klasse durch eine OWL-DL Beschreibung erweitert oder eingeschränkt wird. Durch Nutzung von Mengenoperatoren können OWL-Klassen zu anonymen neuen Klassen zusammengefasst werden.

OWL unterscheidet sechs Typen von explizit und implizit definierten Klassen:

1. Beschreibung einer Klasse über einen *class identifier* (URI)
2. Bildung einer Klasse über die Aufzählung von Individuen *(Enumeration)*

3. Restriktion eines Property (anonyme Klasse, die alle Bedingungen der Restriktion erfüllt)
4. Schnittmenge von zwei oder mehreren Klassen (`owl:intersection`)
5. Vereinigungsmenge von zwei oder mehreren Klassen (`owl:union`)
6. Komplement einer Klassen (`owl:complementOf`)

Das OWL Klassenmodell soll, ähnlich wie in [KFHH] und [FZT04] beschrieben, ausgelesen werden, und die gesammelte Information in der Wissensbasis (KB) abgelegt werden.

Übersetzung von Klassen und Eigenschaften

OWL-Klassen werden zu XML Schema Typen übersetzt. Werden zu einer OWL-Klasse *(Domain)* Eigenschaften (`DatatypeProperty` oder `ObjectProperty`) gefunden, soll die Translation einen `ComplexType` bilden und Properties als Elemente übersetzt hinzufügen. Werden keine Properties für eine OWL-Klasse gefunden, wird ein `SimpleType` erzeugt und anhand der OWL Definition ein Basistyp (`xsd:restriction base`) ermittelt. Da in der Regel für nur wenige OWL-Klassen primitive Datentypen definiert sind (in OWL mögliche primitive Typen entsprechen den primitiven Typen der XML Schema Spezifikation), soll je nach Konfiguration der Translation ein Standardtyp gewählt oder von dem Anwender semi-automatisch ein primitiver Datentyp zugeordnet werden. Die OWL-Klasse, die den *Range* eines `ObjectProperty` beschreibt, wird ebenso interpretiert.

XSD ELEMENTNAME	XSD TYPBEZEICHNUNG	OWL KONZEPT
ID <Property name>	<Class name> | <Class name>Type	owl:Class *(Range)*

Tabelle 5.1: XSD Gen. Bildungsvorschrift: Name des komplexen Schema Typs

Übersetzung der Klassenhierachie

Die in OWL durch `rdfs:subClassOf` abgebildete Spezialisierung beschreibt mit den beteiligten OWL-Klassen eine Klassenhierarchie (Taxonomie). Alle Eigenschaften der Oberklassen werden dabei an die Unterklassen vererbt. Eine der wichtigsten Aufgaben der Translation von OWL nach XML Schema ist es, festlegen zu können, mit welcher Vererbungstiefe Properties von Oberklassen geerbt werden. Die Klassenhierarchie selbst kann in XML Schema über den Einsatz des Entwurfsmusters *Hierarchy Pattern* mit Hilfe von `substitutionGroup` und `restriction base` abgebildet werden (siehe Abschnitt 5.4.2).

Übersetzung von Kardinalitäten und des Wertebereichs

Die OWL Kardinalitäten `minCardinality`, `maxCardinality` und `cardinality` können in XML Schema auf die Attribute `minOccurs` und `maxOccurs` gemappt werden. Die Kardinalität wird in einem Element durch den Wert von `minOccurs` und `maxOccurs` ausgedrückt. Bei Interpretation einer Äquivalenz *(Intersection)* besteht zusätzlich eine gesonderte Behandlung der Restriktion.

`allValuesFrom`, `someValuesFrom` und `hasValue` schränken den Wertebereich *(Range)* ein, indem sie ihn über die Angabe eines Individuals bzw. eines Literals oder eine Menge von Individuen *(Collection)* neu definieren. Dies lässt sich in XML Schema über einen `SimpleType` abbilden, der seinen Wertebereich über eine Enumeration von Facetten restringiert. Bei Äquivalenzen werden Restriktion zusätzlich über anonyme Typen beschrieben. Ein Beispiel zu der Einschränkung des Wertebereichs zeigen die Listings A.2 und A.3.

Übersetzung von anonymen Klassen

OWL ermöglicht mit den Mengenoperatoren `intersectionOf` und `unionOf` die Bildung von Schnittmengen und Vereinigungsmengen, sowie mittels `complementOf` die Definition einer Komplementärmenge. Alle diese Konzepte führen zur Bildung von anonymen Klassenkonzepten (Metamodellebene), die sich durch die Zugehörigkeit anderer OWL-Klassen definieren. Für anonyme Klassen werden neue Schema Typen gebildet. Diese Vorgehensweise ist im Hinblick auf ein späteres Matchmaking von Vorteil.

XSD ELEMENTNAME	XSD TYPBEZEICHNUNG	OWL KONZEPT
hasSufficientIntersection	Sufficient<Class name>Intersection	owl:intersectionOf
hasSufficientUnion	Sufficient<Class name>Union	owl:unionOf
hasDisjointClass	DisjointOf<Class name>	owl:disjointOf

Tabelle 5.2: XSD Gen. Bildungsvorschrift: Namensgebung von neuer Schema Typen

Zwei Fälle müssen bei der Generierung von neuen XML Schema Typen aus anonymen OWL-Klassen unterschieden werden: (1) Alle Sub-Klassen werden über Identifier definiert: falls noch nicht im Schema enthalten, muss für jede beteiligte OWL-Klasse ein neuer XML Schema Typ angelegt werden. (2) Eine Restriktion wird als `ComplexType` übersetzt, der in einem Element das Property beschreibt und in einem weiteren Element die Art der Restriktion ausdrückt.

XSD ELEMENTNAME	XSD TYPBEZEICHNUNG	OWL KONZEPT
elem<n> (n ist int)	<Class> | <Class>Type	owl:Class
elem<n> (n ist int)	<Property><Class><RestrictionType>Restriction	owl:Restriction

Tabelle 5.3: XSD Gen. Bildungsvorschrift: Namensgebung neuer Sub-Typen

Abbildung der OWL Grammatik mit Hilfe von Schema Typen
In [PSH04] findet man eine abstrakte Beschreibung der OWL Grammatik. Die Notation ist angelehnt an die erweiterte Backus Naur Form (Extended BNF bzw. EBNF). Beschrieben werden alle Axiome, Fakten und Annotationsmöglichkeiten, die die *Web Ontology Language* (OWL) zur Verfügung stellt. Anhand dieser Beschreibung wurden XML Schema Typen erstellt, die den Sprachumfang der OWL-DL widerspiegeln. Die so definierten abstrakten Schema Typen können nun als Basistypen für die Schema Typen, die anonyme OWL-Klassen beschreiben, dienen.
Bemerkung: Die Übersetzung von anonymen Typen ist sehr experimentell und richtet sich danach, wie gut die generierten Schema Typen gematcht werden können. Die Funktionalität soll deshalb über die Konfiguration des Schema Generators ein- und ausgeschaltet werden können.

Übersetzung von OWL Individuals

Individuals selbst werden in der aktuellen Version nicht komplett übersetzt, jedoch bei der Übersetzung nach XML Schema beachtet, um die Information über deren Existenz nicht zu verlieren. Gelöst wurde dies über die Einführung eines neuen `SimpleType`, der in einer Enumeration alle IDs der Individuals enthält. Ein Element des neuen Typs, welches die übersetzten Eigenschaften beschreibt, wird in einem `ComplexType` über die Verwendung des Operators `choice` zusätzlich zu der `sequence` zur Verfügung gestellt.

Kompatibilität und Geltungsbereich der Translation

Eine Ontologie kann mit OWL auf unterschiedlichem Wege definiert werden. Der für die Translation *OWL nach XML Schema* benutzte Parser sollte möglichst allgemeingültig sein und die mit Protégé[1] erstellten und als OWL-DL gespeicherten Ontologien lesen können.

```
<owl:Class rdf:ID="Imagery"/>

<owl:DatatypeProperty rdf:ID="isImageClassification">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  <rdfs:domain rdf:resource="#Imagery"/>
</owl:DatatypeProperty>

<!-- Necessary -->
<owl:Class rdf:ID="ImageClassification">
  <rdfs:subClassOf rdf:resource="#Imagery"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isImageClassification"/>
      <owl:hasValue rdf:datatype="&xsd;boolean">true</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<!-- Necessary & Sufficient -->
<owl:Class rdf:ID="ImageClassification">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Imagery"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#isImageClassification"/>
          <owl:hasValue rdf:datatype="&xsd;boolean">true</owl:hasValue>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Listing 5.1: OWL Defintionen: Necessary, Necessary and Sufficient

[1] http://protege.stanford.edu

Der Unterschied zwischen Necessary und Necessary & Sufficient (Listing 5.1) wird auf Seite 14 besprochen. Beide Definitionsmöglichkeiten werden ausgewertet. Die Art der Relation wird in der vorliegenden Version nicht mit übersetzt.

Diskussion

Entsprechen nun OWL-Klassen den Typen einer generierten XML Schema Definition und können OWL-Instanzen vollständig in XML ausgedrückt werden?

Die vorgestellte Translation *OWL nach XML Schema* zeigt eine mögliche Abbildung von OWL-Klassen und Eigenschaften. Durch die als Design Direktive vorgestellte Wissensbasis (Abb. 5.1) sind Änderungen der Translation abhängig von zwei konzeptionellen Aufgaben:

1. Die Implementierung des OWL Parsers legt fest, wie gut eine Ontologie analysiert wird und demnach, wie gut ein OWL-Konzept interpretiert werden kann. Dieser Teil hat Einfluss auf die Auswahl und Zuordnung von Eigenschaften, Restriktionen und Individuen einer OWL-Klasse. In der Wissensbasis werden die Informationen in einem Objektmodell organisiert.

2. Der Generator arbeitet mit den Informationen aus der Wissensbasis (Objektmodell) und nutzt die Beschreibungselemente, die XML Schema bietet, um für jeden Datentyp aus der Wissensbasis einen XML Schema Typen zu generieren.

Die mit XML Schema erstellte Definition beschreibt eine XML-Dokumentstruktur. Eine solche XML-Instanz muss die Werte einer instanzierten OWL-Klasse (Individual) speichern können. Da ein OWL Individual aber nicht zwingend aus allen möglichen Eigenschaften besteht, muss ein entsprechender XML Schema Typ nur hinreichend gut die Eigenschaften einer OWL-Klasse widerspiegeln. Jeder XML Schema Typ, der die tatsächlich genutzten Parameter einer semantischen Web Service Definition beschreibt, ist für eine WSDL Beschreibung geeignet, die von dem WSDL Grounding mit der Prozessbeschreibung gemappt wird.

Beispiele

Beispiele finden sich im Anhang A.

Das Beispiel A.3 zeigt die Übersetzung von anonymen OWL-Klassen und die Abbildung der OWL-Grammatik anhand von XML Schema Typen für das OWL-S Konzept Process.

- Der anonyme Typ SufficientProcessUnion mit dem Basistyp Union besteht aus drei Elementen. Deren Typen sind im Schema enthalten aber selbst keine anonymen Typen.

- Die Restriktion der Kardinalität eines `DatatypeProperty` wird durch Referenzierung der Eigenschaft mit `onProperty` und Nutzung von `cardinality` festgelegt. Der neue Schema Typ, der die Restriktion beschreibt, hat den Schema Typ `Restriction` als Basistyp und die Elemente `onProperty` vom Typ `xsd:anyURI` und `cardinality` vom Typ `Cardinality` mit jeweils fixierten Werten (URI, int value).

5.4.2 Genutzte Entwurfsmuster für XML Schema

Grundlegende Anforderungen für den Aufbau eines XML Schemas sind:

- Wiederverwendbarkeit
- Flexibilität
- Wartbarkeit
- Portabilität

Im Rahmen der Übersetzung *OWL nach XML Schema* können diese wichtigen Punkte um folgende Anforderungen ergänzt werden:

- Generierbarkeit
- Komplexität der Generierung (Implementierung und Schema!)
- In Zusammenhang mit der Bildung eines Transformationsstrings (XSLT), um aus XML Instanzen OWL Individuals zu instanzieren: Auslesbarkeit (Dumpfähigkeit)

Die Basics: Venetian Blind und Garden of Eden

Als grundlegendes Entwurfsmuster für das XML Schema wurde *Venetian Blind* gewählt (Beispiel Abbildungen 5.4 und 5.5). *Venetian Blind* vermeidet das Ineinanderschachteln von komplexen Typinformationen mit Hilfe von anonymen Typbeschreibungen, was Redundanz vermeidet und eine Generierung der Typdefinition vereinfacht. Das Gegenteil ist das Design eines XML Schemas nach dem *Russian Doll* Muster. Das *Venetian Blind* Muster beinhaltet jedoch lediglich eine globale Elementdefinition. Das Gegenteil hierzu ist das *Garden of Eden Muster*, das für jeden Elementtyp (auch primitive Typen) eine eigene globale Elementdefinition anlegt [KS96, PS04].

Ausgehend von der Anforderung, ein XML Schema für die Parameter der WSDL Schnittstellen-Beschreibung zu definieren, genügt es jedoch, die Elemente für die Schnittstelle global zu definieren. Im Rahmen der Translation wird somit ein *erweitertes Venetian Blind Muster* gewählt. Weitere Anforderungen an die Anzahl der global definierten Elementtypen stellt die Verwendung des *Hierarchy Pattern* (Abschnitt 5.4.2).

Bemerkung: Einen sehr guten Schema-Editor, der es auf einfache Art und Weise erlaubt ein bestehendes XML Schema in ein vordefiniertes Muster zu transformieren, ist Netbeans (Sun) mit installierter Enterprise Pack Erweiterung[2].

[2] http://xml.netbeans.org

Das Beispiel 5.4 war ein erster Ansatz für die vorgestellte Translation. Übersetzt wurde die OWL-Klasse http://www.daml.org/2001/10/html/zipcode-ont#ZipCode

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2
3  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4      targetNamespace="http://schemas.sun.com/point/venetianblind"
5      xmlns:tns="http://schemas.sun.com/point/venetianblind"
6      xmlns="http://schemas.sun.com/point/venetianblind"
7      elementFormDefault="qualified">
8
9      <xsd:element name="ZipCode">
10         <xsd:complexType>
11             <xsd:sequence>
12                 <xsd:element name="zip" type="xsd:string" />
13                 <xsd:element name="city" type="xsd:string" />
14                 <xsd:element name="defaultAssociation" type="Association" />
15                 <xsd:element name="association" type="Association" />
16             </xsd:sequence>
17         </xsd:complexType>
18     </xsd:element>
19     <xsd:complexType name="Association">
20         <xsd:sequence>
21             <xsd:element name="acceptable" type="xsd:boolean" />
22             <xsd:element name="state" type="xsd:string" />
23             <xsd:element name="city" type="xsd:string" />
24             <xsd:element name="type" type="xsd:string" />
25         </xsd:sequence>
26     </xsd:complexType>
27
28 </xsd:schema>
```

Abbildung 5.4: XSD Venetian Blind, Quellcode

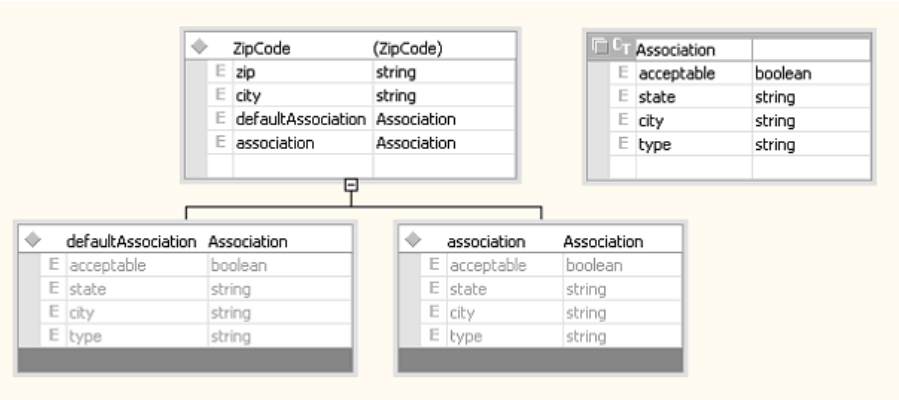


Abbildung 5.5: XSD Venetian Blind, Typ-Darstellung (XMLSPY)

Das XML Schema Hierarchy Pattern

Zusätzliche Anforderung ist die Übersetztung der, mittels `rdfs:subClassOf` festgelegten, Klassenhierarchie. Hier kann das XML Schema Entwurfsmuster *Element Hierarchy Pattern*[3] angewendet werden. Für diesen Zweck wird das Konzept der `substitutionGroup` verwendet, das es Instanzen eines Elements von Typ A erlaubt, die Elemente eines Typs B zu benutzen. Wichtig bei dieser Vorgehensweise, um eine Hierarchie abzubilden, ist es jedoch, dass die Elemente der Oberklasse nicht automatisch vererbt werden, sondern in der Unterklasse wieder aufgeführt werden müssen!

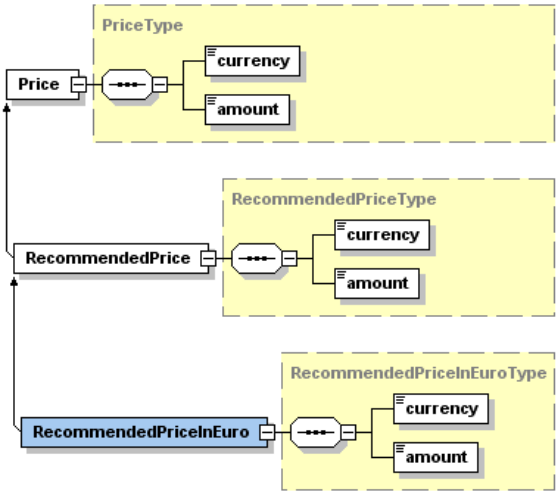


Abbildung 5.6: Beispiel Hierarchy Pattern

Die Abbildung 5.6 zeigt eine Klassenhierarchie ausgehend von der Klasse Price. Die Oberklasse vererbt ihre Eigenschaften an die Unterklassen weiter. Das komplette Beispiel hierzu ist im Benutzerhandbuch in Abschnitt 7.3 auf Seite 70 aufgeführt.
In der aktuellen Implementierung wird diese Vorgehensweise durch das, in XML Schema zur Verfügung stehende, Konzept des Basistyps (`restriction base`) ergänzt. Ein Typ definiert einen anderen Typ als Basistyp, um die Elemente des Basistyps restriktieren oder erweitern zu können. In der Regel werden Properties in OWL durch ihre Unterklasse über eine `owl:Restriction` eingeschränkt (z.B. die Kardinalität oder der Wertebereich).

Mit dem *Hierarchy Pattern* als Werkzeug lässt sich auch die auf Seite 43 vorgestellte Abbildung des OWL Sprachumfangs (Grammatik) nach XML Schema unterstützen (Abbildung A.7).
Das entwickelte *Hierarchy Pattern* ist kompatibel zu dem XML Schema Mapping Konzept und dem *Source Code Generator* des Castor[4] Projekts mit dessen Hilfe Quellcode für Java Klassen aus XML Schema generiert werden kann. Durch die Nutzung des *Source Code Generator* sollten generierte Java Klassen die in XML Schema definierte Typ-Hierarchie widerspiegeln.

[3] `http://www.xfront.com/ElementHierarchy.html`
[4] `http://www.castor.org`

5.4.3 Konfiguration der Translation und manuelle Änderungen

Für die Konfiguration der Translation *OWL nach XML Schema* ergeben sich an dieser Stelle nun folgende Konfigurationsmöglichkeiten:

- Konfiguration eines primitiven *default type* für Klassen ohne Typinformation.
- Konfiguration der Vererbungstiefe von Elementen.
- Ein- und Ausschalten des *Hierarchy Pattern*.
- Übersetzung von anonymen OWL-Klassen *(Intersection, Union, ...)*.
- Bezug zur OWL-Grammatik über *Hierarchy Pattern* herstellen.

Man kann dann von einer semi-automatischen Translation sprechen, sobald der Anwender Einfluss auf die generierten Datentypen hat. Dies ist für `SimpleType` Typen der Fall. Diese werden immer dann gebildet, wenn der XML Schema Generator bei einer konfigurierten Vererbungstiefe für eine zu übersetzende OWL-Klasse keine Eigenschaften (Sub-Elemente) finden kann, die einem `ComplexType` zugeordnet werden könnten. Einem `SimpleType` muss dann ein primitiver Basistyp manuell zugeordnet werden, wenn kein primitiver Typ in OWL definiert wurde. Falls kein primitiver Typ manuell zugeordnet wurde, wird der XML Schema Typ mit dem festgelegten *default type* erstellt.
Wichtig:
Jede manuell durchgeführte Änderungen der Wissensbasis und Konfiguration des XML Schema Generators beeinflusst das spätere Matchmakingverhalten! Für das Matchmakingverhalten ist der Typ eines Elements (SimpleType oder ComplexType) und die Struktur eines komplexen Typs von Bedeutung.

5.5 Translation OWLS2WSDL

Die Übersetzung der WSDL Prozess-Schnittstelle bzw. deren Signatur kann anhand des vorgestellten Mappings (siehe Abschnitt 3.3) *straight forward* durchgeführt werden. Im Gegensatz zu der Übersetzung von OWL nach XML Schema müssen keine Konzepte interpretiert werden.

Da die OWL-S Prozess-Schnittstelle selbst keine Vorgabe für die Reihenfolge der Prozess Ein- und Ausgaben macht, wird diese nach der Parse-Reihenfolge in dem Metamodell gespeichert. Die Reihenfolge wird bei der Generierung von WSDL genutzt, um ein Mapping zwischen WSDL Operationen und atomaren OWL Prozessen zu garantieren. Weiterhin muss ein Mapping der Parametertypen erstellt werden. Diese Informationen müssen in der abstrakten Beschreibung des Services im Metamodell festgehalten werden.

Die Parameter der Prozess-Schnittstelle werden, wie in Kapitel 5.4 ab Seite 41 vorgestellt, gemäß der Konfiguration generiert (Variationsmöglichkeiten bestehen bzgl. der konfigurierten Vererbungstiefe, der Nutzung des *Hierarchy Pattern*, Abbildung von anonymen Typen un Interpretation der OWL-Grammatik).

Die so erstellte WSDL Prozess-Schnittstelle und alle Parametertypen können im WSDL Grounding vollständig gemappt werden. Bei vielen Anwendungsbeispielen, bei denen ein *bottom up* Ansatz bei der Entwicklung der Service-Schnittstelle gewählt wurde, ist dies nicht der Fall. Ein Beispiel ist die Instanzierung der OWL-Klasse ZipCode als Output Parameter der Service Beschreibung ZipCodeFinder (Abbildungen 5.4 und B.1).

5.6 Re-Engineering WSDL2OWL-S

Das *Re-Engineering* erzeugt anhand der WSDL Beschreibung eine OWL-S Definition inklusive WSDL Grounding. Dafür genutzt wird der *WSDL2OWL-S Converter* (OWL-S API).

Beispiel:

```
<grounding:WsdlGrounding rdf:ID="ZipCodeFinderGrounding">
<service:supportedBy rdf:resource="#ZipCodeFinderService"/>
<grounding:hasAtomicProcessGrounding rdf:resource="#ZipCodeFinderProcessGrounding"/>
</grounding:WsdlGrounding>
...
```

Durch die Nutzung der Mapping-Information aus der Service-Kollektion und der Wissensbasis für Datentypen lässt sich für den *WSDL2OWL-S Converter* ein Mapping zwischen den generierten XML Schema Typen und den ursprünglichen OWL-S Parametern und OWL-Klassen festlegen. Die Reihenfolge der Parameter muss dabei eindeutig sein (siehe Abschnitt 5.5).

Kapitel 6

Implementierung

Dieses Kapitel konzentriert sich auf die Beschreibung der Implementierung der im Kapitel 5 (Design) erarbeiteten Programmteile und stellt die dafür genutzten Technologien vor.

Die Implementierung wurde mit Java 1.5 durchgeführt, da ein Großteil der Technologien aus den Bereichen *Semantic Web* und *Semantic Web Services* auf Java basiert. Dies ermöglicht die Nutzung von Bibliotheken, die Objektmodelle *(models)* der verwendeten Spezifikationen implementieren und entsprechende Zugriffsmethoden anbieten. Alle genutzten Bibliotheken (APIs) und Frameworks sind *Open Source*. Entwicklungsumgebung ist *Netbeans* (Sun) gewählt.

Die Implementierung der Translation OWLS2WSDL wurde in folgende Hauptaufgaben aufgeteilt: Zur Basisfunktionalität gehören (1) das Parsen von OWL-DL und OWL-S, (2) die Erstellung des Projektmodells bestehend aus der Wissensbasis für Datentypen und Service-Kollektion, (3) die persitente Datenhaltung und (4) die Codegenerierung von XML Schema und WSDL anhand der Projektdaten (Abbildung 6.1 zeigt die Zusammenhänge).

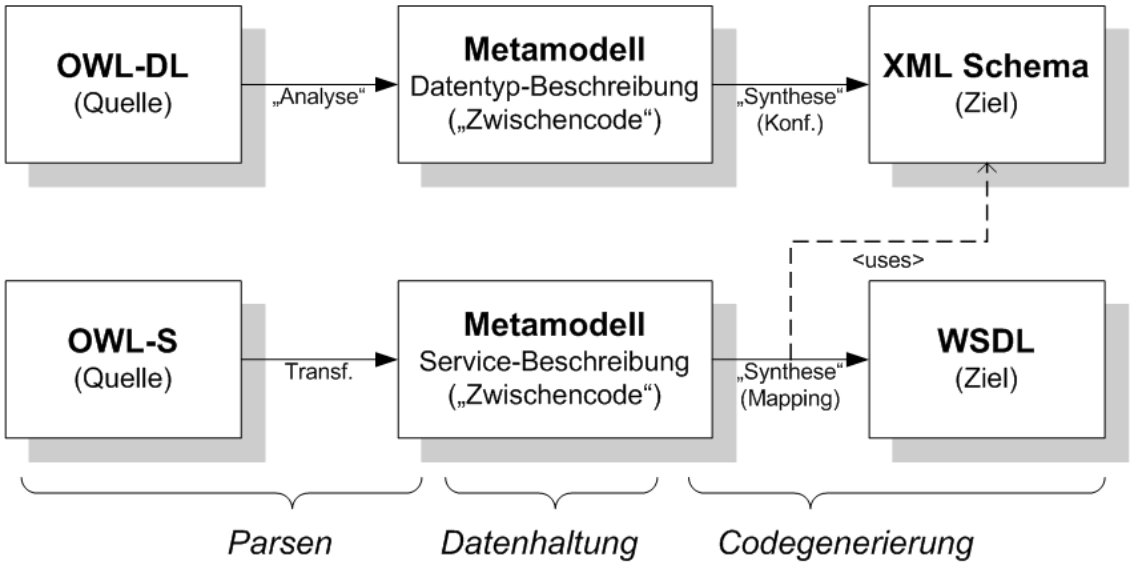


Abbildung 6.1: Übersicht Implementierung

6.1 Der OWL-S Parser

6.1.1 Aufgabenbeschreibung

Der OWL-S Parser muss die Signatur eines atomaren Prozesses innerhalb einer OWL-S Defintion finden und auslesen sowie allgemeine Informationen zu dem Service sammeln. Zu den allgemeinen Informationen gehören der Name des Service, Angaben zum Namespace und importierte Ontologien (Typbeschreibung des Parameters). Aufgenommen wird weiterhin die Position des Parameters innerhalb der Prozess-Beschreibung, um die gleiche Reihenfolge später in der WSDL Beschreibung innerhalb einer WSDL Operation einhalten zu können.

```
<process:Input rdf:ID="_CAR">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://127.0.0.1/ontology/my_ontology.owl#Car
</process:parameterType>
</process:Input>
```

Listing 6.1: OWL-S Parser. Beispiel einer OWL-S Prozess Signatur.

Das Listing zeigt den Input-Parameter `_CAR` vom Typ der OWL-Klasse `#Car`. Die OWL-Klasse wird eindeutig über ihren *Identifier* (URI) gekennzeichnet.

6.1.2 Technologien

Für den `ServiceParser` wurden zwei Implementierungen erzeugt. Anfangs wurde das OWL-S Modell der OWL-S API für diesen Zweck benutzt. Aufgrund der Tatsache, dass es für die OWL-S API je nach Version der OWL-S Spezifikation einen unterschiedlichen Release-Stream gibt, machte dieser Ansatz Probleme. Die API bildet zwar alle Spezifikationen ab, allerdings muss die API für jede OWL-S Version kompiliert werden und die bestimmte OWL-S Versionsklasse als Superklasse, die das OWL-S Modell abbildet, angegeben werden. Während der Laufzeit is so keine flexible Verarbeitung von OWL-S Definitionen mit unterschiedlicher OWL-S Version möglich. Aufgrund dieser Problematik wurde der `ServiceParser` mit Hilfe des Apache Xerces[1] `DOMParser` neu implementiert. Der Parser unterstützt die oben beschriebenen Anforderungen, bildet aber nicht das OWL-S Modell ab. Die OWL-S API beschreibt das OWL-S Modell und beinhaltet nützliche Tools wie eine *Execution Engine*, einen *Translator* für OWL-S 1.0 nach 1.1, einen OWL-S *Validator* und den *WSDL2OWL-S Converter*. Dieser wurde in die grafische Benutzeroberfläche des OWLS2WSDL Tools integriert.
Das mindswap[2] Projekt bietet die API bis zu der Version OWL-S 1.1.0-beta an. Seit August 2006 wird die API von den Entwicklern über GoogleCode[3] angeboten (MIT Lizenz). Die *Head Revision* der API (trunk, svn repository) unterstützt OWL-S 1.1 (aktuelle Version: 1.1.1).
Java packages: `org.apache.xerces.dom.*` und `org.apache.xerces.parsers.DOMParser`

[1] http://xerces.apache.org
[2] http://www.mindswap.org/2004/owl-s/
[3] http://code.google.com/p/owl-s/

6.2 Der OWL Parser

6.2.1 Technologien

Zur Verarbeitung von Ontologien in OWL wird das *Semantic Web Framework* Jena[4] eingesetzt. Benutzte Bestandteile des Frameworks sind die APIs für RDF, RDFS und OWL. Sie ermöglichen das Lesen und Schreiben von RDF/XML und bieten mehrere Möglichkeiten an, um auf das im Speicher aufgebaute Objektmodell `OntModel` zuzugreifen. Das Framework beinhaltet auch bereits eine *Inference Engine*, die neben dem *OWL-DL Reasoner* Pellet eingesetzt wird. Jena und Pellet[5] sind führende *Open Source* Technologien im Bereich *Semantic Web*.
Java packages: `com.hp.hpl.jena.*` und `org.mindswap.pellet.*`

6.2.2 Aufgabenbeschreibung

Die Klasse `DatatypeParser` realisiert den Parser für die OWL Definition (Ontologie). Er arbeitet mit dem `OntModel` von Jena. Zusätzlich wird ein temporäres Objektmodell eingeführt, um Objekte des genutzten Modells zusammenzufassen. Das neue Modell besteht aus den beiden Java-Klassen `OntClassContainer` und `OntClassKB`.

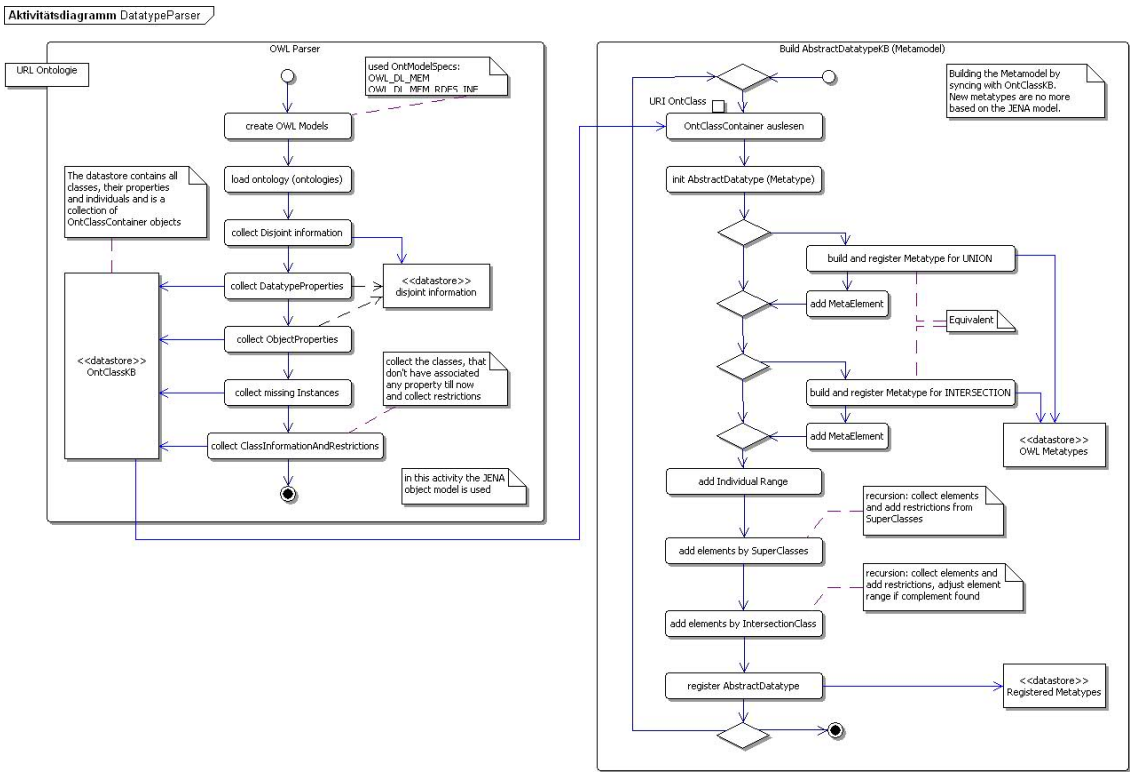


Abbildung 6.2: OWL Parser, Generierung der Wissensbasis

[4] http://jena.sourceforge.net/
[5] http://pellet.owldl.com

Idee: Ein Container sammelt alle Objekte des `OntModel`, die bei Abfragen mit Hilfe der Jena API gefunden werden (dazu zählen *Properties, Individuals, RDFS Comments, OWL Version*). Die implementierte Methode *parse()* arbeitet sequentiell folgende Daten der Ontologie ab:

1. Aufbau einer temporären Datenbank für `Disjoint` Informationen.
2. Einlesen aller Datentyp-Eigenschaften (`DatatypeProperty`). Für jede gefundene Eigenschaft wird ein `OntClassContainer` in der `OntClassKB` mit der *OWL Domain* (Objekt einer OWL-Klasse) angelegt und das `DatatypeProperty` Objekt zugeordnet.
3. Einlesen aller Objekt-Eigenschaften (`ObjectProperty`). Für jede gefundene Eigenschaft wird ein `OntClassContainer` in der `OntClassKB` mit der *OWL Domain* (Object einer OWL-Klasse) angelegt und das `ObjectProperty` Objekt zugeordnet.
4. Einlesen und Zuordnung von Instanzen (OWL Individuals) von bereits registrierten OWL-Klassen (`OntModel`).
5. Einlesen aller bisher noch nicht berücksichtigen OWL-Klassen und Restriktionen.

6.3 Datenhaltung (Aufbau der Wissensbasis)

6.3.1 Technologie

Zur Speicherung von Java-Klassen in XML wird das Castor *Framework* benutzt. Castor[6] ist ein *Open Source Data Binding Framework*. Die wichtigsten Funktionen sind das Binden von Java Objekten in ein XML-Dokument *(engl. XML binding, marshalling)* und die Speicherung von Java Objekten in einer relationalen Datenbank (Persistenz) über das Castor *JDO-Framework*. Der umgekehrte Prozess, aus XML-Dokumenten oder Datensätzen einer DB wieder Objekte zu erzeugen *(engl.: unmarshalling)*, wird ebenfalls unterstützt.

Für diese Arbeit relevante Bestandteile von Castor sind:

- XML Class Mapping API (Binden von Java Objekten in XML)
- XML Schema API (Implementierung des XML Schema Objektmodells)
- XML Code Generator (zum Erzeugen von XML Schema anhand von Java-Klassen)

6.3.2 Verarbeitung der Datentyp-Beschreibung

Um die aus einem Ontologie-Modell gesammelten Informationen persistent speichern zu können, müssen Objekte von Java-Klassen erzeugt werden, die von Castor mit Hilfe einer XML Mapping Definition in XML Dokumente gespeichert werden können.
Aus diesem Grund wird durch den Parsevorgang ein neues Objektmodell (Metamodell) aufgebaut, das abgespeichert werden kann. Die Klassen des neuen Modells basieren dabei nicht mehr

[6] http://www.castor.org

auf dem Jena *Framework*! Zentrale Java-Klasse des Objektmodells, die auch die Wissensbasis für die künftige Generierung von XML Schema darstellt, ist `AbstractDatatypeKB`. Die Klasse, die einen Datentyp in der Wissensbasis beschreibt ist `AbstractDatatype`. Synchronisiert wird die `AbstractDatatypeKB` mit der `OntClassKB` im Anschluss an den Parse-Vorgang ausgehend von der `OntClassKB`. Da Objekte der `OntClassKB` das Ontologie-Modell referenzieren, können während der Synchronisation weitere Abfragen gegen das Ontologie-Modell durchgeführt werden und auch Schlussfolgerungen getroffen werden, mit deren Hilfe die Wissensbasis aufgebaut wird. Ein Beispiel ist die Auswertung von Restriktionen.
Die Wissensbasis unterscheidet zwischen zwei Arten von Datentypen:

- Datentypen deren OWL Ursprungsklassen über einen *Identifier* referenziert werden
- Datentypen deren Ursprungsklassen anonym sind *(Intersection, Union, Complement)*

Die Klasse `AbstractDatatypeKB` wurde als Singleton implementiert, um den Zugriff auf Wissensbasis aus anderen Programmteilen zu erleichtern.

6.3.3 Verarbeitung der Service-Beschreibung

Der genutzte Parser für OWL-S Service-Definition nutzt die Klasse `AbstractService` und `AbstractServiceCollection`, um Daten abzulegen. Objekte beider Klassen können mit Hilfe von Castor direkt als XML-Dokumente gespeichert werden.

6.3.4 Die Klasse Projekt

Die Klasse `Project` umfasst die Wissensbasis und die Service-Kollektion. Die gemeinsame Datenbasis ermöglicht die Auflösung von Abhängigkeiten der Prozessparameter zu den benötigten Datentypen. Wie die Klassen `AbstractDatatypeKB` und `ServiceCollection` lässt sich das Projekt als XML Dokument abspeichern. Die Klasse ist als Singleton implementiert.

6.4 Der XML Schema Generator

6.4.1 Technologie

Die Castor XML API unterstützt die XML Schema (1.0 Second Edition, 2004) Empfehlung des W3Cs. Das XML Schema Objektmodell ist Basis für den Java Quellcode Generator und das Binden von Java-Objekten nach XML. Es eignet sich aber auch für den Aufbau und die Manipulation einer XML Schema Definition.
Java package: `org.exolab.castor.xml.schema`

6.4.2 Aufgabenstellung

Anhand einer Datentyp-Repräsentation aus der Wissensbasis soll ein XML Schema mit entsprechendem XML Schema Typ generiert werden, der dem ursprünglichen OWL Klassenkonzept

entspricht. Für jede Translation von OWL nach XSD wird eine `XSDSchemaGenerator` Instanz erzeugt und konfiguriert, die für Datentypen der Wissensbasis XML Schema Definitionen erzeugt. Die Konfiguration bezieht sich auf

- die Vererbungstiefe. Sie legt fest für welche Vererbungstiefe Eigenschaften von Oberklassen übernommen werden sollen.
- die Auswertung der semi-automatisch festgelegten Typinformationen bei Vererbung.
- den Gebrauch des *Hierarchy Pattern*.

Die persistente Datenhaltung beschleunigt die Translation, da die benötigten Ontologien nicht mehr geparst werden müssen. Der Codegenerator arbeitet mit dem erstellten Zwischencode (Projekt). Die Methoden *toXSD()* und *appendToSchema()* erstellen anhand der Datentypen aus der `AbstractDatatypeKB` Instanz (Wissensbasis) das XML Schema. Die Referenz auf den Datentyp aus der Wissensbasis wird den Methoden als Parameter mitgegeben. Innerhalb des Schema Generators wird die Methode *addComplexType()* rekursiv aufgerufen.

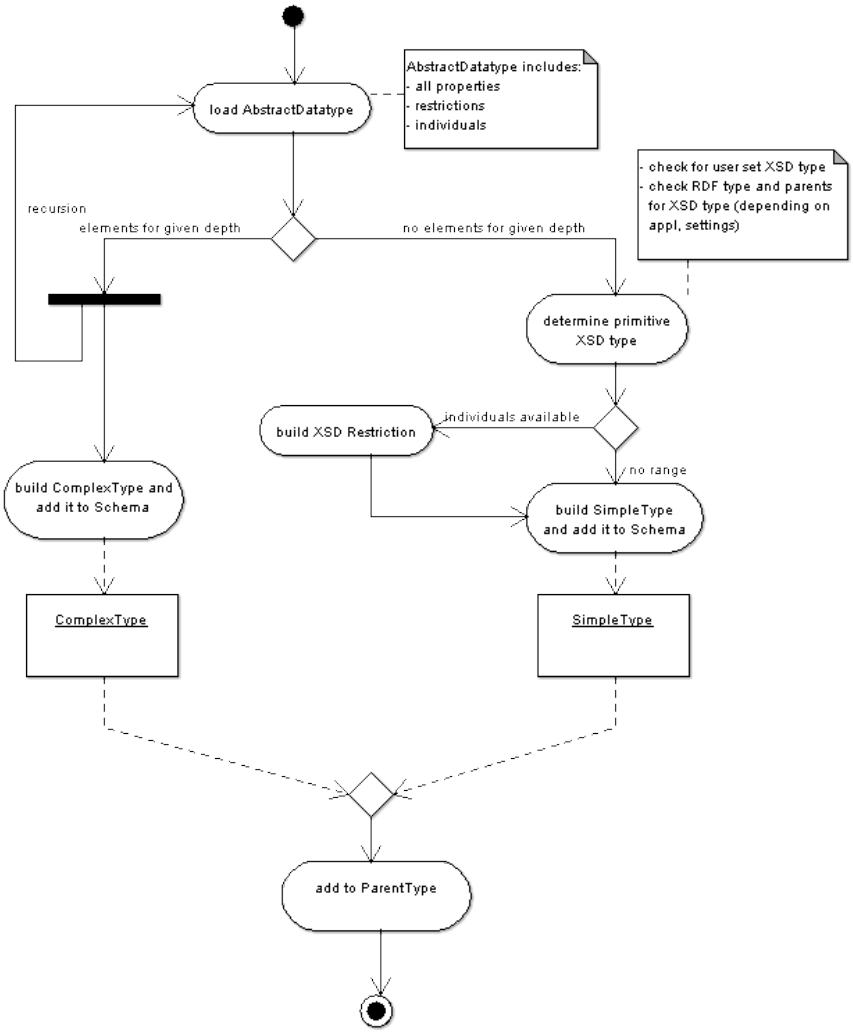


Abbildung 6.3: OWL2XSD. Schema Generator.

6.5 Der WSDL Builder

6.5.1 Technologie

Die WSDL Beschreibungen werden mit dem *WSDL for Java Toolkit* (WSDL4J[7]) aufgebaut. Das Objektmodell von WSDL4J erlaubt den Aufbau und die Manipulation von WSDL Dokumenten. Die Implementierung zusammen mit der Dokumentation bildet den *Java Specification Request 110* (JSR-110) und wird im Rahmen des *Java Community Process*[8] entwickelt.
Java packages: `javax.wsdl.*` und `javax.wsdl.extensions.*`

6.5.2 Aufgabenbeschreibung

Die implementierte Methode *buildDefinition()* der Klasse `WSDLBuilder` erstellt anhand der Parameter `AbstractService` und `XsdSchemaGenerator` ein Objekt, das die WSDL Beschreibung repräsentiert. Mit dem `WSDLWriter` (API) wird das WSDL File erzeugt.

6.6 Das Frontend (GUI)

Die grafische Benutzeroberfläche dient der Darstellung der im Projekt enthaltenen Datentyp- und Service-Information und ermöglicht die manuelle Änderung bzw. Erweiterung der Projektdaten.
Java packages: `javax.swing.*`, `java.awt.*` und `com.jgoodies.*`

Funktionen der Oberfläche sind:

- die semi-automatische Zuordnung von primitiven Datentypen.
- die Konfiguration des XML Schema Generators.
- eine Batchfunktion um mehrere WSDL Beschreibungen auf einmal zu generieren.
- die Integration des *WSDL2OWL-S Converters* (Re-engineering).
- die Markierung von nicht übersetzbaren Datentypen.
- die Anzeige von Fehlern des Parse-Vorgangs (gespeichert in Wissensbasis).

Um den *WSDL2OWL-S Converter* innerhalb der GUI einsetzen zu können, mussten Anpassungen in Teilen der OWL-S API durchgeführt werden. Die Anpassungen betreffen die Oberfläche und die automatische Übernahme der Mapping-Information aus der Service-Kollektion und der Wissensbasis für Datentypen.

6.6.1 Programmablauf (GUI)

Abbildung 6.4 Seite 58

[7] `http://sourceforge.net/projects/wsdl4j`
[8] `http://www.jcp.org/jsr/detail/110.jsp`

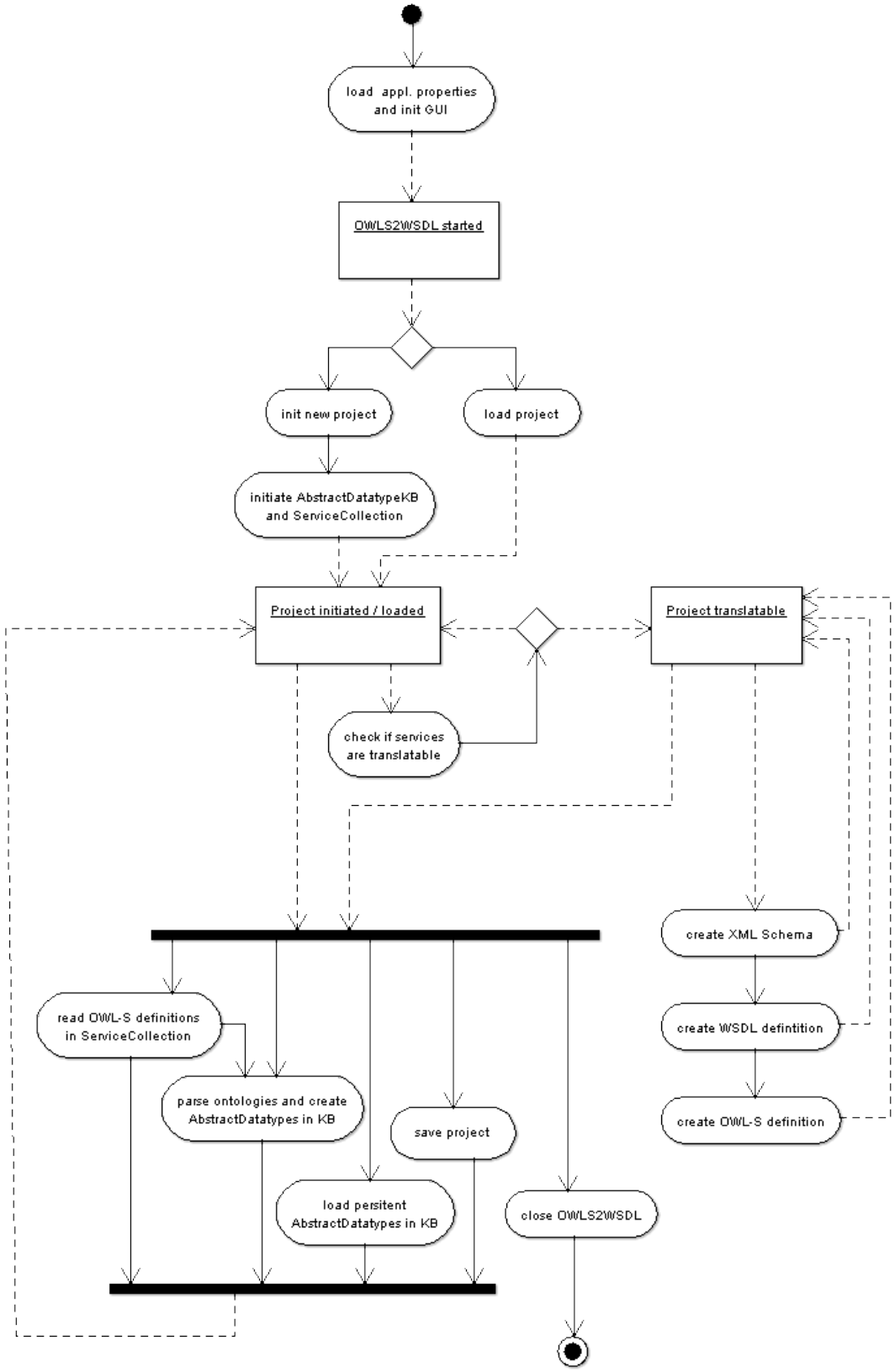


Abbildung 6.4: OWLS2WSDL GUI Application Flow

6.7 Paket Übersicht

Java package: `de.dfki.dmas.owls2wsdl.*`

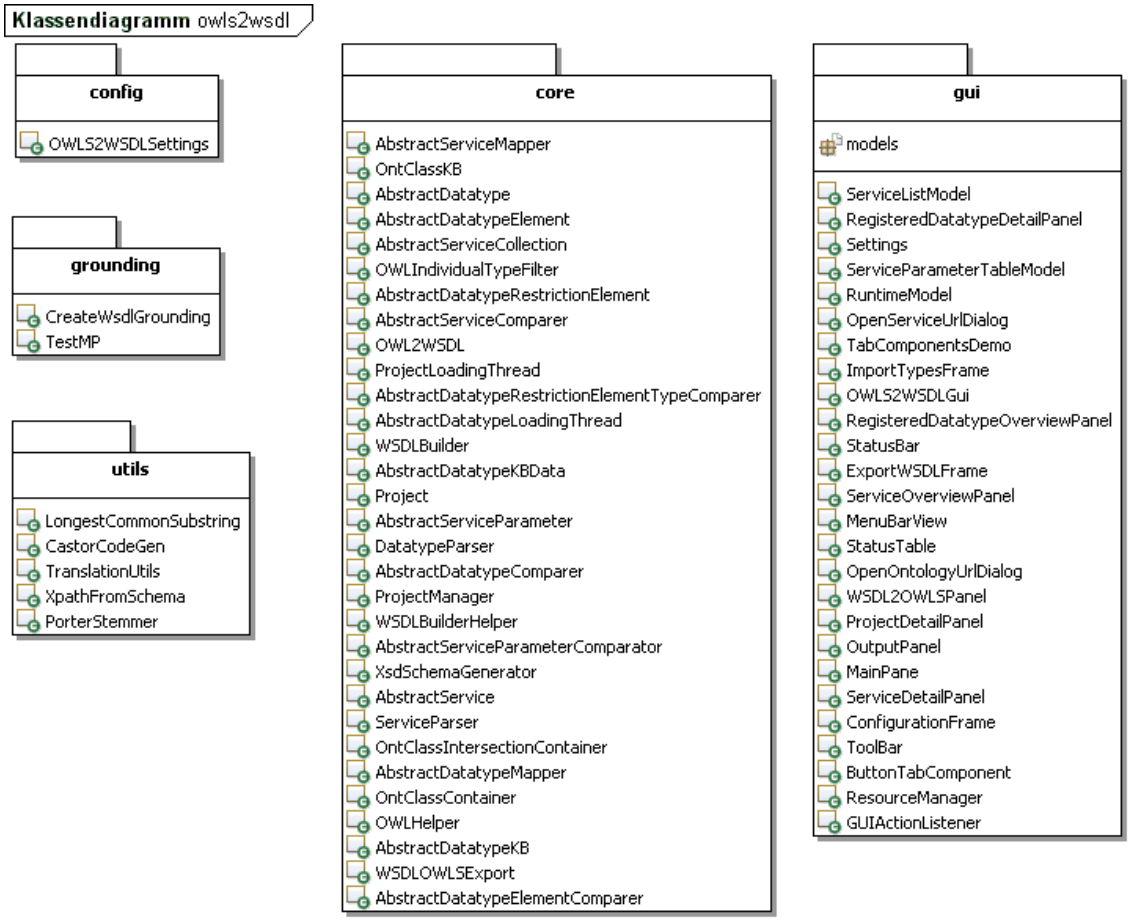


Abbildung 6.5: Implementierung. owls2wsdl packages.

Bemerkung:
In einer künftigen Version des Tools sollte **core** in folgende Pakete aufgesplittet werden:

- parser
- generator
- persistent

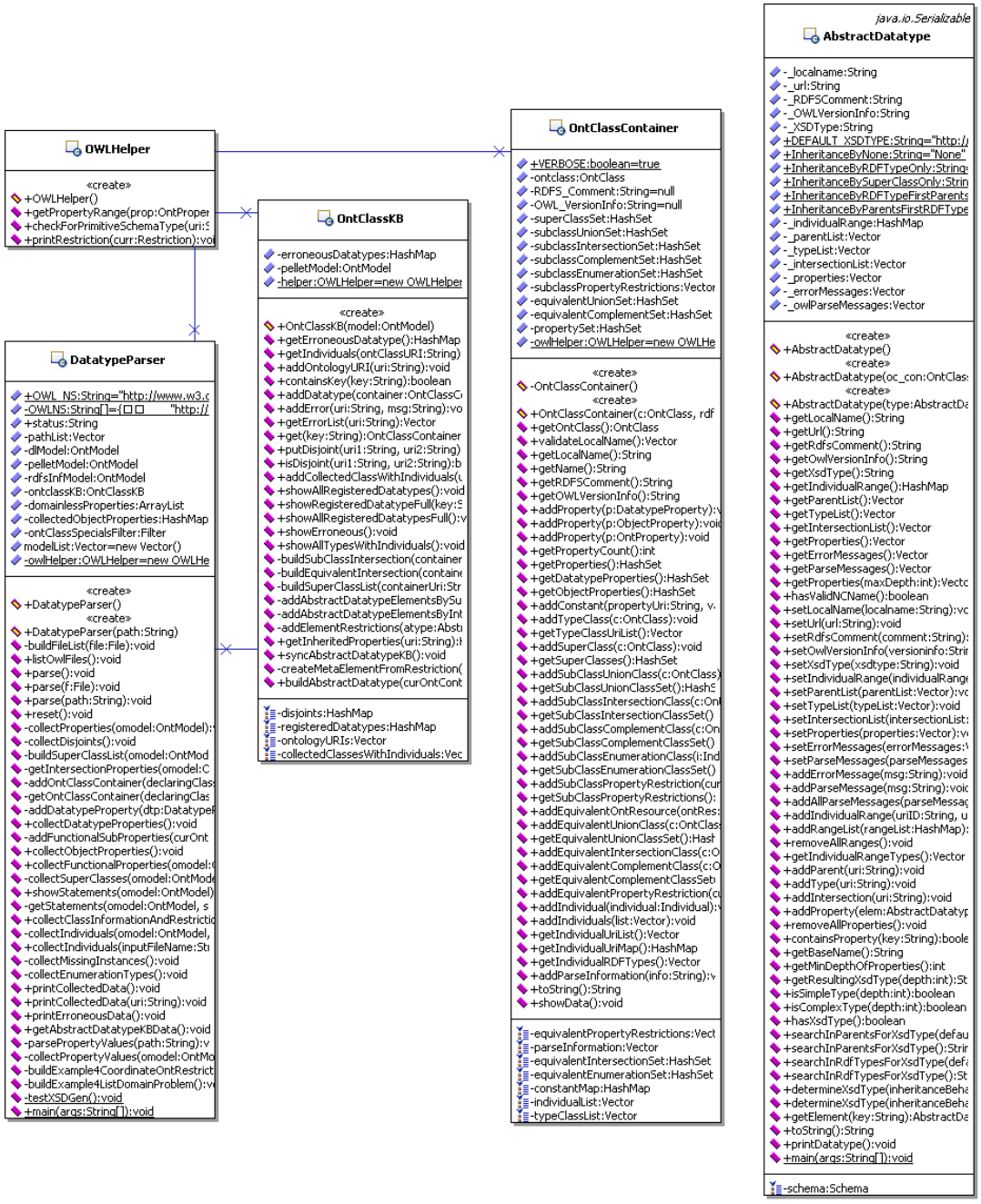


Abbildung 6.6: Implementierung. OWL Parser.

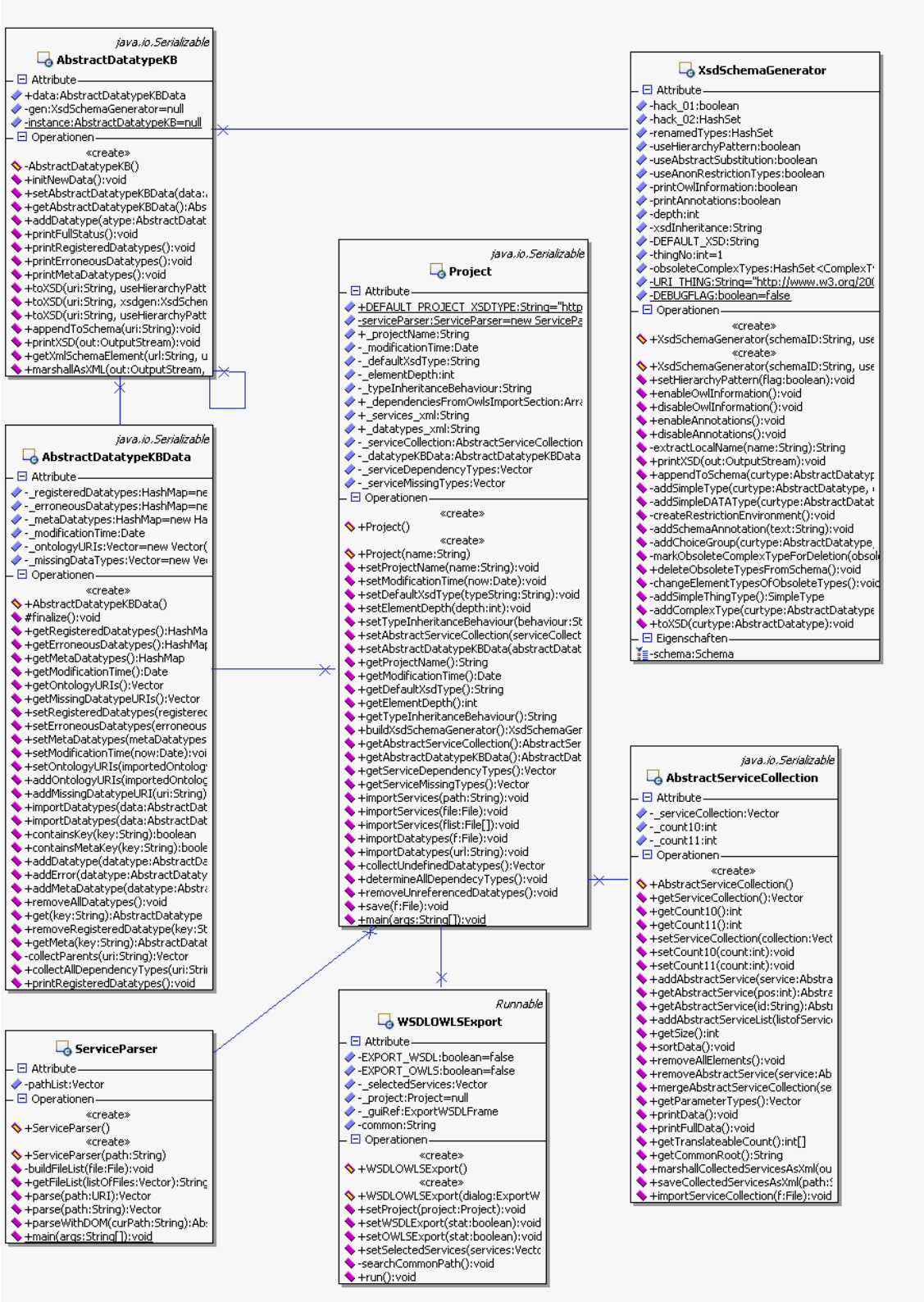


Abbildung 6.7: Implementierung. Projektverarbeitung und XML Schema Generator.

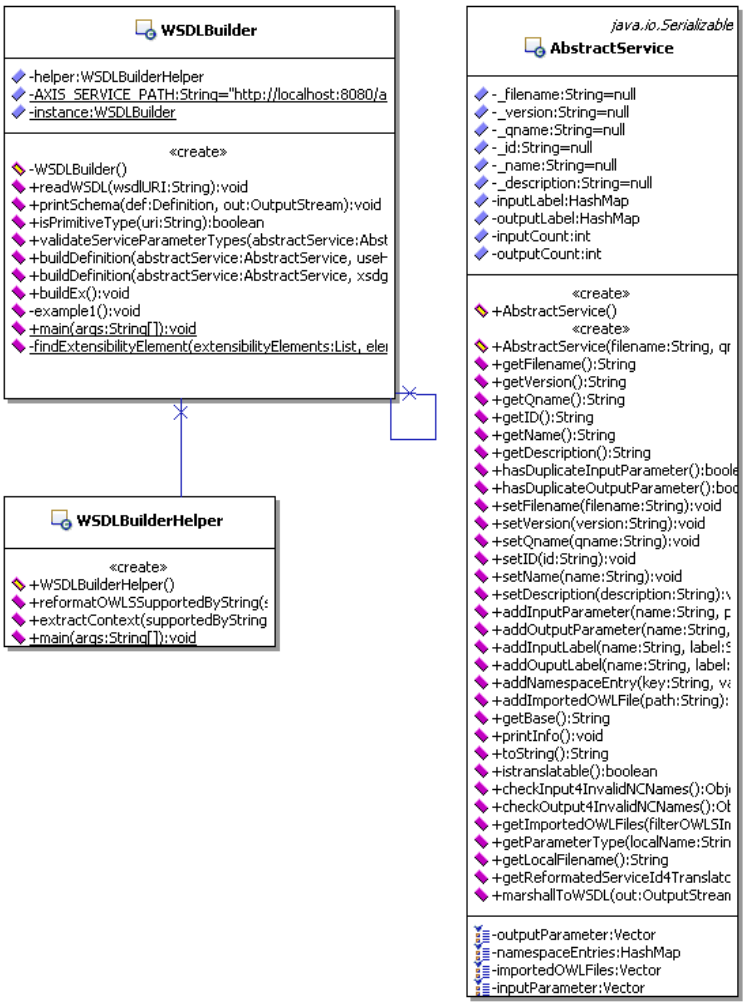


Abbildung 6.8: Implementierung. WSDL Builder.

6.8 Eingesetzte Technologien (Übersicht)

Die Anwendung besitzt folgende Architektur:

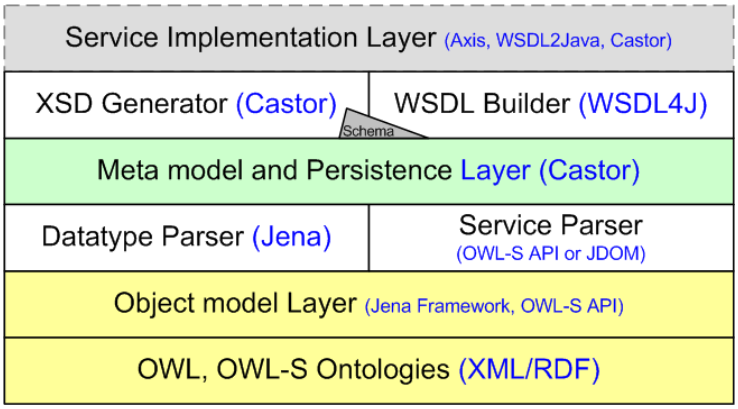


Abbildung 6.9: OWLS2WSDL. Architektur.

Die Abbildung 5.3 auf Seite 40 kann nun um die eingesetzten Technologien erweitert werden:

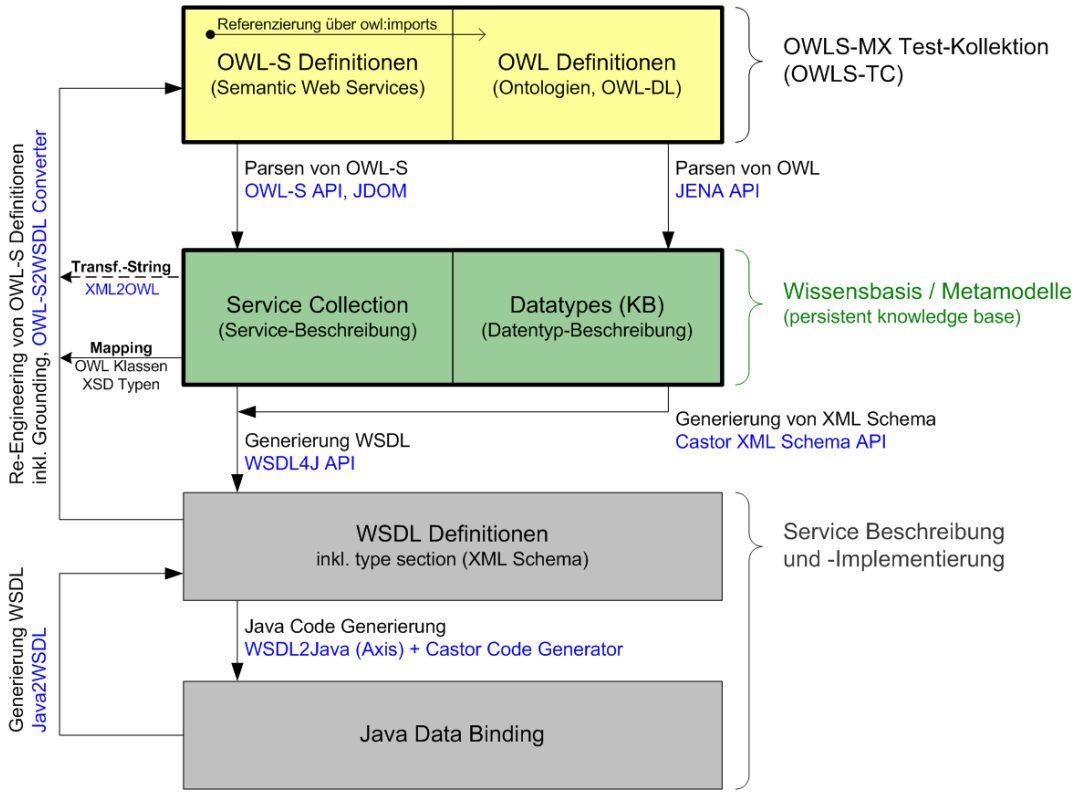


Abbildung 6.10: OWLS2WSDL. Eingesetzte Technologien.

6.9 Future Work

6.9.1 Erweiterung des OWL Parsers

In der aktuellen Version der Implementierung werden bereits Fehlermeldungen, die beim Parsen von OWL-Konzepten auftreten (Java Exceptions des Jena Frameworks) als Informationen zu einem Datentyp der Wissensbasis abgelegt und über die grafische Oberfläche angezeigt. Fehler können in einer Ontologie im Rahmen der Klassen-Beschreibung auftreten oder beispielsweise durch Konflikte bei der Referenzierung von Klassen über den *Namespace* entstehen. Eine zusätzliche Unterstützung bietet hier die Jena Erweiterung Eyeball[9], die das OWL Ontologie-Modell (XML/RDF) auf Fehler bzw. Korrektheit überprüft.

6.9.2 Erweiterung des OWL-S Parsers

Der OWL-S Parser liest in der aktuellen Implementierung die Signatur eines AtomicProcess. Für die Übersetzung der OWLS-TC ist dies ausreichend. Will man küntig Schnittstellen eines CompositeProcess bzw. SimpleProcess (A.7) auslesen, muss der Parser erweitert werden.

6.9.3 Erweiterungen des WSDL Builders

Variation des Zielformats

Das implementierte Tool nutzt das WSDL4J Objektmodell, um WSDL Beschreibungen zu generieren. Durch Änderung des Zielformats und des von dem WSDL Builder genutzten Objektmodells lassen sich auch weitere Beschreibungsformate generieren. Aufgrund der Notwendigkeit, semantische Annotationen in WSDL einzufügen, sollten für künftige Versionen der implementierten Translation Technologien wie WSDL-S[10] und SAWSDL4J[11][12] betrachtet werden.

Automatische Konfiguration

Ein Ziel der Arbeit ist es, WSDL Beschreibungen zu generieren, die bei dem Matchmaking mit dem *WSDL Analyzer* möglichst gute Vergleichswerte zu den *OWLS-MX* Werten liefern. Die Evaluation (Kapitel 8) der implementierten Translation zeigt die Bedeutung der Konfiguration des XML Schema Generators. Gerade die Einstellung der Vererbungstiefe hat direkte Auswirkungen darauf, ob ein `SimpleType` oder ein `ComplexType` erstellt wird, und für einen `ComplexType`, wieviele Elemente von der Superklasse (bzw. ParentType) gererbt werden. Man kann keinen pauschalen Wert für die Vererbungstiefe festlegen, der ein gutes Matchmaking-Verhalten der generierten WSDL Beschreibungen bewirkt. Die Integration des *WSDL Analyzers* in das OWLS2WSDL Tool könnte hier eine automatische Konfiguration ermöglichen, indem Konfigurationen für eine Translation ausprobiert werden und die Konfiguration, die zu dem besten Matchmaking-Ergebnis für eine festgelegte Kandidaten-Menge führt, übernommen wird.

[9] http://jena.sourceforge.net/Eyeball/
[10] http://www.w3.org/Submission/WSDL-S/
[11] http://www.w3.org/2002/ws/sawsdl/
[12] http://knoesis.wright.edu/opensource/sawsdl4j/

6.9.4 Re-Engineering

Generierung der Transformation (XSL, Grounding)

In die Anwendung wurde das *Re-Engineering* für WSDL Beschreibungen nach OWL-S integriert. Genutzt wird die Mapping-Information der Wissensbasis, die auch für die Generierung der WSDL Beschreibungen genutzt wird. Durch Einsatz des *WSDL2OWL-S Converters* kann so eine OWL-S Definition (Version 1.1) generiert werden, die ein WSDL-Grounding der Prozess-Signatur enthält. In dem WSDL Grounding fehlt jedoch der `xsltTransformationString` (XSL), der bei Aufruf eines WSDL Dienstes (über die OWL-S *Execution Engine*) die XML-Instanz (SOAP-Nachricht) in ein OWL Individual (RDF Graph) transformiert.

Neben dem Aufwand, der investiert werden muss, um eine Methode zu implementieren, die anhand der Wissensbasis den XSLT String generiert, wurde JXML2OWL[13] evaluiert. JXML2OWL ist eine Bibliothek inklusive einer GUI, die den Anwender dabei unterstützt Mappings zwischen XML Schema Definitionen und OWL Definitionen zu erstellen. Mit der Mapping-Information könnte eine XSLT Transformation generiert werden, die es ermöglicht, XML-Instanzen zu OWL-Instanzen zu transformieren. Das Mapping verknüpft OWL Klassennamen sowie die Namen der Eigenschaften mit dem jeweiligen XPath aus der XML Schema Definition.

Bewertung der JXML2OWL Technologie:
Die Generierung der XSL Transformationsregel ist für umfangreichere Typen **nicht trivial**.
Ansatz ist, für alle über einen *Dump* gefundenen **XPath Angaben** in JXML2OWL Mappings zu konfigurieren. Probleme machen bei dieser Vorgehensweise definierte Zyklen im Objektmodell bzw. in der XML Schema Definition. Um *Dumps* zu erstellen, müssen für das Auslesen einer XML Schema Definition mit Zyklen Abbruchbedingungen implementiert werden. Für einige umfangreichere Schema Definitionen konnte bislang keine terminierender *Dump Methode* implementiert werden. Trotzdem konnten für "einfachere XML Schema Definitionen" trotz Zyklen *Dumps* erstellt werden. Eine künftige Aufgabenstellung ist die Untersuchung und das Dumpverhalten von XML Schema Definitionen mit Zyklen.
Eine weitere Einschränkung ist, dass die genutzte Java *XSLT Engine* der OWL-S API das von JXML2OWL generierte XSLT Script aufgrund der genutzten XSLT Spezifikation nicht interpretieren kann. Die unterstützten XSLT Spezifikationen unterscheiden sich! Ein zum Sprachumfang gehörendes Element der aktuellen XSLT Spezifikation, das beispielsweise nicht verarbeitet werden kann, ist `xx:node-set` .
Ein künftiges Arbeitspaket ist die Erweiterung der *ProcessExecutionEngine* (OWL-S API), um auch neue Versionen der XSLT Spezifikation einsetzen zu können.

[13] http://jxml2owl.projects.semwebcentral.org

Translation zwischen OWL-S 1.0 und 1.1

Die Testkollektion OWLS-TC beinhaltet OWL-S Service Definitionen im 1.0 und im 1.1 Format. Mit der aktuellen OWL-S API (Objektmodell) können nur OWL-S 1.1 Definitionen generiert werden. Mit Hilfe der API lassen sich beide OWL-S Versionen lesen, da ein Übersetzer für Version 1.0 nach 1.1 eingebaut ist. Um künftig auch auch ohne Austausch der OWL-S API Version OWL-S Definitionen im 1.0 Format rekonstruieren zu können *(Re-Engineering)*, wäre ein Übersetzer für Version 1.1 nach 1.0 nötig *(voluntarily contributions)*.

Validierung von OWL-S Services

Der OWL-S Validierer `org.mindswap.owls.validator` basiert auf der OWL-S API Version 1.0 und ist momentan als *broken* deklariert *(voluntarily contributions)*.

6.9.5 Integration von WSDL2Java

Hilfreich für eine Vorgehensweise nach dem vorgestellen *top down* Ansatzes (Abschnitt 2.2.7) ist die Integration des WSDL2Java Tools. In diesem Zusammenhang sollte das erstellte Objektmodell für die Service-Beschreibung innerhalb der Service-Kollektion um die Service-Adresse und um *Namespace* Angaben erweitert werden.

Kapitel 7

Benutzerhandbuch

Dieses Kapitel erläutert die Verwendung des implementierten OWLS2WSDL Tools. Dabei richtet es sich an verschiedene Benutzergruppen. Das Tool kann eingesetzt werden, um einzelne WSDL Beschreibungen anhand von OWL-S Definitionen zu generieren. Dieser top down *Ansatz wird bei der Service Implementierung innerhalb einer SOA gewählt (MDA). Aus der generierten WSDL Beschreibung kann mittels* WSDL2Java *der Basis-Quellcode für eine Implementierung generiert werden. Die Generierung von einzelnen XML Schema Definitionen ist auch möglich. Weitere Verwendung des Tools innerhalb dieser Arbeit ist die Übersetzung von Testszenarien der OWLS-TC Testkollektion. Hier müssen mehrere Service-Beschreibungen auf einmal übersetzt werden, die dann mit dem* WSDL Analyzer *untersucht werden sollen.*

7.1 Download und Installation

Projektseite des *OWLS2WSDL Tools*:
`http://code.google.com/p/owls2wsdl/`

Voraussetzungen:

- Java Version 1.5

Die grafische Oberfläche wird mit `java -jar OWLS2WSDL.jar` gestartet. Das Tool bietet zusätzlich ein *command line interface* (CLI). Den Gebrauch zeigt Abbildung 7.1.

```
java -jar OWLS2WSDL.jar -help

 usage: owls2wsdl [-help] [FILE.owl|URL.owl|FILE.xml|URL.xml]

 FILE.owl      parse an OWL file and generate a persistent knowledgebase
 FILE.xml      generate XML Schema

 without arguments, start GUI
```

Listing 7.1: OWL2WSDL. Command line interface.

7.2 Erstellung der Wissensbasis

Die Wissensbasis enthält nach dem Parsen einer Ontologiebeschreibung (OWL-DL) **alle Klassen** der Ontologie mit gefundenen Eigenschaften (*brute force* Ansatz, um sicherstellen zu können, dass alle Abhängigkeiten einer Service Schnittstelle aufgelöst werden). OWL-Klassen werden, sobald sie in die Wissensbasis eingelesen wurden, als Metatypen oder Datentypen bezeichnet. Ihre Eigenschaften werden als Elemente zu einem Datentyp angzeigt. Der Elementname entspricht dem Namen der Eigenschaft, der Elementtyp kann entweder ein primitiver Datentyp (nach XML Schema Spezifikation), ein `SimpleType` oder `ComplexType` sein. Durch das Parsen einer Ontologie (OWL) werden auch refernzierte OWL Konzepte einer anderen Ontologie Beschreibung eingelesen.

7.2.1 Aufbau der Wissensbasis (GUI)

Ontologien (OWL Dateien) lassen sich über die Funktion **Parse ontology** (siehe Abb. 7.5) aus dem Dateisystem lesen oder über die Angabe der URL laden. Die Inhalte der Wissensbasis werden die grafische Benutzeroberfläche verwaltet.

7.2.2 Aufbau der Wissensbasis (CLI)

Da das Parsen durch Interpretation der Ontologie (Abbildung 5.1) je nach Ontologie (OWL) sehr zeitaufwändig ist, wurde das Tool mit einer Schnittstelle zur Kommandozeile versehen. Das Parsen der beiden Ontologien Mid-level-ontology.owl und SUMO.owl (Übersicht C) dauert beispielsweise etwas länger. Das Tool schreibt alle Datentypen in die Wissenbasis (KB), die am Ende als XML Dokument gespeichert wird. Die Datentyp Informationen aus diesem Dokument können bei einer späteren Projektbearbeitung importiert werden.

```
usage: owls2wsdl [options] FILE.owl
 -help           print help message
 -kbdir <dir>    knowledgebase directory; necessary
 -test           parse only, don't save
```

Listing 7.2: OWL Parser. Help message (CLI)

Da der Gebrauch der Jena API sehr speicherintensiv ist, muss beim Starten des Programms darauf geachtet werden, dass genug Arbeitsspeicher für die Java Umgebung zur Verfügung steht: Beispiel: `java -Xms256m -Xmx512m -jar OWLS2WSDL.jar -kbdir . SUMO.owl`

7.3 Generierung von XML Schema (OWL2XSD)

7.3.1 Konfiguration des XML Schema Generators

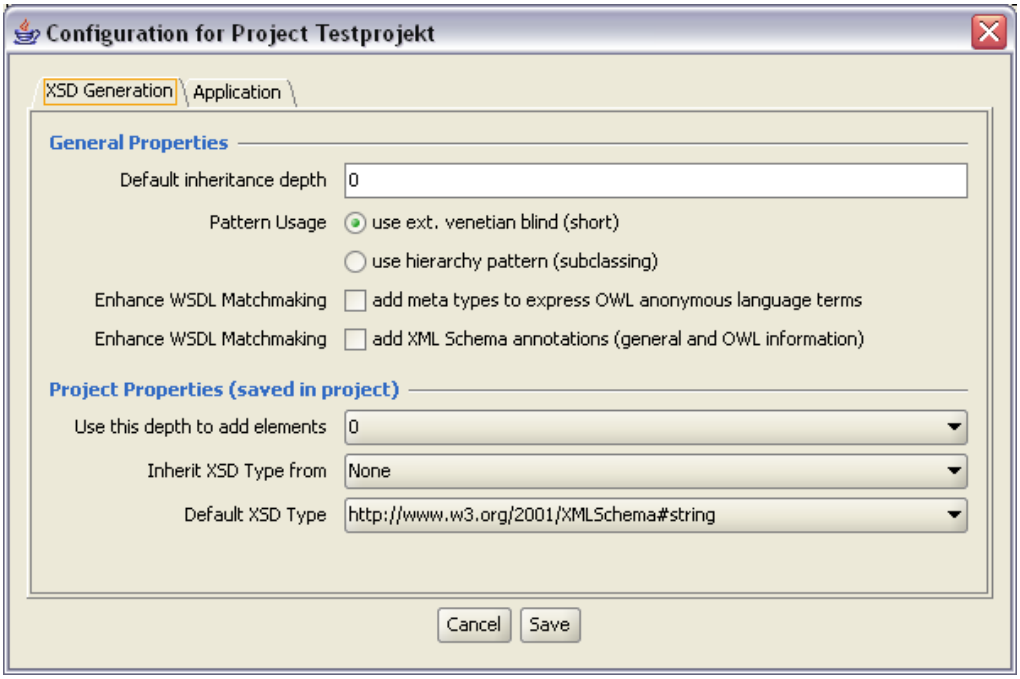


Abbildung 7.1: OWLS2WSDL Tool. Konfiguration XML Schema Generator.

Default inheritance depth legt einen Standardwert für die Vererbungstiefe fest, mit der Elemente zu einer Typbeschreibung aufgenommen werden sollen.

Pattern usage Auswahl des Entwurfsmusters. Bei Auswahl des *erweiterten Venetian Blind Pattern* wird für jeden Parameter einer WSDL Message ein XML Schema Element mit kompletter Typbeschreibung angelegt. Bei Auswahl des *Hierarchy Pattern* wird zusätzlich die OWL Vererbungshierarchie in der XML Schema Definition abgebildet.

Enhance matchmaking Die erste Option integriert eine Interpretation der anonymen Klassenkonzepte (z.B. *intersection, union*). Die zweite Interpretation erstellt Schema Annotationen anhand OWL Informationen der Wissensbasis (z.B. *isFunktional, isTransitive*).

Inherit XSD Type from Zur Unterstützung der semi-automatischen Angabe von primitiven Datentypen, kann mit dieser Option festgelegt werden, ob bereits festgelegte Datentypen aus Oberklassen geerbt werden sollen. Außerdem lässt sich festlegen, ob der RDF-Type, falls er von dem Standardwert `#Ressource` abweicht, übernommen werden soll.

Default XSD Type Für alle Typen, für die kein primitiver Typ gefunden wurde und kein Typ manuell gesetzt wurde, wird dieser Standardtyp verwendet.

7.3.2 Konfiguration der Vererbungstiefe (Beispiel 1)

Voraussetzung: Als Voraussetzung, um XML Schema Definitionen generieren zu können, muss ein Projekt angelegt sein, das die Datentypen aus einer Wissensbasis importiert hat bzw. selbst bereits Datentypen aus eine Ontologie gelesen hat. Ein einfaches Beispiel, das die Bedeutung der Konfiguration der Vererbungstiefe zeigt, ist die Generierung eines Schema Typs für den Datentyp RecommendedPriceInEuro aus der concept.owl Ontologie.

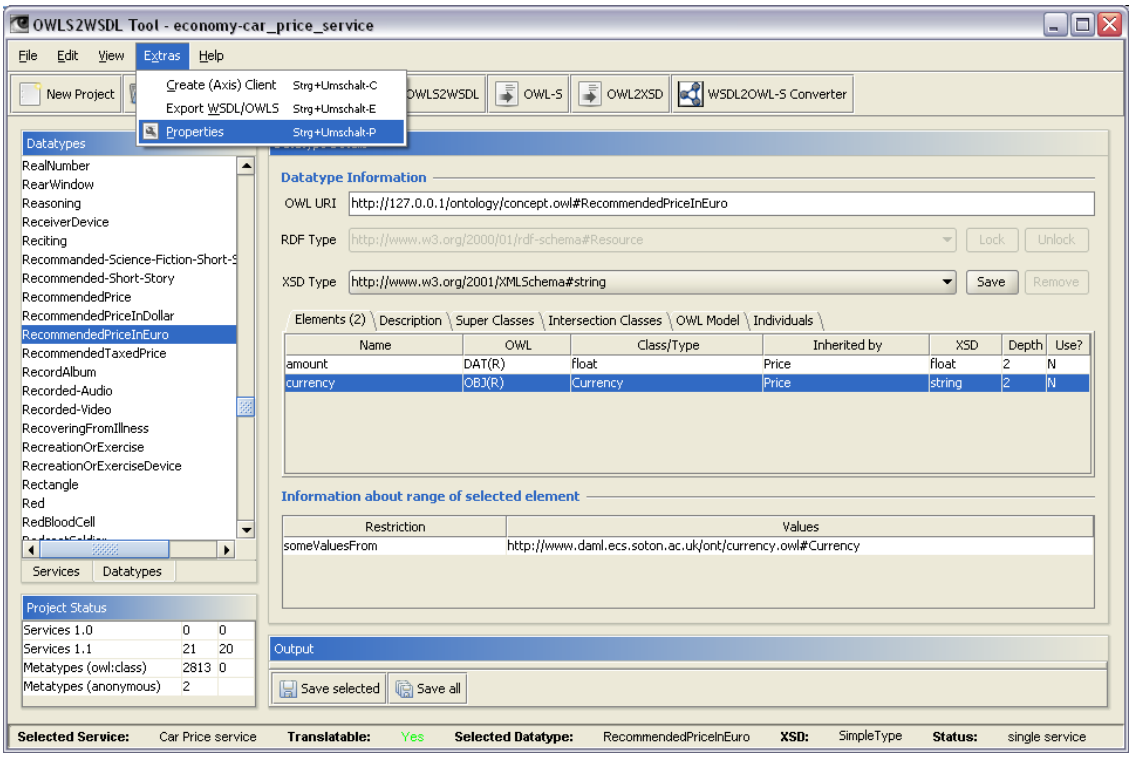


Abbildung 7.2: http://127.0.0.1/ontology/concept.owl#RecommendedPriceInEuro

Die Datentypen der Wissensbasis werden auf der linken Seite in einer Liste angezeigt. Sobald man einen Datentyp auswählt, zeigt das Panel **Datatype Details** Informationen zu dem Typ an.

OWL URI URI der OWL Ursprungsklasse

RDF TYPE Eintrag `type` in der OWL Klassendefinition (selten)

XSD TYPE Primitiver Datentyp der gewählt wird, wenn keine Elemente zur Verfügung stehen, um einen `ComplexType` zu bauen.

Element TabbedPane Hier werden die Sub-Elemente eines Datentyps gelistet. Weiterhin findet man an dieser Stelle die OWL Beschreibung, eine Liste der OWL Superklassen und Intersectionklassen sowie OWL Individuals der OWL Klasse.

Range Die Tabelle listet alle Element-Restriktionen für ein ausgewähltes Element auf.

Abbildung 7.2 zeigt, dass der Datentyp für RecommendedPriceInEuro ab Vererbungstiefe 2 aus den zwei Elementen amount und currency besteht. Beide wurden durch Auswertung einer Restriktion der Superklasse Price gefunden. amount ist in OWL ein DatatypeProperty mit der Typinformation `float` und currency ein ObjectProperty mit dem *Range* Currency. Currency hat innerhalb der Wissensbasis für die konfigurierte Vererbungstiefe keine weiteren Sub-Elemente und wird deshalb als `SimpleType` mit primitiven Basistypen `string` interpretiert. Der primitive Basistyp für die Klasse Currency kann manuell gesetzt werden.

Die Generierung der XML Schema Definition wird mit den Button **OWL2XSD** gestartet. Das Schema wird in dem **Output** Panel angezeigt. Für jede Konfiguration kann ein neues XML Schema generiert werden, das den Datentyp unterschiedlich interpretiert. Der Button **save selected** ermöglicht das Speichern der generierten Schema Definition.

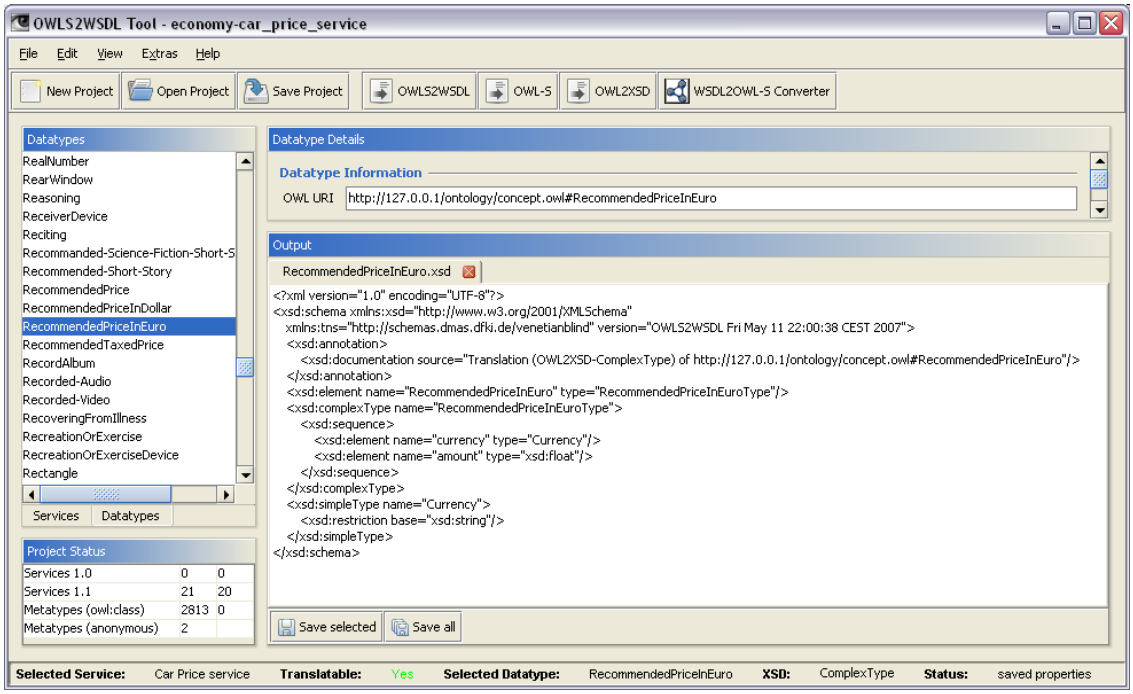


Abbildung 7.3: http://127.0.0.1/ontology/concept.owl#RecommendedPriceInEuro (Schema)

Generierter XML Schema Code je nach Konfiguration

Abhängig von der konfigurierten Vererbungstiefe, werden OWL Klassen unterschiedlich interpretiert. Beispiel: RecommendedPriceInEuro

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.dmas.dfki.de/venetianblind"
    version="OWLS2WSDL Thu May 10 14:00:39 CEST 2007">
    <xsd:annotation>
        <xsd:documentation source="Translation (OWL2XSD-SimpleType) of
            http://127.0.0.1/ontology/concept.owl#RecommendedPriceInEuro"/>
    </xsd:annotation>
    <xsd:element name="RecommendedPriceInEuro" type="RecommendedPriceInEuroType"/>
    <xsd:simpleType name="RecommendedPriceInEuroType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
</xsd:schema>
```

Listing 7.3: Beispiel RecommendedPriceInEuro, Konfiguration 1: Tiefe 0 (d0)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.dmas.dfki.de/venetianblind"
    version="OWLS2WSDL Thu May 10 14:07:08 CEST 2007">
    <xsd:annotation>
        <xsd:documentation source="Translation (OWL2XSD-ComplexType) of
            http://127.0.0.1/ontology/concept.owl#RecommendedPriceInEuro"/>
    </xsd:annotation>
    <xsd:element name="RecommendedPriceInEuro" type="RecommendedPriceInEuroType"/>
    <xsd:complexType name="RecommendedPriceInEuroType">
        <xsd:sequence>
            <xsd:element name="currency" type="Currency"/>
            <xsd:element name="amount" type="xsd:float"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="Currency">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
</xsd:schema>
```

Listing 7.4: Beispiel RecommendedPriceInEuro, Konfiguration 2: Tiefe 2 (d2)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:tns="http://schemas.dmas.dfki.de/venetianblind"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    version="OWLS2WSDL Thu May 10 17:00:54 CEST 2007">
  <xsd:annotation>
    <xsd:documentation source="Translation (OWL2XSD-ComplexType) of
        http://127.0.0.1/ontology/concept.owl#RecommendedPriceInEuro"/>
  </xsd:annotation>
  <xsd:element name="RecommendedPriceInEuro" type="RecommendedPriceInEuroType"
               substitutionGroup="RecommendedPrice"/>
  <xsd:element name="RecommendedPrice" type="RecommendedPriceType" abstract="true"
               substitutionGroup="Price"/>
  <xsd:element name="Price" type="PriceType" abstract="true"
               substitutionGroup="UntangibleObjects"/>
  <xsd:element name="UntangibleObjects" type="UntangibleObjects" abstract="true"/>
  <xsd:complexType name="PriceType">
    <xsd:sequence>
      <xsd:element name="currency" type="Currency"/>
      <xsd:element name="amount" type="xsd:float"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="RecommendedPriceType">
    <xsd:complexContent>
      <xsd:restriction base="PriceType">
        <xsd:sequence>
          <xsd:element name="currency" type="Currency"/>
          <xsd:element name="amount" type="xsd:float"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="RecommendedPriceInEuroType">
    <xsd:complexContent>
      <xsd:restriction base="RecommendedPriceType">
        <xsd:sequence>
          <xsd:element name="currency" type="Currency"/>
          <xsd:element name="amount" type="xsd:float"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType name="Currency">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="UntangibleObjects">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:schema>
```

Listing 7.5: Beispiel RecommendedPriceInEuro, Konfiguration 3: Tiefe 2 + Hierarchy (d2h)

Das Listing 7.3 zeigt einen generierten `SimpleType` mit der Basis `xsd:string`. Das Listing 7.4 zeigt den entsprechenden `ComplexType` bei Auswertung mit Ableitungstiefe 2, und Listing 7.5 ergänzt das Schema um das vorgestellte *Hierarchy Pattern*.

7.3.3 Generierung von XML Schema über das CLI

Das *command line interface* bietet die selben Funktionen an wie die grafische Benutzeroberfläche.

```
usage: owls2wsdl [options] [FILE.xml|URL]
 -d,--depth            set recursion depth
 -h,--hierarchy        use hierarchy pattern
 -help                 print help message
 -info                 print datatype information
 -keys                 list all owlclass keys
 -owlclass <class>     owl class to translate; necessary
 -p,--primitive        set default primitive type
 -xsd                  generate XML Schema
```

Listing 7.6: XML Schema Generator. Help message (CLI)

7.3.4 Erstellung eines komplexen XML Schema Typs (Beispiel 2)

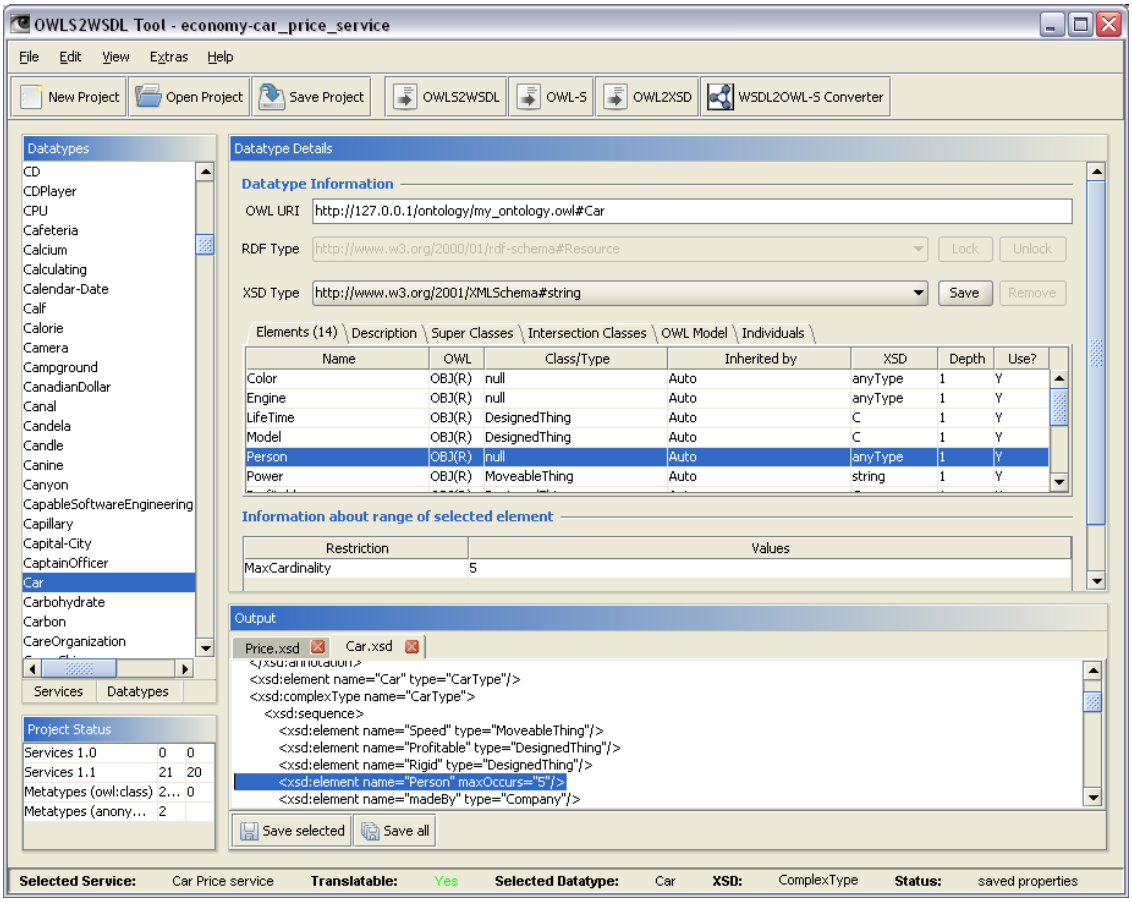


Abbildung 7.4: OWL2XSD. http://127.0.0.1/ontology/my_ontology.owl#Car

Durch die Vererbungstiefe 1 erbt der Typ Car alle Elemente von Auto. Die Spalte **/Use?/** der Elementinformation zeigt an, ob das Element bei aktueller Konfiguration mit in das XML Schema aufgenommen wird. Die Spalte XSD zeigt an, mit welchem Datentyp ein Element interpretiert wird. Da für Person kein Typ in OWL gefunden wurde (**/Class/** ist `null`), wird dem Element kein Typ zugeordnet. Das Schema bleibt trotzdem gültig (impliziter Gebrauch von anyType).

7.4 Translation einer OWL-S Definition (GUI)

Dieser Abschnitt beschreibt die Hauptfunktion des Tools, die Translation von OWL-S Definitionen (Version 1.0 und 1.1) nach WSDL Beschreibungen.

7.4.1 Projekte anlegen und verwalten

Das Tool bietet die Möglichkeit, Service-Beschreibungen inklusive ihrer Abhängigkeiten zu Datentypen mittels Projekten zu organisieren. Ein Projekt enthält eine Service-Kollektion und verknüpft diese mit der Wissensbasis, die die Datentypen enthält. So können für Parameter einer Service-Beschreibung die entsprechenden Datentypen verwaltet werden.

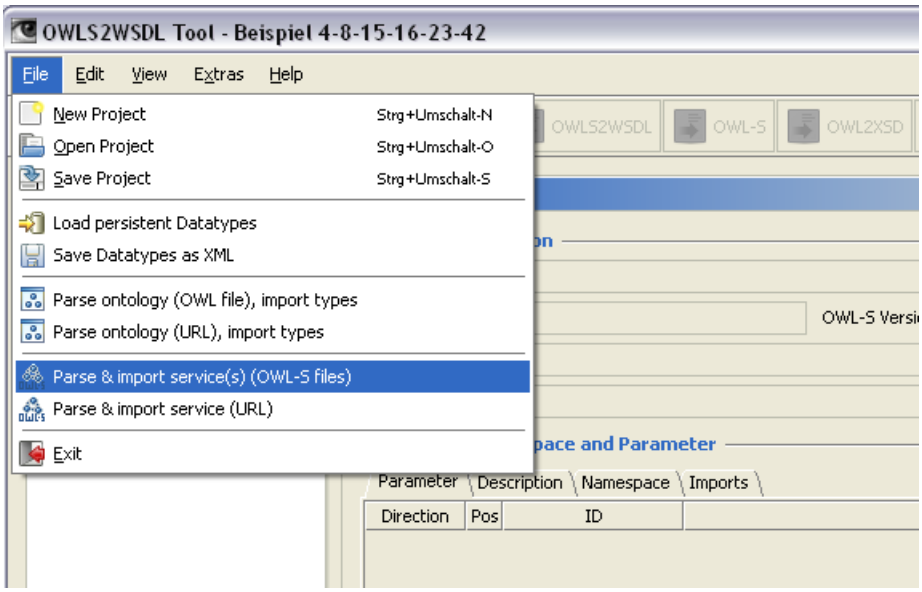


Abbildung 7.5: OWLS2WSDL Tool. Zusammenstellung eines Projekts.

Vorgehensweise:

1. Bevor Service-Beschreibungen geparst und in der Service-Kollektion abgelegt werden können, muss ein neues Projekt angelegt werden.

2. Für ein neues Projekt lässt sich die Service-Kollektion aufbauen, indem OWL-S Definitionen aus dem Dateisystem oder über Angabe der Webadresse importiert werden. Der Import Dialog des Tools erlaubt die Angabe von mehreren OWL-S Dateien und die Angabe von Verzeichnissen. Die Dateien müssen die Endung .owls besitzen (Abb. 7.6).

3. Datentypen können über das Parsen von Ontologien zu der Projekt-Wissensbasis hinzugefügt werden (siehe Abschnitt 7.2.1).

4. Bereits erstelle Datentypen, die zu Wissenbasis (KB) zusammengefasst und persistent abgespeichert wurden, können über die Funktion **Load persistent Datatypes** der Projekt-Wissensbasis hinzugefügt werden.

Das Programm überprüft bei Änderung des Projekts ständig die Abhängigkeiten der Service-Beschreibungen zu den referenzierten Datentypen und erlaubt eine Übersetzung nur dann, wenn die Abhängigkeiten aufgelöst werden können.

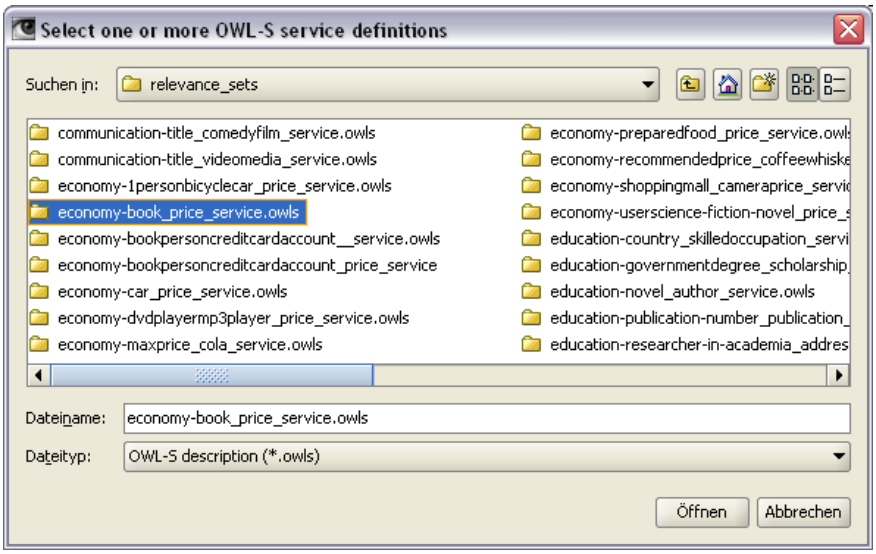


Abbildung 7.6: OWLS2WSDL Tool. Projekt. Import Dialog.

5. Inhalte (Services und Datentypen) können gelöscht werden.
6. Projekte können als XML Dokumente persistent speichern.
7. Die Wissensbasis des Projekts kann separat als XML Dokument gespeichert werden. Dafür benutzt wird der Menüpunkt **Save Datatypes as XML** im Menü.

7.4.2 Service-Information

Ein Beispiel einer aus der OWLS-TC:

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<rdf:RDF
xmlns:owl = "http://www.w3.org/2002/07/owl#"
xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:service = "http://www.daml.org/services/owl-s/1.1/Service.owl#"
xmlns:process = "http://www.daml.org/services/owl-s/1.1/Process.owl#"
xmlns:profile = "http://www.daml.org/services/owl-s/1.1/Profile.owl#"
xmlns:grounding = "http://www.daml.org/services/owl-s/1.1/Grounding.owl#"
xml:base = "http://127.0.0.1/services/1.1/car_recommendedpriceineuro_service.owls">

<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://127.0.0.1/ontology/Service.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Process.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Profile.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Grounding.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/my_ontology.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/concept.owl" />
</owl:Ontology>
```

```
<service:Service rdf:ID="CAR_RECOMMENDEDPRICEINEURO_SERVICE">
<service:presents rdf:resource="#CAR_RECOMMENDEDPRICEINEURO_PROFILE"/>
<service:describedBy rdf:resource="#CAR_RECOMMENDEDPRICEINEURO_PROCESS_MODEL"/>
<service:supports rdf:resource="#CAR_RECOMMENDEDPRICEINEURO_GROUNDING"/>
</service:Service>

<!-- snip -->

<process:AtomicProcess rdf:ID="CAR_RECOMMENDEDPRICEINEURO_PROCESS">
<process:hasInput  rdf:resource="#_CAR"/>
<process:hasOutput rdf:resource="#_RECOMMENDEDPRICEINEURO"/>
</process:AtomicProcess>

<process:Input rdf:ID="_CAR">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://127.0.0.1/ontology/my_ontology.owl#Car
</process:parameterType>
</process:Input>

<process:Output  rdf:ID="_RECOMMENDEDPRICEINEURO">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://127.0.0.1/ontology/concept.owl#RecommendedPriceInEuro
</process:parameterType>
</process:Output>

<!-- snip -->

</rdf:RDF>
```

Sobald eine Service-Bezeichnung in der Service-Kollektion markiert wird, werden detaillierte Informationen zu einem Service eingeblendet (ID, Service Name im Profile, Basename, Version, Imports, Beschreibung, Namespace).

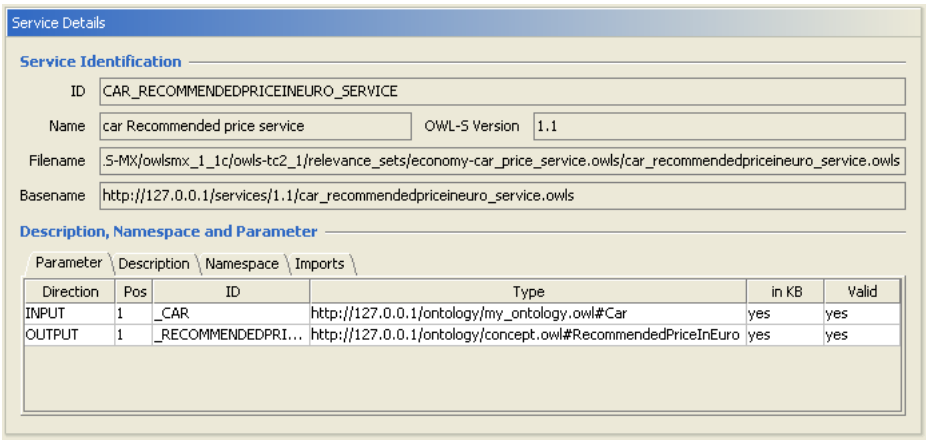


Abbildung 7.7: OWLS2WSDL. Service Details.

Die Abbildung zeigt Details zu dem Service `car_recommendedpriceineuro_service.owls`, dessen Signatur aus 2 Parametern besteht. Diese referenzieren die unterschiedlichen OWL Konzepte Car und RecommendedPriceInEuro.

7.4.3 WSDL Beschreibung

Die Generierung der WSDL Beschreibung wird mit den Button **OWLS2WSDL** gestartet. Die Translation der referenzierten OWL Konzepte (Parameter-Typen) nach XML Schema (OWL2XSD), die in Abschnitt 7.3 vorgestellt wurde, ist impliziert. Die generierte WSDL Beschreibung wird in dem **Output Panel** angezeigt.

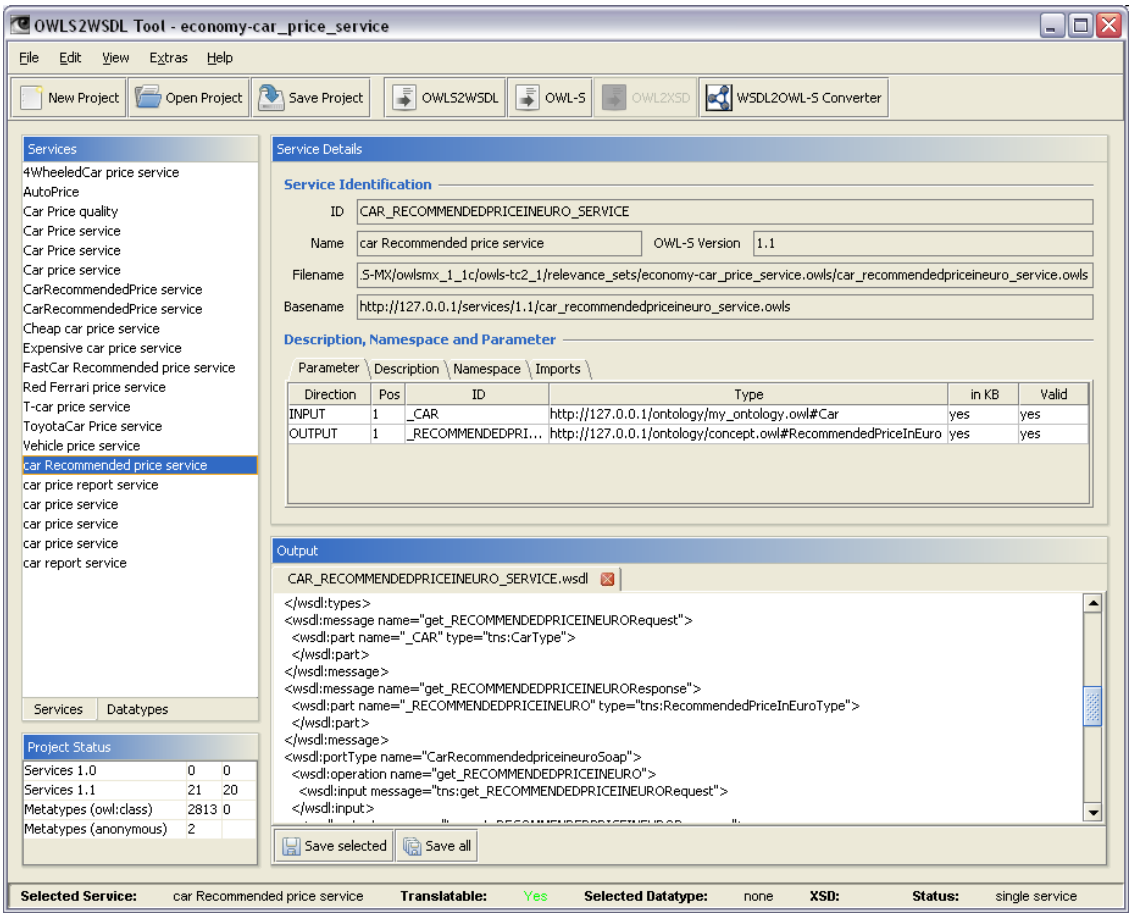


Abbildung 7.8: OWLS2WSDL. WSDL Output.

Der Outputparameter des Beispiels 7.8 ist vom Typ RecommendedPriceInEuro. Die XML Schema Generierung für diesen Typ wurde auf Seite 72 vorgestellt. Der Ausschnitt des generierten Quellcodes zeigt zwei *WSDL Nachrichten*, eine *Request-* und eine *Response Message*, die zusammen die *WSDL Operation* `getRECOMMENDEDPRICEINEURO` beschreiben.

Das XML Schema wird in die WSDL Beschreibung integriert *(types)*. Die Informationen zu Adresse und Angaben zum Namespace werden automatisch in die WSDL Beschreibung geschrieben. In einer künftigen Version des Tools sollen diese Informationen in der Service-Beschreibung manuell gesetzt werden können.

7.4.4 Generierung von mehreren WSDL Beschreibungen (Batch Processing)

Will man mehrere OWL-S Definitionen auf einmal zu WSDL Beschreibungen übersetzen, z.B. mit der Absicht eine Kandidatenmenge für ein Matchmaking im *WSDL Analyzer* zu erstellen, kann die *Batch Processing* Funktion des Tools genutzt werden. In die Exportmenge aufgenommen werden alle, in der Service-Liste (Service-Kollektion) markierten, Services. Über die Funktion **Edit Select all Services** können alle Services markiert werden.

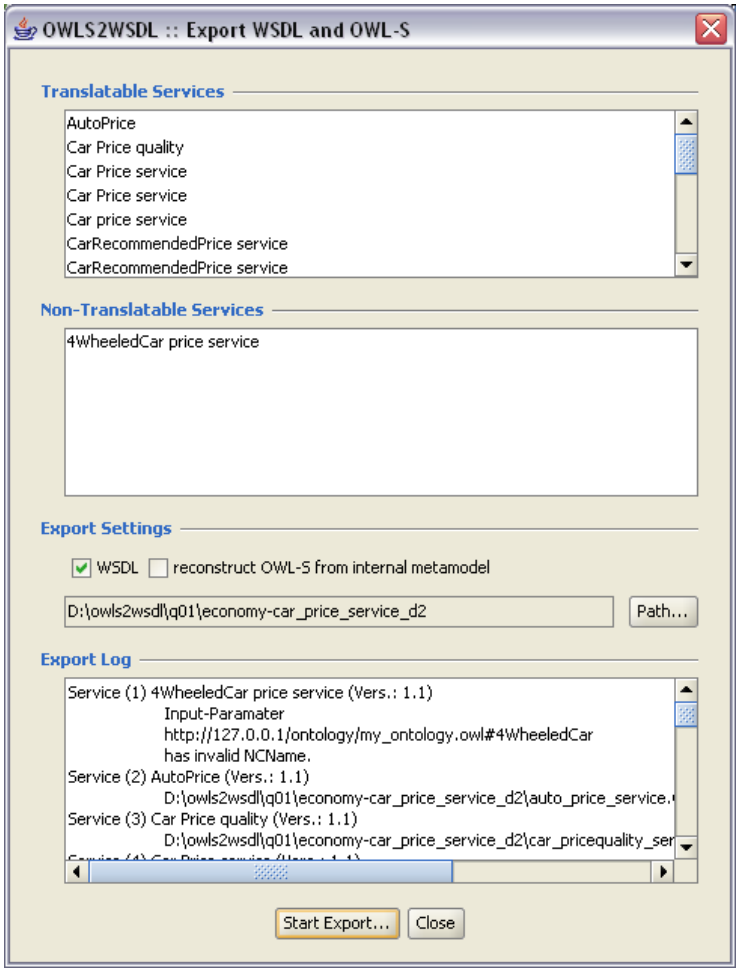


Abbildung 7.9: OWLS2WSDL. Batch Processing.

Die Funktion kann über den Menüpunkt **Extras** - **Export WSDL/OWL-S** aufgerufen werden.

Bemerkung:
Die Abbildung 7.9 unterteilt die ausgewählte Servicemenge in übersetzbare und nicht-übersetzbare Mengen. Gründe dafür, dass ein Service nicht übersetzt werden kann ist die fehlende Information eines Sub-Datentyps oder ein Fehler in der Namensgebung des XML Schema Typs (Vorgaben XML Schema Spezifikation).

7.4.5 Re-Engineering

Das *Re-Engineering von OWL-S* basiert auf der sequentiellen Generierung der WSDL Beschreibung aus der Service-Information (Projekt) und der Konvertierung der so erzeugten WSDL Beschreibung zu OWL-S. Dieser Ablauf wurde auf zwei Arten implementiert: Da die Funktionalität auf dem *WSDL2OWL-S Converter* (GUI, OWL-S API) basiert, wurde dieser komplett integriert. Der markierte Service wird über den Button **Load Selected Service** in den Converter geladen. Die Integration ermöglicht eine Übersicht der WSDL Message Parameter und der OWL-S Prozess Parameter mit den jeweiliger Klassen- bzw. Typinformation aus der Wissensbasis. Das Mapping wird automatisch anhand der im Projekt gespeicherten Service-Information durchgeführt. Die Generierung der OWL-S Definition lässt sich über den Button **Generate OWL-S** antriggern.

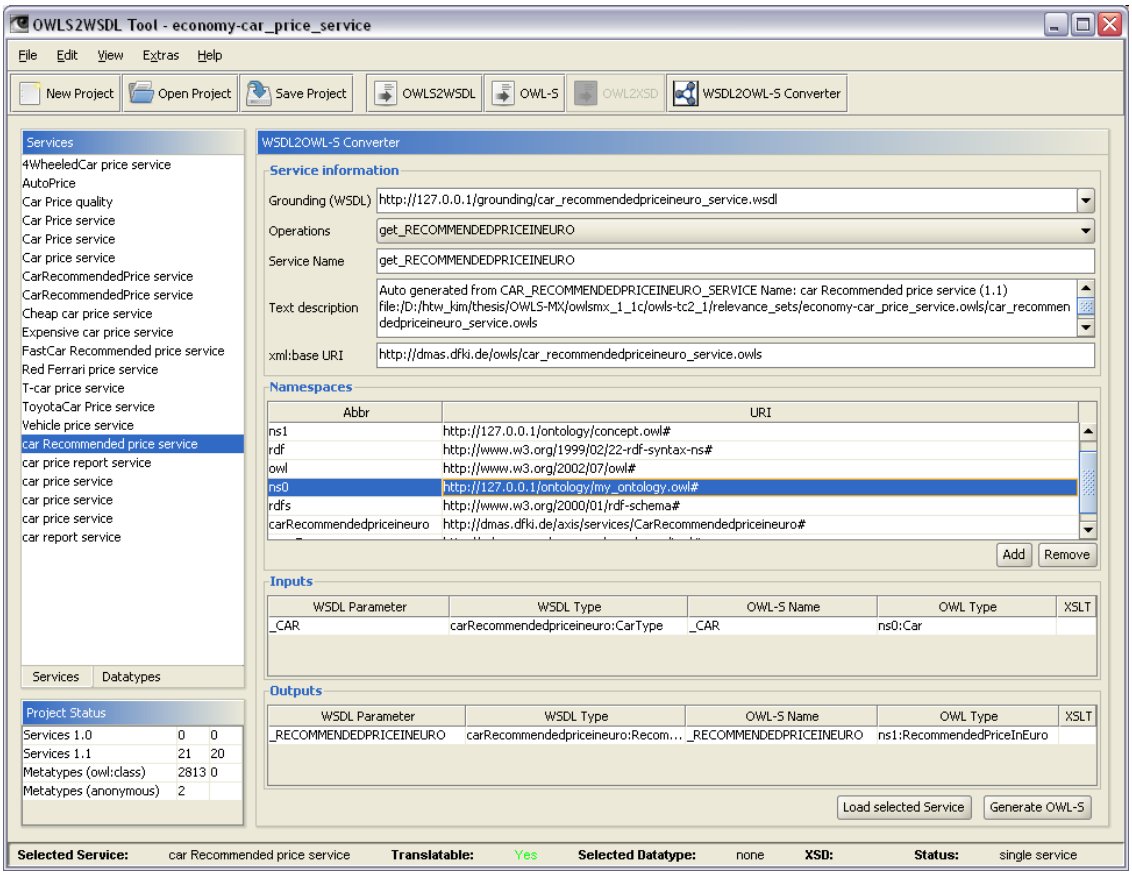


Abbildung 7.10: Integrierter WSDL2OWL-S Converter (OWL-S API)

Zusätzlich wurde der Button **OWL-S** eingeführt, der für den markierten Service die Generierung der OWL-S Definition antriggert, die dann im **Output Panel** angezeigt wird. Um mehrere OWL-S Definitionen generieren zu können, wurde die Export Funktion (siehe Abschnitt 7.4.4) um ein *Batch Processing* für OWL-S erweitert.

7.5 Walkthrough: Translation der OWLS-TC und WSDL Matchmaking

Grundlage für die Translation ist die Erstellung eines Projekts. (Abschnitt 7.4.1, Abbildung 7.11). Danach empfiehlt sich der Import aller OWL-S Definitionen eines *Relevance Set* über die Auswahl eines kompletten Verzeichnisses der OWLS-TC.

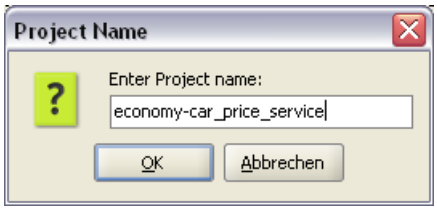


Abbildung 7.11: Walkthrough. Projekt anlegen.

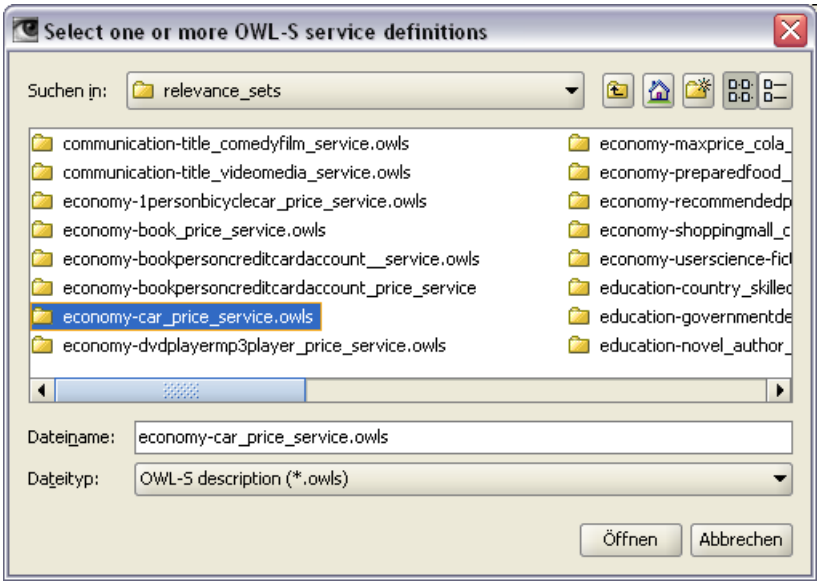


Abbildung 7.12: Walkthrough. Import Relevance Set.

Sind die Service-Informationen geladen, hilft das Panel **Projekt Details** (Abb. 7.13), welche Parametertypen aufgrund von Abhängigkeiten benötigt werden. Die Datentypen, die die von den Prozess Parametern benötigten Konzepte widerspiegeln, müssen nun zu dem Projekt hinzugefügt werden. Dazu lassen sich Ontologien direkt aus der GUI parsen oder bequemer, die Wissensbasen (KB) bereits vorher geparster Ontologien laden.

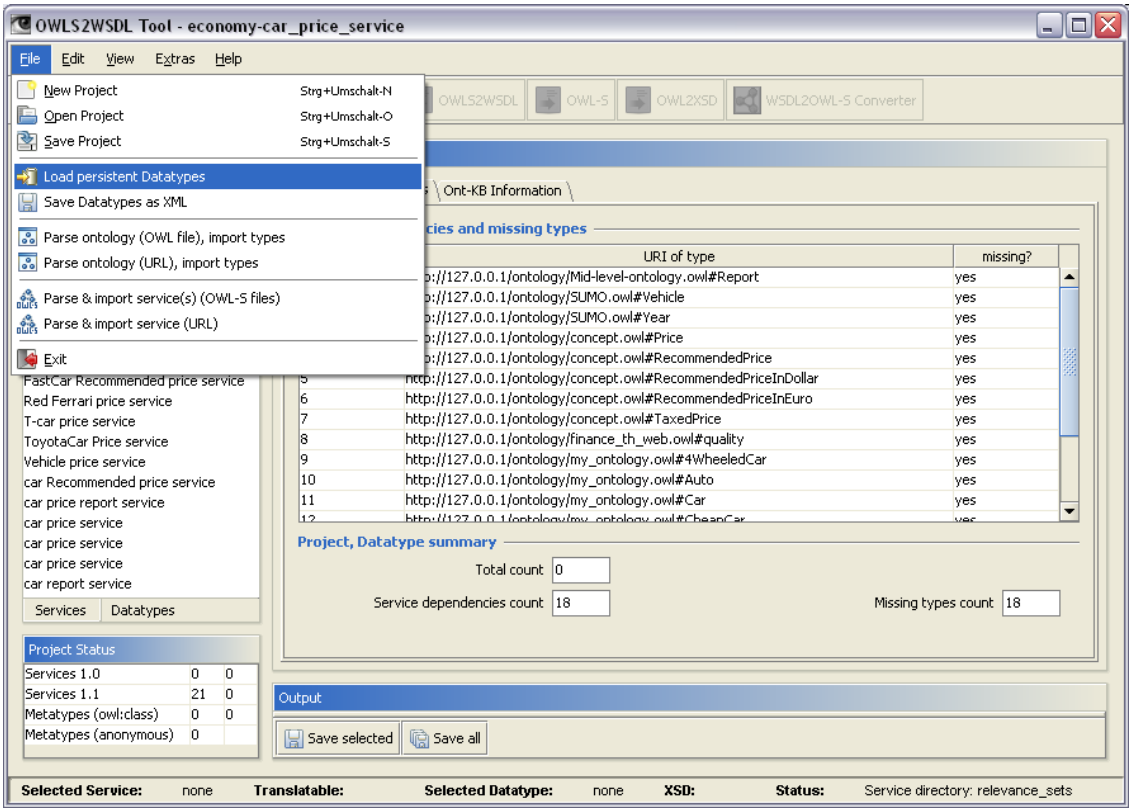


Abbildung 7.13: Walkthrough. Laden der Datentypen.

Der letzte Schritt muss solange durchgeführt werden bis alle Abhängigkeiten aufgelöst sind. Die **Export Funktion** (Abb. 7.9) kann eingesetzt werden, um die markierten WSDL Beschreibungen der Service-Kollektion als WSDL Kandidatenmenge in einem neuen Verzeichnis abzulegen.

Für jede Konfiguration des XML Schema Generators (Abschnitt 7.3.1) sollte ein eigenes Verzeichnis angelegt werden, in dem die generierten WSDL Beschreibungen abgelegt werden. Der Verzeichnisname bzw. die Verzeichnisstruktur sollte der Query-Bezeichnung aus der OWLS-TC entsprechen (Nummer und Name des *Query*).

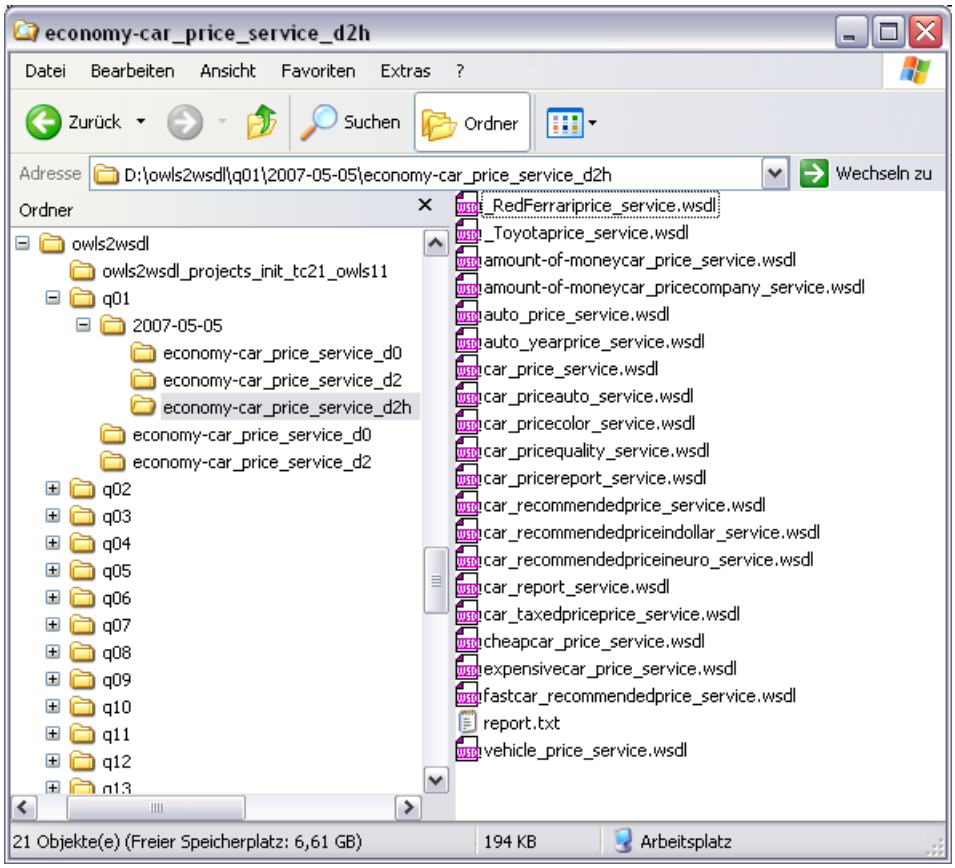


Abbildung 7.14: Walkthrough. WSDL Relevance Sets.

Der *WSDL Analyzer* (WA) unterstützt den Import eines kompletten Verzeichnisses. Die generierten WSDL Beschreibungen des OWLS-MX Relevance Sets werden so als Kandidaten in den WA geladen. Als Requirement wird der *Query* Service ausgewählt.

Zusammenfassung

Abbildung 7.15 zeigt eine Zusammenfassung aller Schritte, die notwendig sind, um aus der Testkollektion (OWLS-TC) WSDL Projekte zu erzeugen, die mit dem *WSDL Analyzer* verarbeitet werden können.

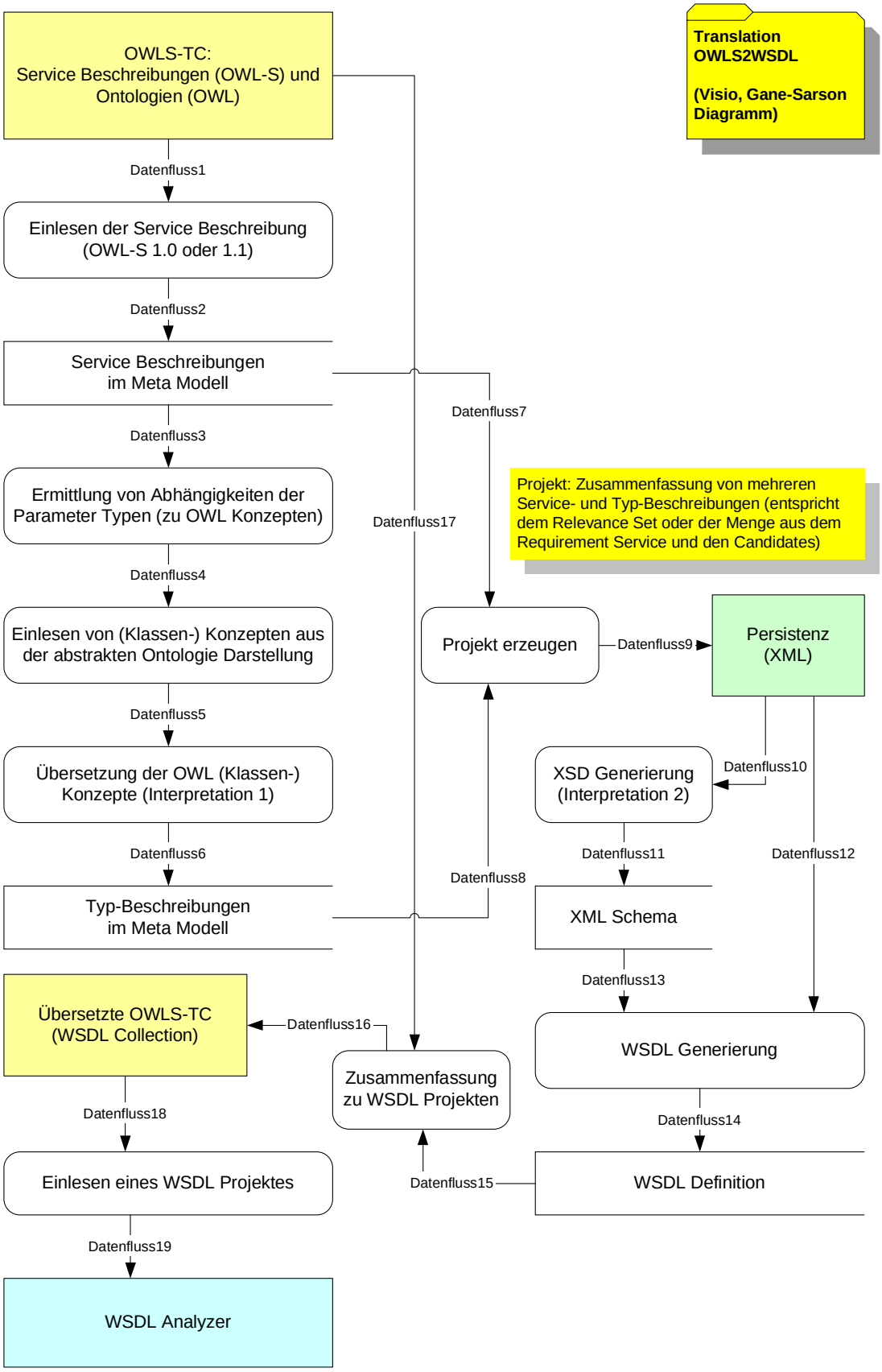


Abbildung 7.15: Translation der OWLS-TC

7.6 Data Binding (CodeGen und WSDL2Java)

In einem *top down* Ansatz (siehe Abschnitt 2.2.7) ist es möglich, anhand der ursprünglichen OWL-S und OWL Definitionen Java Quellcode zu generieren. WSDL Beschreibungen und XML Schema Definitionen sind dabei die benötigten Zwischenformate.
Der Source Code Generator des *Castor*[1] Projekts kann aus XML Schema Java Quellcode generieren. Er interpretiert auch das für das XML Schema genutzte *Hierarchy Pattern* und baut eine Vererbungsstruktur in Java auf (Spezialisierung).

```
public class CarType extends car.AutoType
implements java.io.Serializable
{
    private java.util.Vector _personList;
    private java.util.Vector _wheelList;
...
```

Listing 7.7: Castor Code Generator. Vererbung.

Um anhand der WSDL Beschreibung den Java Quellcode für die Service Schnittstelle zu generieren, kann das Tool **WSDL2Java** des Apache *Axis*[2] Projekts genutzt werden. Generiert werden alle Klassen, die benötigt werden, um einen Web Service zu implementieren.

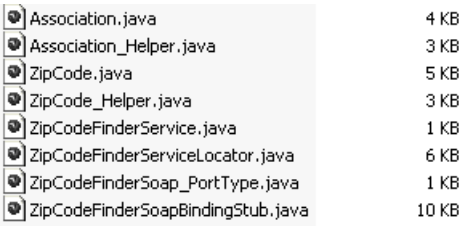


Abbildung 7.16: WSDL2Java

```
public class ZipCode  implements java.io.Serializable {
    private java.lang.String zip;
    private java.lang.String city;
    private zipCodeGenerated.wsdl.Association defaultAssociation;
    private zipCodeGenerated.wsdl.Association association;
...
```

Listing 7.8: Axis Code Generator. Aggregation.

Zusammenspiel von Axis und Castor: Interessant ist die Verwendung des *Castor* Code Generators zusammen mit *Axis*. Diese Kombination eignet sich sehr gut zur Realisierung von Web Services, deren Schnittstellen aus umfangreichen komplexen XML Schema Typen aufgebaut sind. Über die Konfiguration von *Axis* lassen sich die Standard-Mechanismen zum Serialisiern und Deserialisieren von Request- und Response Nachrichten ändern. So können die Fabrikklassen *(factories)*, die Serializer und Deserializer für die Request- und Response Typen bauen, durch entsprechende *Castor* Fabrikklassen ausgetauscht werden. [FP07, KG03]

[1] http://www.castor.org
[2] http://ws.apache.org/axis/

Kapitel 8

Experimentelle Evaluierung

Wichtige Aspekte dieser Arbeit sind die Analyse der generierten WSDL Beschreibungen und die inhaltliche Untersuchung (Evaluierung) der Translation durch den Vergleich der beiden Matchmaker OWLS-MX *und* WSDL Analyzer. *Das Kapitel zeigt Diagramme mit Ergebnissen der beiden Matchmaker, um den Vergleich der ermittelten Ähnlichkeitswerte zu ermöglichen.*

OWL-S Referenzdienste und Ontologien finden sich innerhalb der *OWL-S Service Retrival Test Collection* (OWLS-TC). Anhand der OWL-TC soll mit Hilfe der Translation eine entsprechend aufgebaute Testkollektion generiert werden, die aus WSDL Beschreibung besteht.

8.1 Die Technologien

Eine Teilaufgabe dieser Arbeit lautet: "Vergleichende Analyse der Ergebnisse einer ähnlichkeitsbasierten Dienstsuche mit dem *WSDL Analyzer* (WA) und dem *OWLS-MX*."
Beides sind Matchmaker für Service-Beschreibungen.

Der Referenzdienst wird folgend als *Requirement* (WA) oder *Query* (OWLS-MX) bezeichnet. Die zu vergleichenden Service-Beschreibungen (WSDL, OWL-S) werden als *Candidates* (WA) oder *Relevance Set* (OWLS-MX) bezeichnet. Das *Relevance Set* ist dabei eine manuell festgelegte Menge, die der OWLS-MX primär zur Bestimmung von Ähnlichkeiteswerten betrachtet. Innerhalb einer ähnlichkeitsbasierten Dienstsuche betrachtet der OWLS-MX alle registrierten Dienstbeschreibungen. Die Resultate eines Matchmakings werden in einem *Ranking* gelistet.

8.1.1 Der WSDL Analyzer

Der *WSDL Analyzer* (WA) wurde im Rahmen des **ATHENA IP** Projekts zur Unterstützung der Implementierung von *Serviceorientierten Architekturen* (SOAs) erstellt (siehe [ZGBFV06]). Vorgestellt wird er in [ZRF05]: *"The WSDL Analyzer is first and foremost a tool for detecting similarities and differences between WSDL files. The tool can be used to find a list of similar services (matchmaking, service discovery)."* Zusätzlich erzeugt der WA Mappings zwischen Diensten und den Schnittstellen der Dienste. Diese Funktion ist für die Implementierung einer SOA interessant, da eine flexible Integration von Kollaborationspartnern ermöglicht wird.

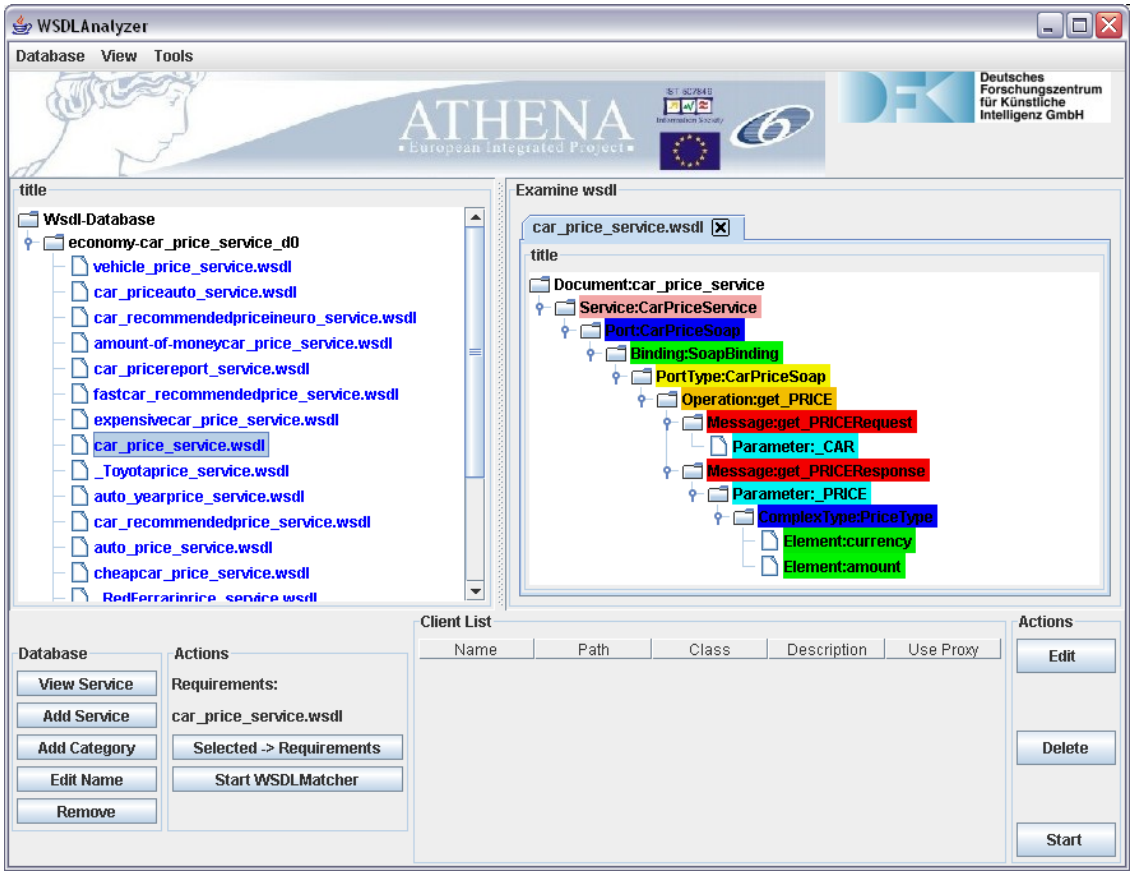


Abbildung 8.1: WSDL Analyzer: Übersicht über geladene Service-Beschreibungen

Der WA untersucht WSDL Beschreibungen anhand ihrer Struktur, anhand der verwendeten Terminologie und mit Hilfe des *Stemming* Verfahrens aus dem Bereich des *Information Retrieval* (IR). Theoretische Hintergründe sind das *tree-edit distance measure* Verfahren und das Konzept von *weak subsumption* Beziehungen [ZRF05]. Die Konfiguration des WA Matching Algorithmus wird durch die Gewichtung der Parameter Structure Weight, Name Weight und Wordnet Weight vorgenommen (siehe Abbildung 8.2).

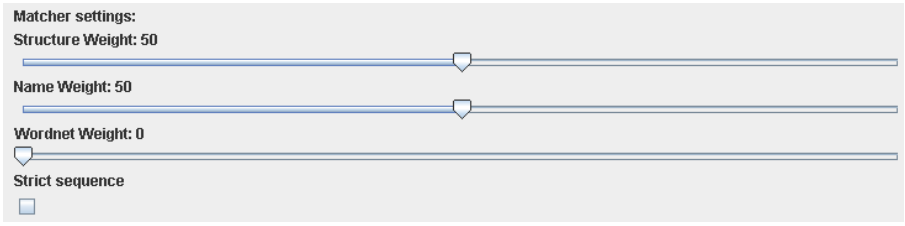


Abbildung 8.2: WSDL Analyzer. Settings

Die Abfrage der **WordNet**[1] Datenbank ist eine zusätzlich integrierte Funktion, um nach semantischen und lexikalischen Beziehungen zwischen Wörtern zu suchen. Der Rückgabewert fließt in die Berechnung des Ähnlichkeitswertes mit ein.

[1] http://wordnet.princeton.edu

Darstellung von Matching Ergebnissen (WA)

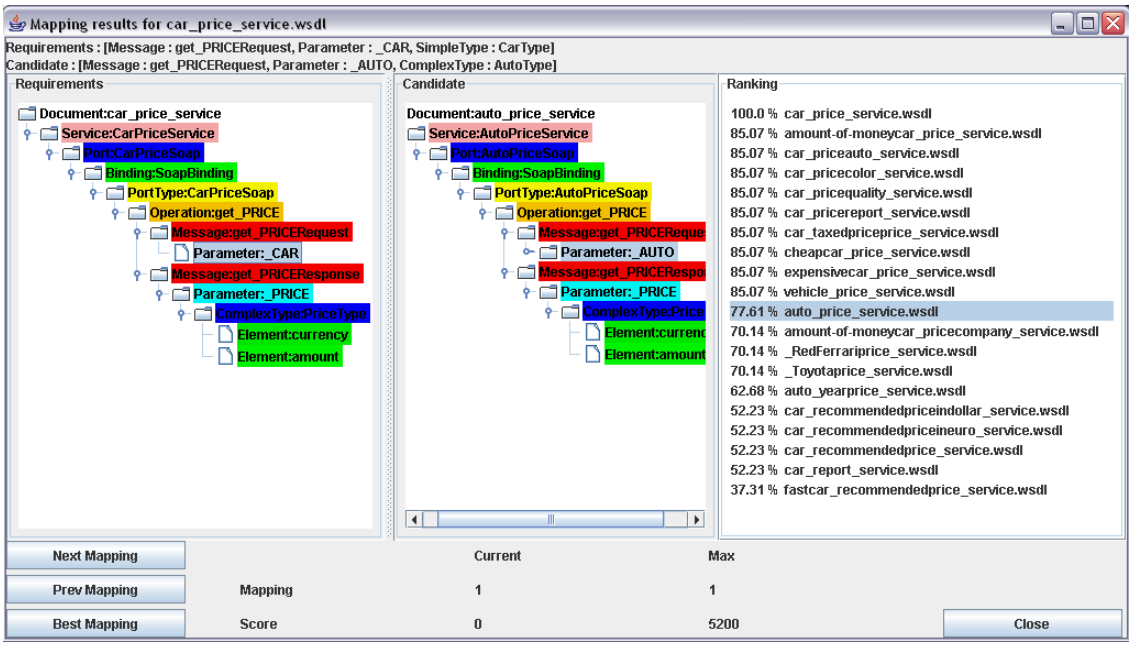


Abbildung 8.3: WSDL Analyzer. Ergebnisse eines Queries der OWLS-TC.

Bemerkungen

- Das WordNet konnte im Rahmen dieser Arbeit nicht benutzt werden, da der vorliegenden Implementierung die Funktionalität fehlte.
- Trotz der Erweiterungen, die während dieser Arbeit eingeflossen sind, kann der WA noch nicht alle verwendeten Schema Definitionen verarbeiten. Gerade die Verarbeitung von Zyklen muss künftig noch weiter verbessert werden.

Erweiterungen

Im Rahmen dieser Arbeit wurde die Kompatibilität des *WSDL Analyzers* bzgl. der einzulesenden XML Schema Definitionen bereits erweitert. Einerseits betrifft dies die Erweiterung des verwendeten Parsers und die Erweiterung des internen Datenmodells, um bestimmte XML Schema Strukturen lesen und verarbeiten zu können (siehe Abschnitt 8.3). Andererseits musste eine Erweiterung erstellt werden, um die gefundenen Ähnlichkeitswerte in einem Format abspeichern zu können, das die einfache Erstellung von Diagrammen erlaubt. Verwendet wurde XML, um die Daten persistent zu speichern. Eine Umwandlung nach CSV ermöglicht das Lesen der Daten mit *Excel* oder *OpenOffice*.

8.1.2 Der OWLS-MX

Der *OWLS-MX*[2] [BF06] ist ein am DFKI entwickelter hybrid-semantischer Matchmaker. Er untersucht Service-Beschreibungen semantisch und syntaktisch.

Anwendung: Ein Dienstnehmer *(requester)* (Nutzer/Agent) der weiß, welche Fähigkeiten er von einem Dienst erwartet, will diesen oder einen vergleichbaren Dienst auffinden. Die Idee ist, dass der Nutzer einen Wunschdienst modelliert und der Matchmaker diesen oder einen semantisch ähnlichen Dienst aus einer Menge von Diensten findet *(discovery)*. Bestimmt wird die semantische Ähnlichkeit durch Auswertung der OWL Klassenhierarchie (DB) sowie durch Betrachtung der verwendeten deskriptiven Logik (OWL-DL). Der Einsatz deskriptiver Logik erweitert die Klassenhierarchie und verändert Klasseneigenschaften. Zur Bestimmung der semantischen Ähnlichkeit wird eine *Matchmaking Engine* eingesetzt (generalisierendes Matchmaking), die auf einer *Inference Engine* bzw. einem *Reasoner* basiert.

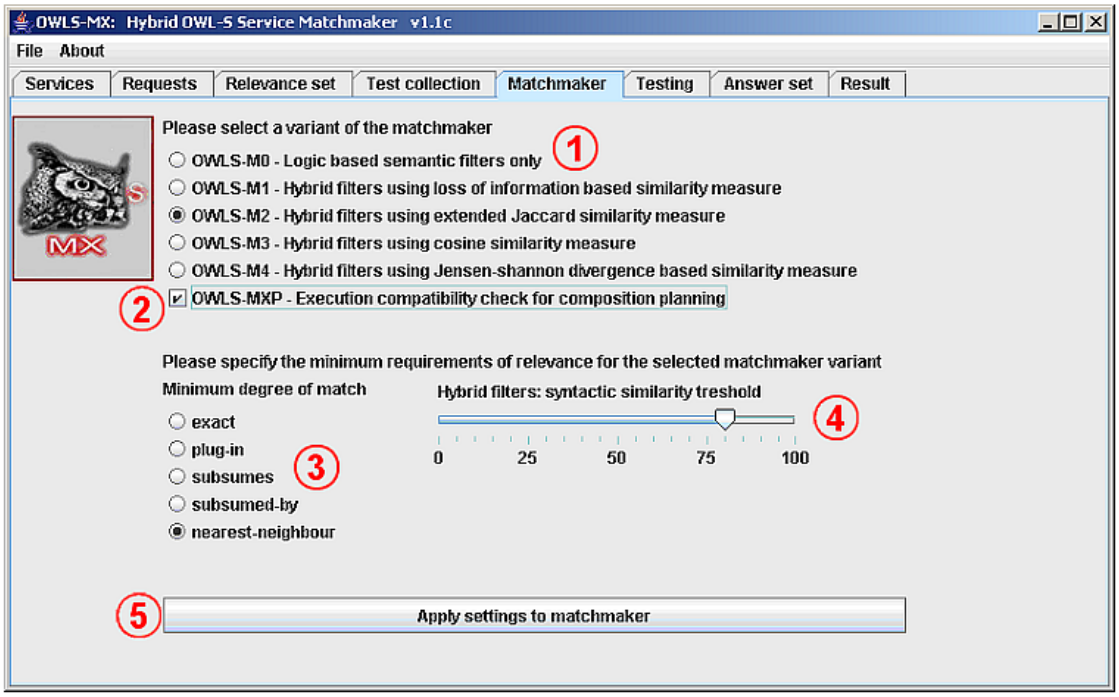


Abbildung 8.4: OWLS-MX. Konfigurationsmöglichkeiten

Bei dem OWLS-MX handelt es sich um einen hybrid-semantisch arbeitetenden Matchmaker. Er besitzt eine syntaktische (IR Metriken) und eine semantische Matchmaking Engine [BF06], die wie folgt konfiguriert werden. In (1) lässt sich dabei die Variante festlegen, die benutzt werden soll. Durch Aktivierung von (2) wird überprüft, ob die gefundenen OWL-S Definitionen ein Grounding in WSDL besitzen. Die erforderliche semantische Ähnlichkeit wird in (3) gesetzt. Mit (4) können Grenzwerte für die hybriden Filter gesetzt werden. (5) wendet die Konfiguration auf die aktuelle Testkollektion an.

[2] http://projects.semwebcentral.org/projects/owls-mx

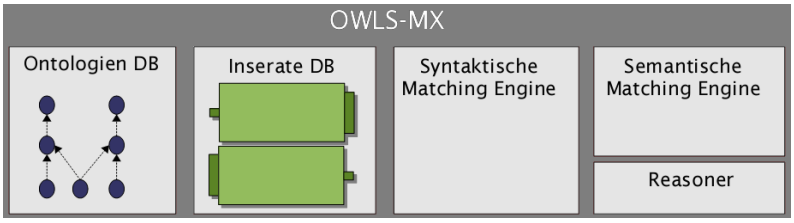


Abbildung 8.5: OWLS-MX. Architektur [BF06]

Für den OWLS-MX wurde eine Testkollektion erstellt, die *OWL-S Service Retrival Test Collection* (OWLS-TC). Die OWLS-TC enthält mehrere Matchingscenarios (Queries) und bereits ermittelte Ähnlichkeitswerte. Sie sollen mit den Resultaten des *WSDL Analyzers* verglichen werden (Abschnitt 8.5). Die Werte der OWLS-TC basieren auf folgenden IR-Metriken:

- Cosine (Cos) metric
- Extended Jaccard (EJ) metric
- Jensen Shannon (JS) metric

8.2 Validierung

Bevor die generierten WSDL Beschreibungen für ein Matchmaking genutzt werden können, muss sichergestellt werden, dass die WSDL Beschreibung inklusive der XML Schema Definition validieren. Durch die Nutzung der Objektmodelle von Castor und WSDL4J kann bereits eine erste Garantie abgegeben werden, dass die generierten Beschreibungen korrekt sind. Die API enthalten auch bereits Validatoren, die genutzt werden können, um das jeweilige Objektmodell zu prüfen. Um die generierten WSDL Beschreibungen für den Gebrauch in dem WSDL Analyzer frei zu geben, wird zusätzlich eine Validierung in XMLSpy von ALTOVA[3] durchgeführt. XMLSpy ist als XML Editor industrieller Standard.

8.3 Voraussetzungen

Voraussetzung für die Analyse und ein Vergleich der Matchmaking Ergebnisse ist die korrekte Arbeitsweise des *WSDL Analyzers* (WA). Die generierten WSDL Beschreibungen müssen geparst und verarbeitet werden. Dazu werden auch zwei Programmteile des WA unterschieden: Der **Parser**, der das WSDL File einliest und das interne Datenmodell aufbaut, und der **Matcher**, der WSDL Service-Beschreibungen (Kandidaten) mit einem *Requirement* Service vergleicht und dann die Ähnlichkeitswerte der Kandidaten zu dem *Requirement* bestimmt. Für den Vergleich unterschiedlicher XML Schema Typen gibt es jeweils eigene Methoden (SimpleType Matching, ComplexType Matching, MixedType Matching).

[3] http://www.altova.com

Die im Rahmen dieser Arbeit erstellten WSDL Beschreibungen können nicht alle in den *WSDL Analyzer* geladen und verarbeitet werden. Aufgrund der Anforderung an den WA, alle validierten WSDL Beschreibungen lesen und verarbeiten zu können, wurde die Kompatiblität zu XML Schema im Rahmen dieser Arbeit verbessert. Für die WSDL Beschreibungen, die bis zur Abgabe der Arbeit noch nicht verarbeitet werden können, wird eine entsprechende Anforderung dokumentiert. Das in Kapitel 4 gezeigte Vorgehensmodell zeigt die Bedeutung des *WSDL Analyzers* für die Entwicklung der Translation von OWL-S nach WSDL.

Kritische Punkte bei der Arbeit mit dem WA sind:

- Zyklen innerhalb der XML Schema Definition. Beim Aufbau eine Matchingstruktur ab dem Message Knoten innerhalb des WSDL Modells führen Zyklen aufgrund resultierender Schleifen bei der Abarbeitung der Schematypen dazu, dass Definitionen nicht eingelesen werden. Hier müssen geeignete Abbruchbedingungen für den Parser gefunden werden, die das Einlesen und das spätere Matchen ermöglichen (teilweise gelöst).
- Nutzung eines komplexen Basistypen innerhalb der XML Schema Definition. Basistypen werden genutzt, um die Typhierarchie abbilden zu können (gelöst).
- Es gibt verschiedene Matchmaking-Methoden, die je nach Anwendungsfall benutzt werden *(matchSimpleType matchComplexType matchMixedType)*. Eine Problem tritt in Zusammenhang mit definierten Zyklen auf (teilweise gelöst). Die Abbildung 8.6 zeigt ein Beispiel.

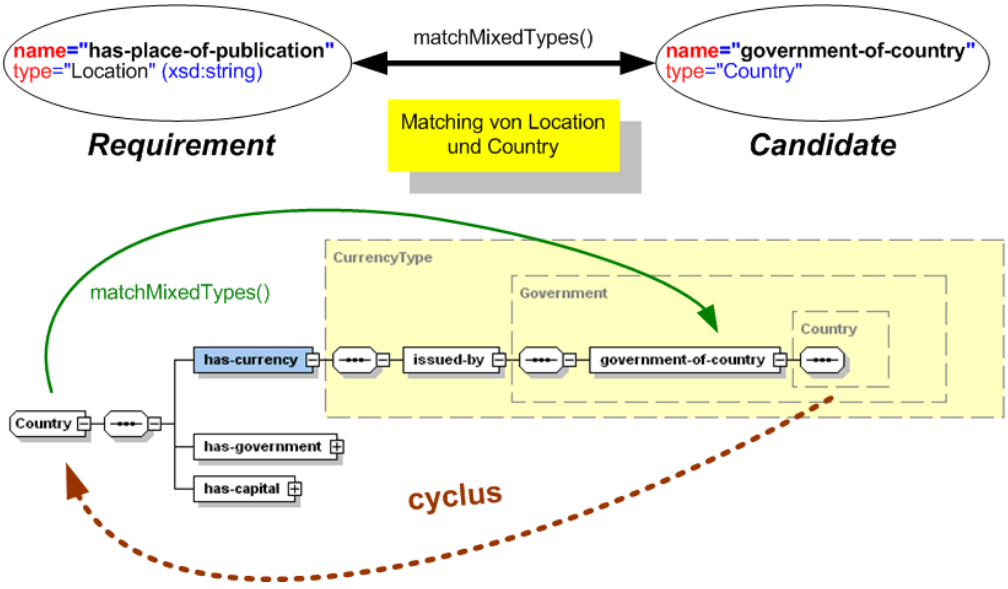


Abbildung 8.6: Beispiel: Zyklus-Problematik beim Matchen von zwei Datentypen.

8.4 Übersetzung der OWLS-MX Testkollektion

Nachdem ein *Query* und das dazugehörige *Relevance Set* der OWLS-MX Testkollektion übersetzt worden ist, können mit dem *WSDL Analyzer* Ähnlichkeitswerte für die generierten WSDL Beschreibungen erzeugt werden und die Resultate mit den Werten des OWLS-MX verglichen werden. Im Abschnitt 7.5 des Benutzerhandbuchs wird schrittweise erklärt, wie man ein OWLS2WSDL Projekt für jeweils einen *Query* der OWLS-MX Testkollektion erstellt und für die OWL-S Definitionen eines *Relevance Sets* entsprechende WSDL Beschreibungen generiert.

8.5 Vergleich der Matchmaking Ergebnisse

Die Translation der OWLS-TC *Queries* und die anschließende Analyse mit dem WA wurde mit folgenden Konfigurationen des XML Schema Generators durchgeführt:

WA(d0) keine Vererbung *(depth=0)*

WA(d1) Vererbung von Eigenschaften der direkten Superklasse *(depth=1)*

WA(d2) Vererbung von Eigenschaften mit Vererbungstiefe 2 *(depth=2)*

WA(d2h) Vererbung von Eigenschaften mit Vererbungstiefe 2 und *Hierarchy Pattern*

Bei Vererbung der Typinformation für `SimpleType` Deklarationen wird zuerst ein möglicher RDF-Type Eintrag betrachtet und danach ein manuell gesetzter primitiver Datentyp. Da für keinen Fall die Wissensbasis manuell bearbeitet wurde, wird der Defaultwert `xsd:string` gesetzt.

Konfigruation des *WSDL Analyzer*:

- Structure Weight: 50
- Name Weight: 50
- Wordnet Weight: 0 (nicht implementiert)
- Strict sequence: no

Query 01: econonmy/car_price_service.wsdl

Candidate service	WA(d0)	Cos	EJ	JS	Semantic degree
car_price_service.wsdl	100,00	98,00	100,00	100,00	Exact
amount-of-moneycar_price_service.wsdl	85,07	91,00	87,00	93,00	Failed
car_priceauto_service.wsdl	85,07	64,00	51,00	61,00	Exact
car_pricecolor_service.wsdl	85,07	57,00	55,00	59,00	Exact
car_pricequality_service.wsdl	85,07	92,00	89,00	93,00	Exact
car_pricereport_service.wsdl	85,07	67,00	57,00	67,00	Exact
car_taxedpriceprice_service.wsdl	85,07	95,00	85,00	94,00	Exact
cheapcar_price_service.wsdl	85,07	97,00	94,00	97,00	Failed
expensivecar_price_service.wsdl	85,07	98,00	99,00	100,00	Failed
vehicle_price_service.wsdl	85,07	64,00	58,00	68,00	Failed
auto_price_service.wsdl	77,61	98,00	99,00	100,00	Plugin
amount-of-moneycar_pricecompany_service.wsdl	70,14	91,00	87,00	93,00	Failed
_RedFerrariprice_service.wsdl	70,14	50,00	50,00	50,00	Exact
_Toyotaprice_service.wsdl	70,14	50,00	50,00	50,00	Exact
auto_yearprice_service.wsdl	62,68	82,00	72,00	81,00	Plugin
car_recommendedpriceindollar_service.wsdl	52,23	92,00	84,00	90,00	Subsumes
car_recommendedpriceineuro_service.wsdl	52,23	92,00	89,00	93,00	Subsumes
car_recommendedprice_service.wsdl	52,23	94,00	89,00	94,00	Plugin
car_report_service.wsdl	52,23	48,00	50,00	47,00	Failed
fastcar_recommendedprice_service.wsdl	37,31	95,00	93,00	96,00	Failed

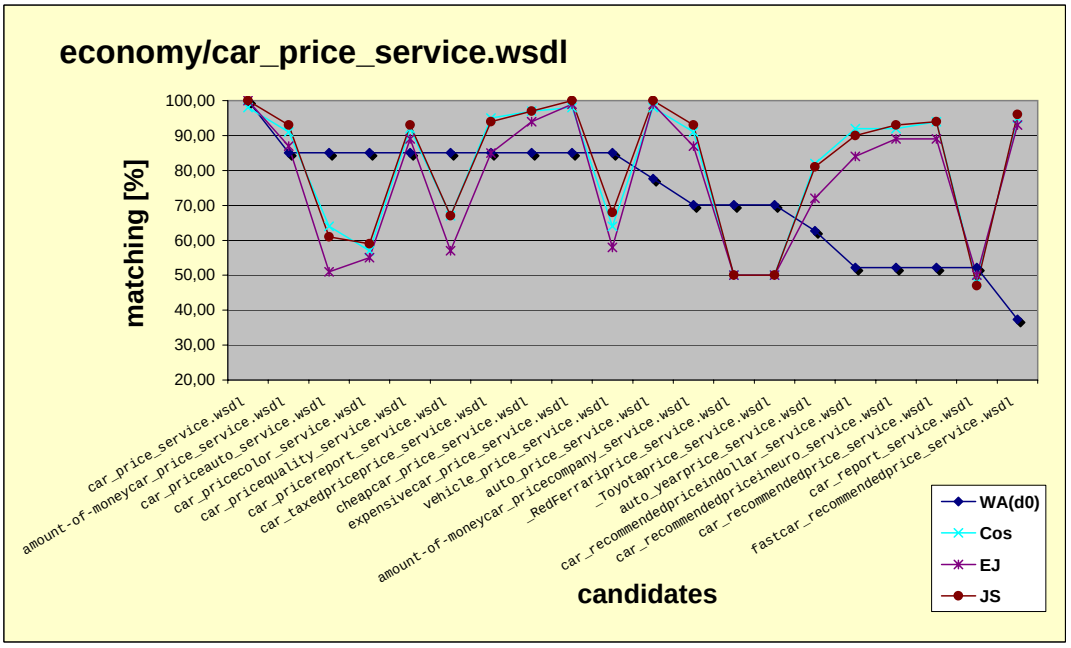


Abbildung 8.7: Query 01, requirement: economy/car_price_service.wsdl

Mit dem *WSDL Analyzer* konnten keine weiteren Auswertungen vorgenommen werden. Vermutung: Die größeren Abweichungen könnten durch Betrachtung weiterer Elemente, die über die Vererbung hinzugefügt werden, ausgeglichen werden.

Ab WA(d1) treten Probleme beim Parsen auf. Gründe:

- Zyklus ab `TimeInterval` kann nicht eingelesen werden.
- Programmabbruch mit Java *heap space exception*.

Query 02: economy/book_price_service.wsdl

Candidate service	WA(d0)	WA(d2)	WA(d2h)	Cos	EJ	JS	Semantic degree
BookPrice.wsdl	100,00	100,00	100,00	99,00	100,00	100,00	Exact
book_Cheapestprice_service.wsdl	100,00	100,00	100,00	99,00	100,00	100,00	Exact
book_price_service.wsdl	100,00	100,00	100,00	99,00	100,00	100,00	Exact
bookpersonOptional_price_service.wsdl	90,90	93,46	89,36	99,00	100,00	100,00	Failed
book_authorprice_Novelservice.wsdl	90,90	93,46	89,36	93,00	89,00	93,00	Exact
book_authorprice_service.wsdl	90,90	93,46	89,36	93,00	89,00	93,00	Exact
book_pricereviewbook_service.wsdl	90,90	93,46	89,36	72,00	61,00	70,00	Exact
book_pricesizebook-type_service.wsdl	90,90	93,46	89,36	93,00	89,00	93,00	Exact
book_reviewprice_service.wsdl	90,90	93,46	89,36	96,00	94,00	96,00	Exact
book_taxedpriceprice_service.wsdl	90,90	93,46	89,36	96,00	86,00	96,00	Exact
book_recommendedpriceindollar_service.w	70,90	93,46	60,63	93,00	89,00	93,00	Subsumes
book_recommendedprice_service.wsdl	70,90	93,46	84,04	96,00	94,00	96,00	Plugin
Tizonbook_recommendedpriceindollar_sen	70,90	93,46	60,63	93,00	89,00	93,00	Subsumes
monograph_price_service.wsdl	57,27	65,35	89,36	86,00	80,00	88,00	Plugin
author_bookprice_service.wsdl	42,72	30,71	50,00	41,00	17,00	34,00	Failed
title_pricebook_service.wsdl	42,72	30,71	50,00	41,00	17,00	35,00	Failed
monograph_recommendedpriceineuro_sen	28,18	58,82	50,00	80,00	68,00	81,00	Subsumes

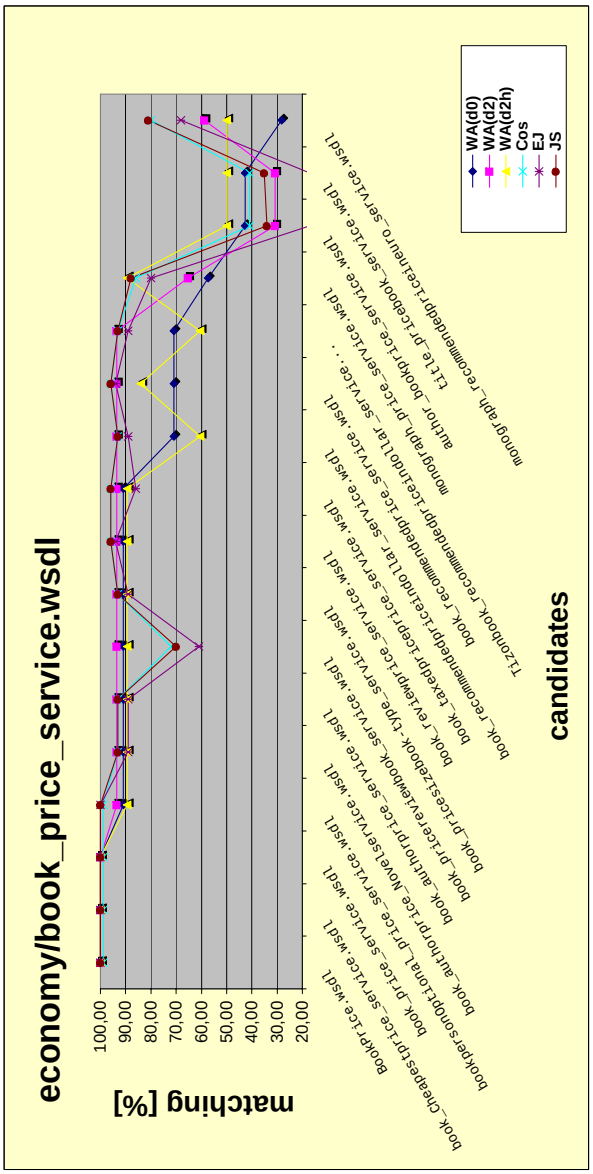


Abbildung 8.8: Query 02, requirement: economy/book_price_service.wsdl

Der Vergleich der Ergebnisse beider Matchmaker für diesen Query zeigt, dass die Ergebnisse des *WSDL Analyzer* in etwa denen des OWLS-MX entsprechen. Das Ergebnis für WA(d2) entspricht dabei am ehesten den Werten für Cos, EJ und JS. Bei WA(d2h) fallen zwei Ausreißer auf:

- book_recommendedpriceindollar_service.wsdl
- Tizonbook_recommendedpriceindollar_service.wsdl

Query 03: economy/dvdplayermp3player_price_serice.wsdl

Candidate service	WA(d0)	WA(d2)	WA(d2h)	Cos	EJ	JS	Semantic degree
dvdplayermp3player_price_MDservice.wsdl	100,00	100,00	100,00	97,00	100,00	100,00	Exact
dvdplayermp3player_price_Rservice.wsdl	100,00	100,00	100,00	92,00	94,00	97,00	Exact
dvdplayermp3player_price_service.wsdl	100,00	100,00	100,00	94,00	98,00	99,00	Exact
dvdplayermp3player_pricemessage_service.wsdl	87,01	87,01	87,01	69,00	58,00	73,00	Exact
mp3playerportabledvdplayer_price_service.wsdl	87,01	87,01	87,01	96,00	98,00	99,00	Failed
mp3playerdvdplayer_priceshipping_service.wsdl	74,02	74,02	74,02	80,00	72,00	81,00	Exact
dvdplayermp3player_Recpricemessage_service.wsdl	72,72	87,01	80,51	70,00	61,00	73,00	Plugin
dvdplayermp3player_RecpriceEuro_service.wsdl	58,44	87,01	51,94	88,00	86,00	92,00	Subsumes
dvdplayermp3player_Recprice_service.wsdl	58,44	87,01	80,51	93,00	94,00	96,00	Plugin
portabledvdplayermp3player_recommendedpricetaxedprice_service.w:	53,24	74,02	67,53	90,00	79,00	92,00	Failed
mp3playerdvdplayer_recommendedprice_service.wsdl	45,45	74,02	67,53	91,00	91,00	91,00	Plugin
mp3playerdvdplayer_Recpriceshipping_service.wsdl	45,45	74,02	67,53	76,00	67,00	79,00	Plugin
mp3playerdvdplayer_Recpriceshipping_USservice.wsdl	45,45	74,02	45,45	75,00	66,00	77,00	Subsumes
mp3playerportabledvdplayer_recommendedpricequality_service.wsdl	45,45	74,02	67,53	88,00	83,00	89,00	Failed

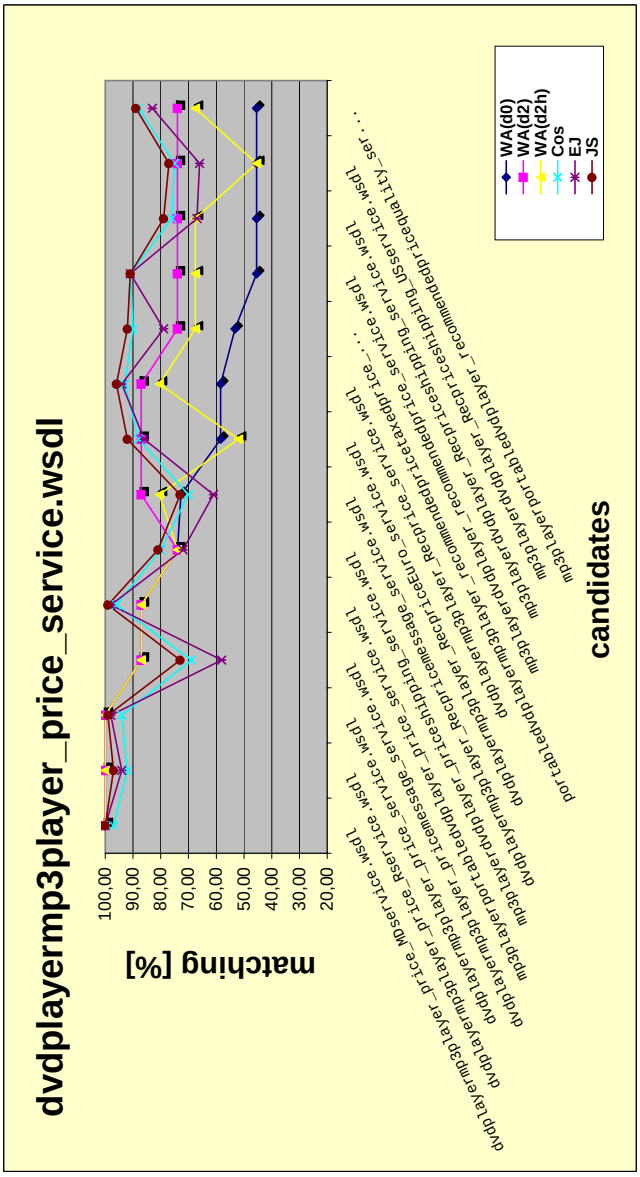


Abbildung 8.9: Query 03, requirement: economy/dvdplayermp3player_price_serice.wsdl

Der Vergleich der Resultate beider Matchmaker für diesen *Query* zeigt, dass die Ergebnisse des *WSDL Analyzer* in etwa den des OWLS-MX entsprechen. Auch hier entspricht das Ergebnis für WA(d2) am ehesten den Werten für Cos, EJ und JS. Bei WA(d2h) fallen zwei Ausreißer auf:

- dvdplayermp3player_RecpriceEuro_service.wsdl
- mp3playerdvdplayer_Recpriceshipping_USservice.wsdl

Query 04: economy/1personbicyclecar_price_service.wsdl

Dieser *Query* kann nicht komplett übersetzt werden, da einige XML Schema keine gültige XML Bezeichnung (XML NCName) haben. Weiterhin finden sich bei Übersetzung der gültigen WSDL Beschreibungen wieder ein Zyklus ab `TimeInterval`.

Es wurde ein alternatives Matchmaking WA(d0) durchgeführt:
Requirement ist `carcycle_price_service.wsdl`.

Candidate service	**WA(d0)**
carcycle_price_service.wsdl	100,00
cyclecar_pricetaxedprice_service.wsdl	72,22
bicyclecar_priceyear_service.wsdl	65,27
cyclecar_recommendedpriceineuro_service.wsdl	41,66
carbicycle_recommendedprice_service.wsdl	34,72

Query 05: economy/bookpersoncreditcardaccount_price_service.wsdl

Candidate service	WA(d0)	WA(d2)	WA(d2h)	Cos	EJ	JS	Semantic degree
bookpersoncreditcardaccount_price_service.wsdl	100,00	100,00	100,00	97,00	100,00	100,00	Exact
bookpersonOptional_price_service.wsdl	87,51	90,80	85,58	89,00	93,00	96,00	Exact
bookperson_price_service.wsdl	87,51	90,80	85,58	89,00	93,00	96,00	Exact
BookPrice.wsdl	83,34	87,73	80,77	89,00	93,00	96,00	Exact
book_Cheapestprice_service.wsdl	83,34	87,73	80,77	89,00	93,00	96,00	Exact
book_price_service.wsdl	83,34	87,73	80,77	89,00	93,00	96,00	Exact
book_pricereviewbook_service.wsdl	75,02	81,60	71,16	62,00	54,00	66,00	Exact
book_pricesizebook-type_service.wsdl	75,02	81,60	71,16	83,00	81,00	89,00	Exact
book_reviewprice_service.wsdl	75,02	81,60	71,16	86,00	86,00	92,00	Exact
book_taxedpriceprice_service.wsdl	75,02	81,60	71,16	86,00	78,00	92,00	Exact
bookpersoncreditcardaccount_recommendedprice_service.wsdl	73,31	93,86	85,53	94,00	94,00	96,00	Plugin
bookpersoncreditcardaccount_taxedfreeprice_service.wsdl	64,98	87,73	75,92	94,00	94,00	96,00	Plugin
book_recommendedpriceindollar_service.wsdl	56,66	81,60	45,17	83,00	81,00	89,00	Subsumes
book_recommendedprice_RegisteredUserservice.wsdl	56,66	81,60	66,31	86,00	86,00	92,00	Plugin
book_recommendedprice_service.wsdl	56,66	81,60	66,31	86,00	86,00	92,00	Plugin
Tizonbook_recommendedpriceindollar_service.wsdl	56,66	81,60	45,17	83,00	81,00	89,00	Subsumes
printedmaterialpersoncreditcardaccount_price_service.wsdl	55,82	41,10	64,44	82,00	68,00	79,00	Plugin
monograph_price_service.wsdl	52,49	55,21	80,77	79,00	75,00	85,00	Plugin
printedmaterial_price_service.wsdl	47,50	34,97	54,82	61,00	57,00	66,00	Plugin
monographpersoncreditcardaccount_recommendedprice_service.wsd	34,13	55,21	75,92	87,00	78,00	87,00	Plugin
monograph_recommendedpriceineuro_service.wsdl	25,81	49,08	45,17	73,00	64,00	78,00	Subsumes

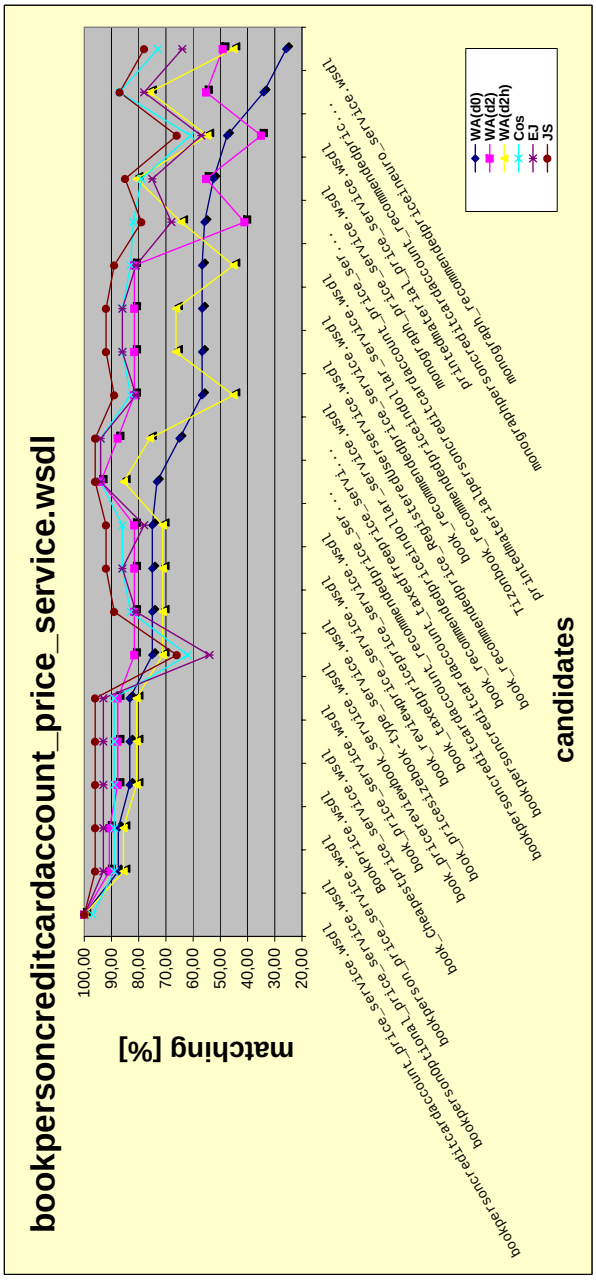


Abbildung 8.10: Query 05, req.: economy/bookpersoncreditcardaccount_price_service.wsdl

Wieder ist das WA(d2) Matchmaking nahe an den Ergebnissen des OWLS-MX. Hier fällt aber auch das Matchmaking von WA(d2h) Service-Beschreibungen aufgrund guter Werte auf.

Query 06: economy/maxprice_cola_service.wsdl

Candidate service	WA(d0)	WA(d2)	WA(d2h)	Cos	EJ	JS	Semantic degree
maxprice_cola_Bestservice.wsdl	100,00	100,00	100,00	99,00	100,00	100,00	Exact
maxprice_cola_service.wsdl	100,00	100,00	100,00	99,00	100,00	100,00	Exact
maxprice_beercola_service.wsdl	77,77	85,08	85,07	92,00	86,00	94,00	Exact
maxprice_colabeer_service.wsdl	77,77	85,08	85,07	89,00	80,00	90,00	Exact
maxprice_colabreadorbiscuit_Bothservice.wsdl	77,77	85,08	85,07	77,00	65,00	75,00	Exact
maxprice_drinks_service.wsdl	77,77	85,08	85,07	86,00	83,00	90,00	Subsumed-by
maxprice_whiskeycolabeer_service.wsdl	77,77	85,08	85,07	89,00	75,00	89,00	Plugin
price_cola_Gudduservice.wsdl	77,77	85,08	77,61	93,00	94,00	96,00	Plugin
price_cola_Hallo2service.wsdl	77,77	85,08	77,61	93,00	94,00	96,00	Plugin
price_cola_Halloservice.wsdl	77,77	85,08	85,07	94,00	90,00	94,00	Failed
qualitymaxprice_cola_service.wsdl	77,77	85,08	85,07	94,00	90,00	94,00	Failed
taxfreeprice_cola_service.wsdl	77,77	85,08	85,07	91,00	89,00	93,00	Failed

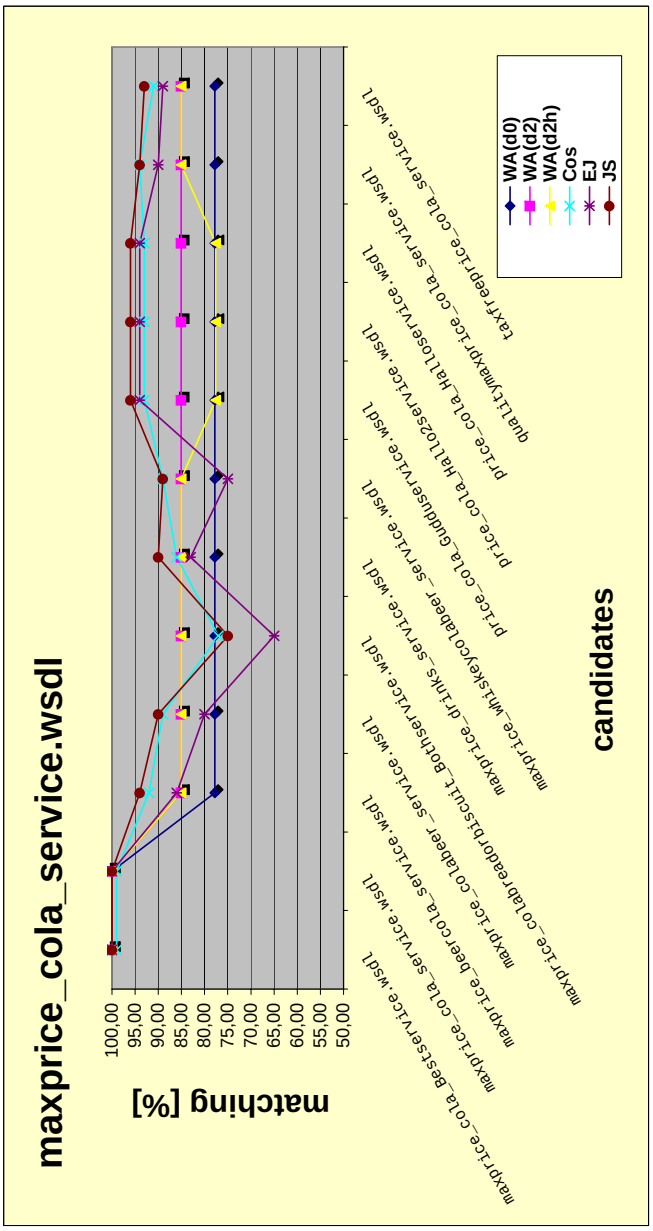


Abbildung 8.11: Query 06, req.: economy/maxprice_cola_service.wsdl

Es fällt auf, dass das Matchmaking für WA(d2h) bei drei Service-Beschreibungen schlechter wird:

- price_cola_Gudduservice.wsdl
- price_cola_Hallo2service.wsdl
- price_cola_Halloservice.wsdl

Query 07: economy/bookpersoncreditcardaccount__service.wsdl

Candidate service	WA(d0)	WA(d2)	WA(d2h)	Cos	EJ	JS	Semantic degree
bookpersoncreditcardaccount__BShopservice.wsdl	100,00	100,00	100,00	95,00	100,00	100,00	Exact
bookpersoncreditcardaccount__service.wsdl	100,00	100,00	100,00	95,00	100,00	100,00	Exact
bookpersoncreditaccount__Beaservice.wsdl	87,18	91,73	83,87	90,00	97,00	98,00	Plugin
bookpersoncreditaccount__service.wsdl	87,18	91,73	83,87	90,00	97,00	98,00	Plugin
bookusercreditcardaccount__service.wsdl	87,18	91,73	83,87	93,00	97,00	98,00	Failed
personbookliabilityaccount__service.wsdl	87,18	91,73	83,87	86,00	93,00	96,00	Plugin
bookperson__service.wsdl	80,78	87,60	75,80	78,00	86,00	92,00	Exact
publicationperson__service.wsdl	40,99	38,82	67,74	50,00	44,00	63,00	Plugin
personmonographcreditcardaccount__service.wsdl	39,71	47,91	83,87	80,00	68,00	81,00	Plugin
monographperson__service.wsdl	33,31	43,78	75,80	59,00	54,00	71,00	Plugin
printedmaterialpersoncreditcardaccount__service.wsdl	32,03	20,65	40,32	63,00	36,00	58,00	Plugin
printedmaterialperson__service.wsdl	25,62	16,52	32,25	31,00	22,00	43,00	Plugin

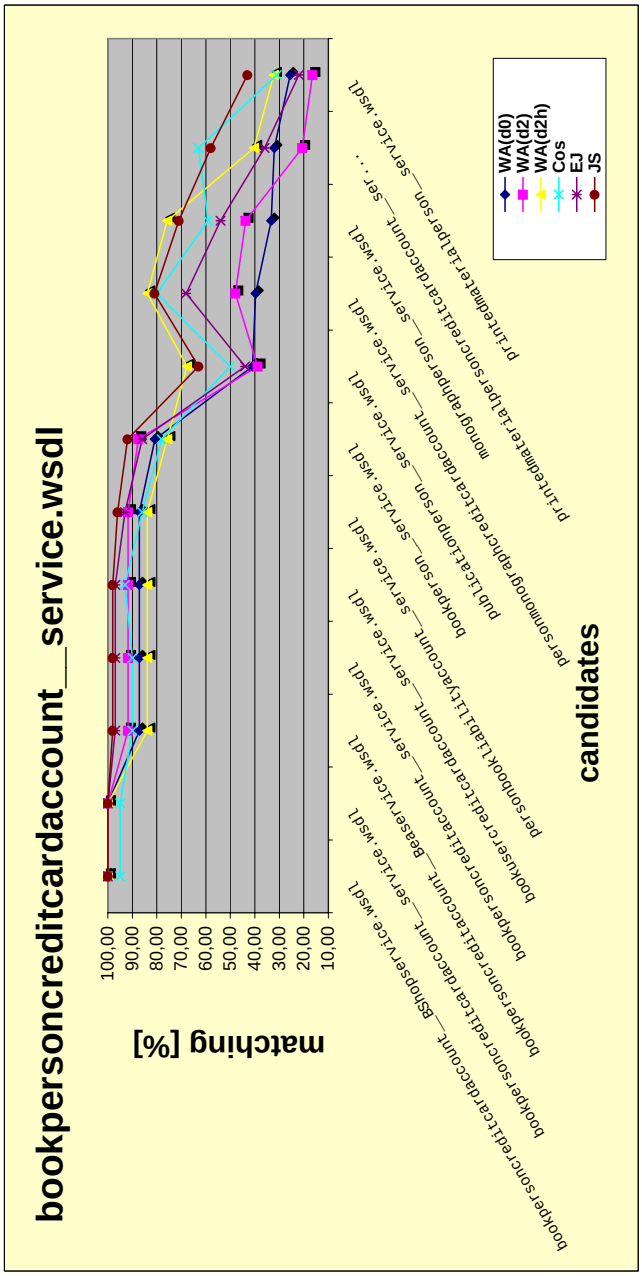


Abbildung 8.12: Query 07, requirement: economy/bookpersoncreditcardaccount__service.wsdl

Bemerkung: Die Matchmaking-Ergebnisse sind sich sehr ähnlich.

Query 08: economy/shoppingmall_cameraprice_service.wsdl

Candidate service	WA(d0)	WA(d1)	WA(d1h)	Cos	EJ	JS	Semantic degree
shoppingmall_cameraprice_service.wsdl	100,00	100,00	100,00	91,00	100,00	100,00	Exact
shoppingmall_calendar-datepricecamera_service.wsdl			96,69	70,00	64,00	76,00	Exact
_pricecamera_Wallmartservice.wsdl	90,09	90,09	90,09	45,00	50,00	50,00	Exact
PhillipDigCamera_price_service.wsdl	15,51	15,51	15,51	72,00	87,00	92,00	Fail
shoppingmall_pricecellphonewithcamera_service.wsdl	41,58	41,58	41,58	69,00	66,00	76,00	Failed
KodakDigCamera_price_service.wsdl	12,21	12,21	12,21	29,00	39,00	43,00	Failed
_price_CannonCameraservice.wsdl	12,21	12,21	12,21	29,00	39,00	43,00	Failed
shoppingmall_analogpricecalendar-date_service.wsdl			95,04	62,00	59,00	70,00	Plugin
shoppingmall_pricedigitalanalog_service.wsdl			95,04	69,00	61,00	77,00	Plugin
_digitalstandardpriceprice_MediaMarktservice.wsdl			88,44	27,00	22,00	35,00	Plugin
shoppingmall_purchaseableitemprice_service.wsdl	20,46	20,46	20,46	82,00	94,00	97,00	Subsumed-by
shoppingmall_digital-slrpricecalendar-date_service.wsdl			101,98	59,00	60,00	70,00	Subsumes
shoppingmall_maxpricedigital-video_service.wsdl	11,55	18,81	93,39	69,00	66,00	76,00	Subsumes
SRcamera_service.wsdl	89,43	89,43	89,43	86,00	91,00	95,00	Subsumes

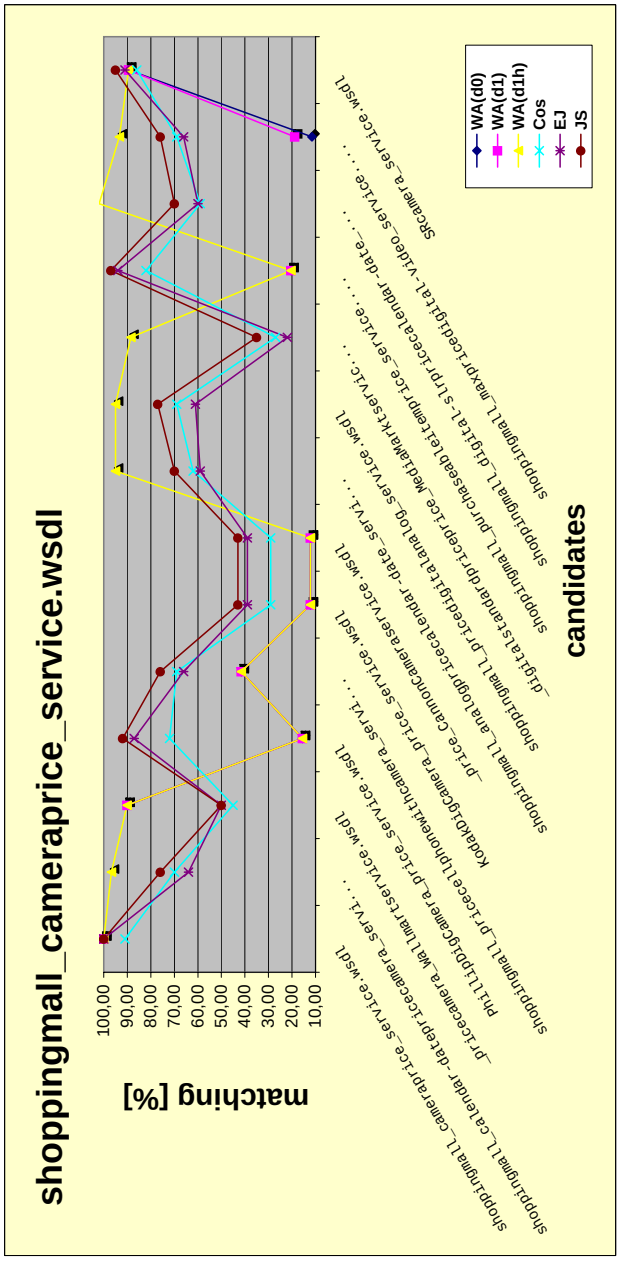


Abbildung 8.13: Query 08, requirement: economy/shoppingmall_cameraprice_service.wsdl

Bemerkung: Die Matchmaking-Ergebnisse sind sich ähnlich.

Query 09: economy/recommendedprice_coffeewhiskey_service.wsdl

Der Kandidat recommendedprice_contentbearingobjectwhiskeycoffee_service.wsdl kann bereits bei dem WA(d0) Set nicht von dem WA Parser geladen werden. Weiterhin fällt für WA(d2) der bereits erwähnte Zyklus in `ComplexType TimeInterval` auf.

Query 10: economy/userscience-fiction-novel_price_service.wsdl

Candidate service	WA(d0)	WA(d2)	WA(d2h)	Cos	EJ	JS	Semantic degree
DeoSFN_price_service.wsdl	100,00	100,00	100,00	98,00	100,00	100,00	Exact
userscience-fiction-novel_price_Bestservice.wsdl	100,00	100,00	100,00	98,00	100,00	100,00	Exact
userscience-fiction-novel_price_service.wsdl	100,00	100,00	100,00	98,00	100,00	100,00	Exact
userscience-fiction-novel_Relprice_service.wsdl	100,00	100,00	100,00	98,00	100,00	100,00	Exact
novel_price_service.wsdl	72,22	79,04	44,31	93,00	86,00	97,00	Plugin
ScienceFNovelReview_service.wsdl	70,37	94,01	91,01	92,00	89,00	93,00	Plugin
SFNRecommendedPrice_service.wsdl	70,37	94,01	91,01	95,00	94,00	96,00	Plugin
novelperson_price_service.wsdl	67,59	76,04	41,31	92,00	86,00	97,00	Plugin
userbook_price_service.wsdl	67,59	70,05	41,31	92,00	85,00	96,00	Plugin
sciencefictionbookuser_price_service.wsdl	57,40	66,46	41,31	93,00	86,00	96,00	Plugin
science-fiction-noveluser_recommendedpriceindollar_service.wsdl	51,85	82,03	65,86	92,00	89,00	93,00	Subsumes
SFNovelReview_service.wsdl	51,85	82,03	68,86	90,00	85,00	90,00	

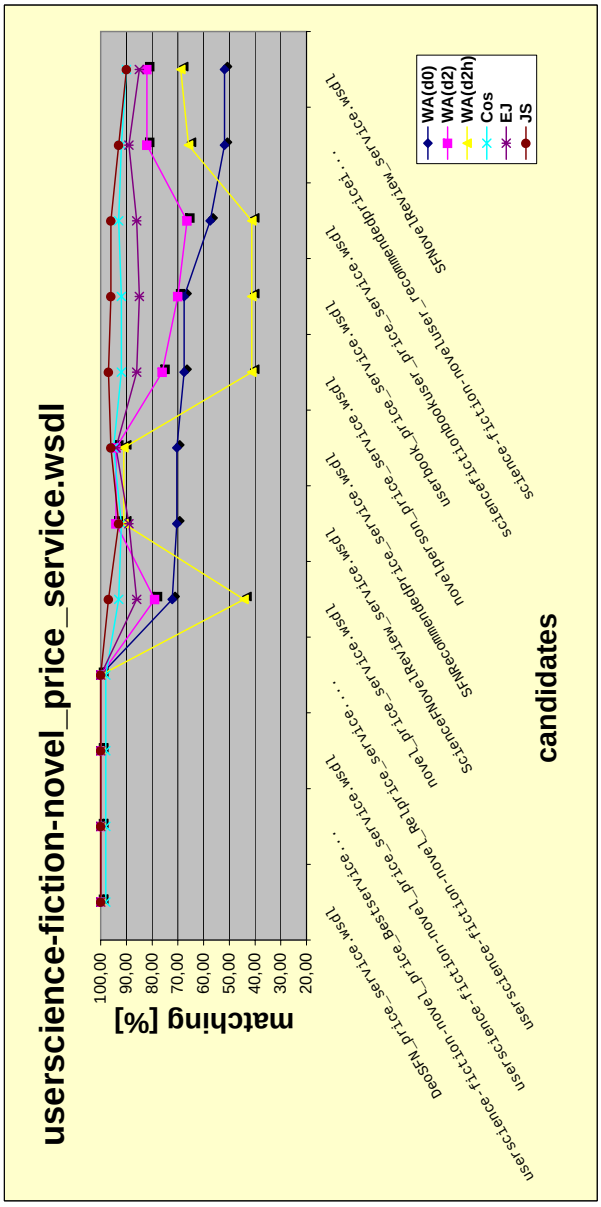


Abbildung 8.14: Query 10, requirement: economy/userscience-fiction-novel_price_service.wsdl

Die WA(d2) Werte sind relativ gut mit den OWLS-MX Werten vergleichbar. Die WA(d2h) Reihe zeigt teilweise eine gute Annäherung.

Query 11: economy/economy-preparedfood_price_service.wsdl

Problem (Parser): Für d0 lässt sich agent_price_MianMarktservice.wsdl nicht einlesen.
Ab WA(d2) treten Zyklen auf: z.B. für `TimeInterval` und `Agent`

Candidate service	WA(d0)
preparedfood_price_service.wsdl	100
food_price_service.wsdl	85,07
MarkoPS_service.wsdl	85,07
MerkelD_service.wsdl	85,07
preparedfood_priceday_service.wsdl	85,07
preparedfood_taxedpriceindollarprice_service.wsdl	85,07
Ben_service.wsdl	52,23
MAK_service.wsdl	52,23
preparedfood_GSprice_service.wsdl	52,23
preparedfood_taxfreeprice_service.wsdl	52,23
ZAD_service.wsdl	52,23
breadorbiscuit_recPricetaxedpriceineuro_service.wsdl	37,31
food_maxpricequantity_Aldiservice.wsdl	37,31
food_recommendedprice_service.wsdl	37,31

Abbildung 8.15: Query 11, requirement: economy/economy-preparedfood_price_service.wsdl

Query 12: food/grocerystore_food_service.wsdl

Problem (Parser): Ab WA(d1) treten Zyklen auf: `TimeInterval`

Candidate service	WA(d0)
grocerystore_food_service.wsdl	100
Available_preparedfoodquantity_service.wsdl	77,77
grocerystore_flourdoughbutter_service.wsdl	77,77
grocerystore_foodquantity_service.wsdl	77,77
grocerystore_preparedfoodprice_service.wsdl	77,77
grocerystore_preparedfood_service.wsdl	77,77
grocerystore_teaprice_service.wsdl	77,77
mercantileorganization_food_service.wsdl	77,77
drugstore_tea_service.wsdl	55,55
retailstore_apple_service.wsdl	55,55
retailstore_preparedfood_service.wsdl	55,55
store_preparedfood_Merchantservice.wsdl	55,55
store_preparedfood_service.wsdl	55,55
_food_HEBgroceryCompservice.wsdl	55,55
_cerealgrain_PlantSeedservice.wsdl	33,33
_honey_Providerservice.wsdl	33,33

Abbildung 8.16: Query 12, requirement: food/grocerystore_food_service.wsdl

Query 13: communication/title_comedyfilm_service.wsdl

Candidate service	WA(d0)	WA(d2)	WA(d2h)	Cos	EJ	JS	Semantic degree
title_comedyfilm_BFservice.wsdl	100,00	100,00	100,00	100,00	100,00	100,00	Exact
title_comedyfilm_Megaservice.wsdl	100,00	100,00	100,00	100,00	100,00	100,00	Exact
title_comedyfilm_service.wsdl	100,00	100,00	100,00	100,00	100,00	100,00	Exact
title_actionfilm_service.wsdl	77,77	80,39	80,39	84,00	80,00	88,00	Failed
title_filmpricequality_service.wsdl	77,77	80,39	70,58	70,00	62,00	71,00	Subsumed-by
title_highcomedyfilmreport_service.wsdl	77,77			64,00	54,00	63,00	Plugin
title_lowcomedyfilm_service.wsdl	77,77	80,39	80,39	94,00	90,00	95,00	Plugin
title_media_service.wsdl	77,77	58,82	58,82	71,00	63,00	73,00	Failed
title_filmActionComedy_service.wsdl	66,66	80,39	70,58	90,00	88,00	93,00	Subsumed-by
title_filmP2P_service.wsdl	66,66	80,39	70,58	90,00	88,00	93,00	Subsumed-by
title_film_service.wsdl	66,66	80,39	70,58	90,00	88,00	93,00	Subsumed-by
_comedyfilmactionfilm_service.wsdl	33,33	41,17	41,17	45,00	36,00	46,00	Exact
_comedyfilmfantacyfilm_service.wsdl	33,33	41,17	41,17	45,00	36,00	46,00	Exact
_filmHighlyRated_service.wsdl	22,22	41,17	31,37	40,00	38,00	43,00	Subsumed-by

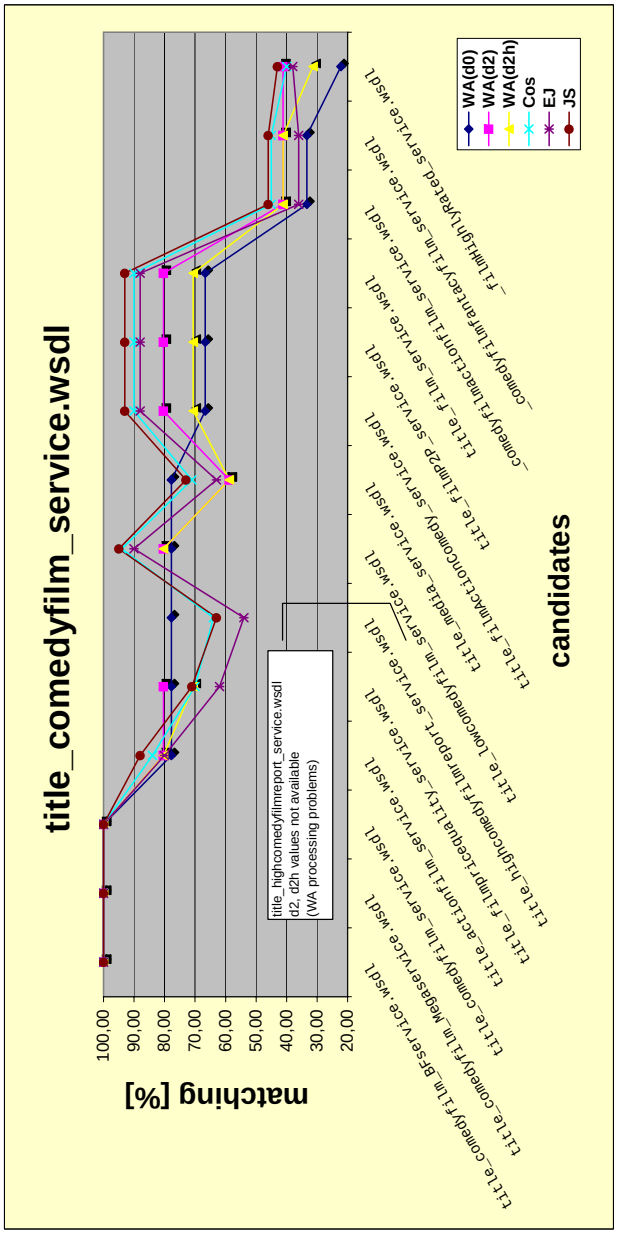


Abbildung 8.17: Query 13, requirement: communication/title_comedyfilm_service.wsdl

Query 14: communication/title_videomedia_service.wsdl

Candidate service	WA(d0)
title_videomediaMM_service.wsdl	100
title_videomedia_service.wsdl	100
linguisticexpression_videomedia_service.wsdl	77,77
title_media_service.wsdl	77,77
title_obtainablevideomedia_service.wsdl	77,77
title_vhsdvd_service.wsdl	77,77
title_vhs_service.wsdl	77,77
title_videomediarecommendedprice_service.wsdl	77,77
_videomediaBBC_service.wsdl	55,55
_videomediaSaturn_service.wsdl	55,55
_videomediaSmithLee_service.wsdl	55,55
_filmvideomediaDiscoveryChannel_service.wsdl	33,33

Abbildung 8.18: Query 14, requirement: communication/title_videomedia_service.wsdl

Query 15: education/researcher-in-academia_address_service.wsdl

Parsen und Darstellung der WSDL Beschreibungen mit dem WA ist möglich.
Ein Matchmaking des *Requirement* ist jedoch nicht möglich. Matchmakings mit anderen WSDL Beschreibungen als Requirement können durchgeführt werden.
Fehler: matcher.WsdlServiceMatcher.matchMixedTypes()

Query 16: education/universiy_lecturer-in-academia_service.wsdl

Candidate service	WA(d0)	WA(d2h)	Cos	EJ	JS	Semantic degree
university_lecturer-in-academiaCurrentSemmester_service.wsdl	100,00	100,00	98,00	100,00	100,00	Exact
university_lecturer-in-academia_Recommendservice.wsdl	100,00	100,00	98,00	100,00	100,00	Exact
university_lecturer-in-academia_service.wsdl	100,00	100,00	98,00	100,00	100,00	Exact
university_senior-lecturer-in-academia_service.wsdl	100,00	100,00	98,00	99,00	100,00	Plugin
university_professor-in-academia_service.wsdl	89,69	80,00	98,00	99,00	100,00	Failed
university_research-fellow-in-academia_service.wsdl	89,69	80,00	97,00	98,00	99,00	Failed
university_researcher_service.wsdl	79,38	60,00	77,00	60,00	83,00	Failed
university_academic-support-staff_service.wsdl	74,22	60,00	97,00	99,00	99,00	Failed
organization_lecturer-in-academia_service.wsdl	69,07	70,00	95,00	97,00	98,00	Plugin
educational-organization_lecturer-in-academia_service.wsdl	46,39	70,00	96,00	98,00	99,00	Plugin
higher-educational-organization_lecturer-in-academia_MostUsedservice.wsdl	46,39	70,00	97,00	99,00	99,00	Plugin
higher-educational-organization_lecturer-in-academia_service.wsdl	46,39	70,00	97,00	99,00	99,00	Plugin
learning-centred-organization_lecturer-in-academia_service.wsdl	46,39	70,00	95,00	98,00	98,00	Plugin
higher-educational-organization_professor-in-academia_Pioneerservice.wsdl	36,08	50,00	96,00	98,00	99,00	Failed
higher-educational-organization_professor-in-academia_service.wsdl	36,08	50,00	96,00	98,00	99,00	Failed
higher-educational-organization_senior-research-fellow-in-academia_Firstservice.wsd	36,08	50,00	96,00	98,00	99,00	Failed
higher-educational-organization_senior-research-fellow-in-academia_service.wsdl	36,08	50,00	96,00	98,00	99,00	Failed
_lecturer-in-academiaMuenchenUniversity_service.wsdl	36,08	60,00	50,00	50,00	50,00	Exact
_lecturer-in-academiaSaarlandUniversity_service.wsdl	36,08	60,00	50,00	50,00	50,00	Exact
_lecturer-in-academiaZambiaUniversity_service.wsdl	36,08	60,00	50,00	50,00	50,00	Exact

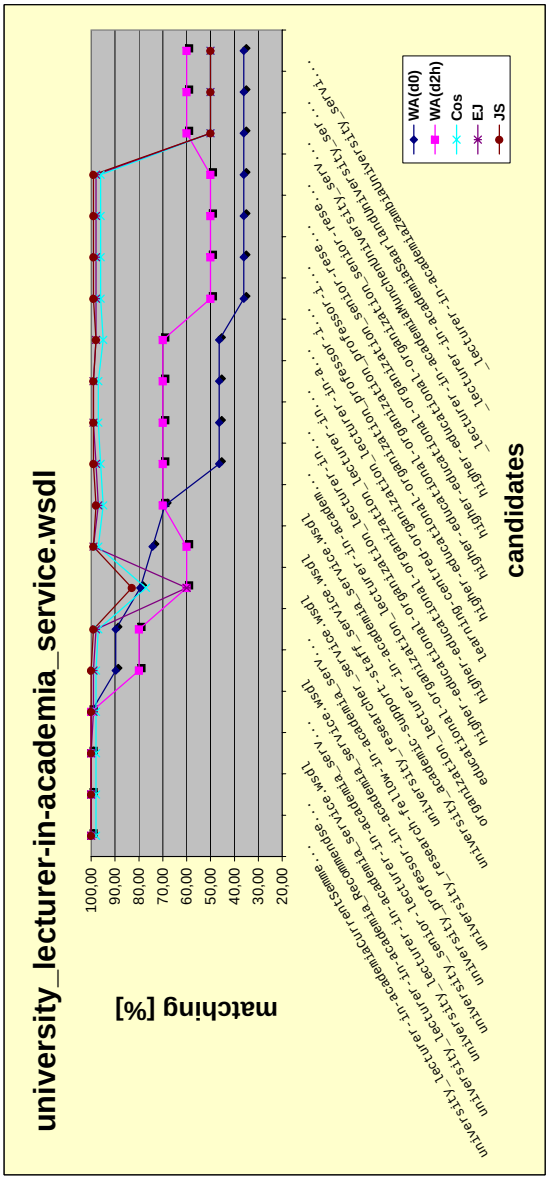


Abbildung 8.19: Query 16, requirement: education/universiy_lecturer-in-academia_service.wsdl

WA(d2) wird zwar geparst, kann aber nicht gematcht werden. Wird zusätzlich bei der Vererbungstiefe von 2 beim Generieren von XML Schema das *Hierarchy Pattern* angewandt funktioniert das Matching!

Query 17: education/governmentdegree_scholarship_service.wsdl

Candidate service	WA(d0)	Cos	EJ	JS	Semantic degree
governmentdegree_scholarship_service.wsdl	100,00	94,00	100,00	100,00	Exact
governmentdegree_scholarship_TheBestservice.wsdl	100,00	94,00	100,00	100,00	Exact
governmentdegree_scholarshipquantity_service.wsdl	80,00	88,00	88,00	93,00	Exact
government_scholarship_service.wsdl	80,00	86,00	93,00	95,00	Exact
government_scholarshiporganization_service.wsdl		69,00	64,00	80,00	Exact
degreegovernmentorganization_scholarship_service.wsdl	80,00	92,00	98,00	99,00	Plugin
governmentaward_scholarship_service.wsdl	80,00	91,00	98,00	99,00	Plugin
award_scholarship_GermanGovservice.wsdl	70,00	70,00	56,00	65,00	Plugin
award_scholarshipduration_SwissGovservice.wsdl	50,00	54,00	27,00	48,00	Plugin
governmentorganization_scholarship_service.wsdl	70,00	84,00	91,00	94,00	Plugin
degreegovernment_funding_service.wsdl	60,00	88,00	94,00	97,00	Subsumed-by
government_funding_ForPhDservice.wsdl	50,00	80,00	87,00	92,00	Subsumed-by
academic-degreegovernment_scholarship_service.wsdl	80,00	93,00	98,00	99,00	Failed
degreegovernment_unilateralgiving_service.wsdl	60,00	81,00	85,00	91,00	Failed
governmentdegree_welfare_service.wsdl	80,00	86,00	90,00	94,00	Failed
degreenationalgovernment_scholarship_service.wsdl	80,00	93,00	98,00	99,00	Failed
nationalgovernment_scholarship_service.wsdl	70,00	85,00	91,00	94,00	Failed
nationalgovernment_scholarshipquantity_service.wsdl	50,00	78,00	79,00	87,00	Failed
government_welfare_service.wsdl	50,00	77,00	83,00	90,00	Failed
nationalgovernment_scholarshipquantityduration_service.wsdl	50,00	65,00	59,00	75,00	Failed
nationalgovernment_physical-quantityscholarshiplandarea_service.wsdl	50,00	72,00	68,00	81,00	Failed

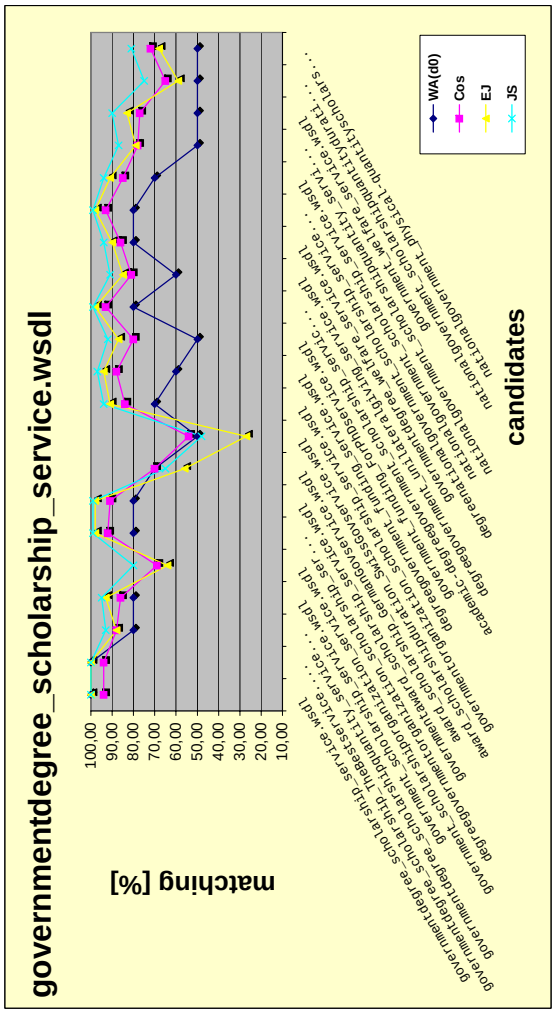


Abbildung 8.20: Query 17: education/governmentdegree_scholarship_service.wsdl

Query 18: education/publication-number_publication.wsdl

Alle generierten WSDL Beschreibungen WA(d0) - WA(d2h) werden eingelesen und dargestellt. Für WA(d0) kann der Matcher keine gültigen Werte bestimmen. WA(d2) kann vom Matcher nicht verarbeitet werden. Für WA(d2h) liefert der *WSDL Analyzer* Resultate.

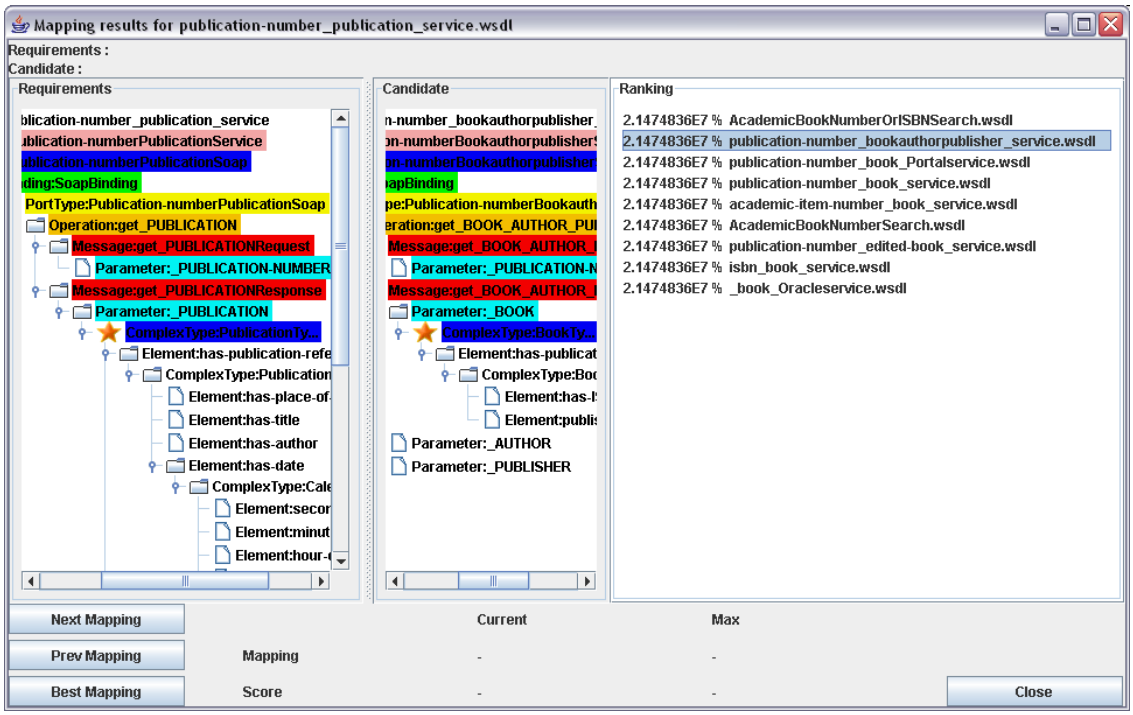


Abbildung 8.21: Query 18, education/publication-number_publication.wsdl, Fehler WA(d0)

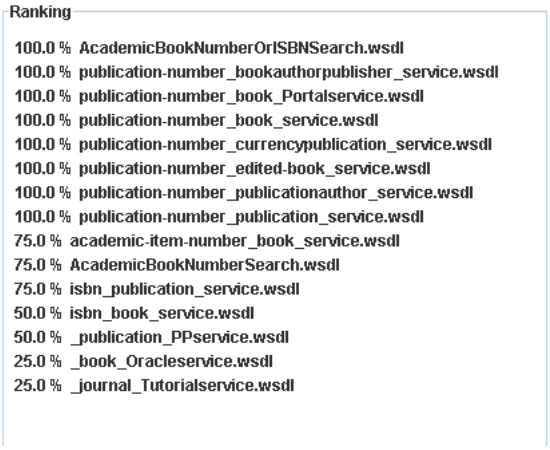


Abbildung 8.22: Query 18, education/publication-number_publication.wsdl, WA(d2h)

Query 19: education/novel_author_service.wsdl

Die WSDL Beschreibungen der Konfiguration d0 werden eingelesen. Bei den Konfigurationen d2 und d2h kann die Datei book_authortext_service.wsdl nicht eingelesen werden.
Zyklus: `SelfConnectedObject`

Candidate service	WA(d0)	WA(d2)	WA(d2h)	Cos	EJ	JS	Semantic degree
novel_author_BookOntoservice.wsdl	100,00	100,00	100,00	98,00	100,00	100,00	Exact
novel_author_MyOntoservice.wsdl	100,00	100,00	100,00	96,00	92,00	98,00	Exact
novel_author_service.wsdl	100,00	100,00	100,00	98,00	100,00	100,00	Exact
novel_authorcommitting_service.wsdl	83,60	91,66	86,11	71,00	60,00	70,00	Exact
novel_authorgenre_service.wsdl	83,60	91,66	86,11	89,00	83,00	90,00	Exact
novel_authortime_service.wsdl	83,60	91,66	86,11	89,00	83,00	90,00	Exact
novel_person_Writerservice.wsdl	83,60	91,66	86,11	79,00	75,00	84,00	Subsumed-by
novel_userreviewauthor_service.wsdl	83,60	91,66	86,11	84,00	75,00	84,00	Exact
book_author_service.wsdl	75,40	83,33	86,11	94,00	94,00	97,00	Plugin
publication_author_service.wsdl	67,21	43,33	79,16	77,00	70,00	81,00	Plugin
book_authorbook-type_service.wsdl	59,01	75,00	72,22	85,00	78,00	87,00	Plugin
book_authorprice_Novelservice.wsdl	59,01	75,00	72,22	68,00	56,00	70,00	Plugin
book_authortext_service.wsdl	59,01			60,00	49,00	61,00	Plugin

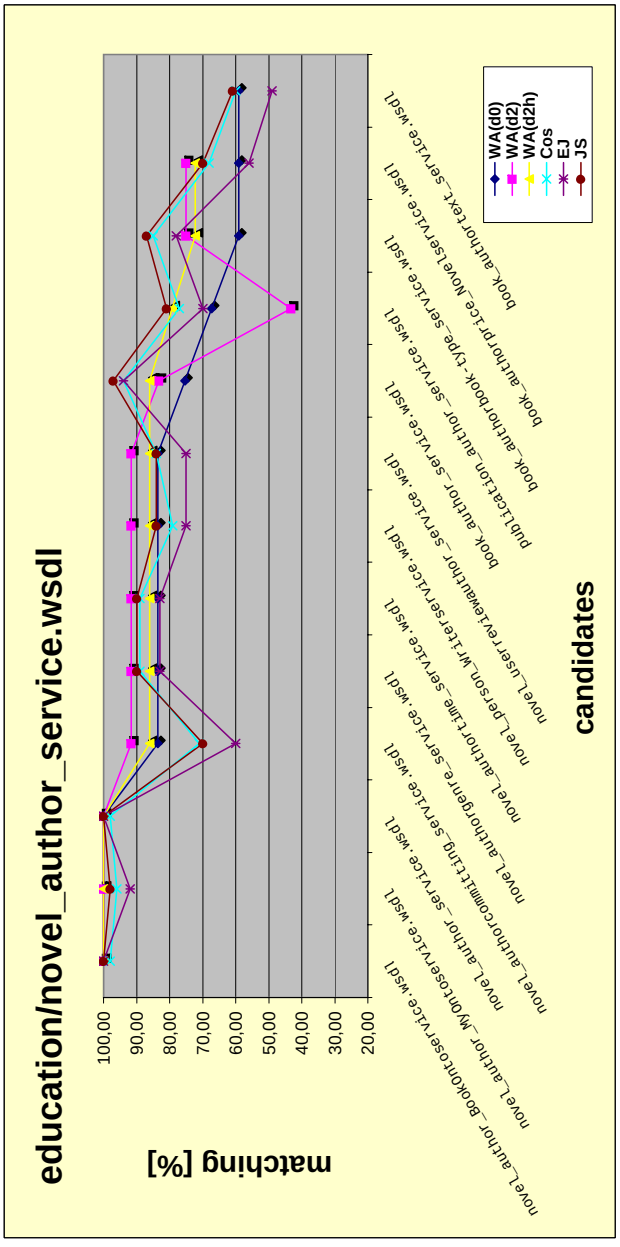


Abbildung 8.23: Query 19: education/novel_author_service.wsdl

WA(d2h) liefert gute Vergleichswerte zu den OWLS-MX Ergebnissen.

Query 20: education/country_skilledoccupation_service.wsd

Das Einlesen der WSDL Beschreibungen für Konfiguration d0 kann durchgeführt werden. Das WA-Matcher erzeugt jedoch keine Resultate.

Die Wahl eines anderes *Requirement* erzeugt ein *Ranking*.

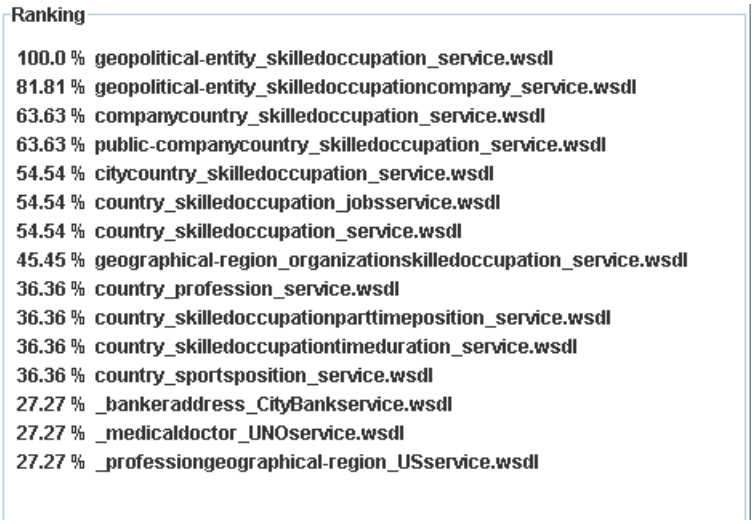


Abbildung 8.24: Query 20, geopolitical/entity_skilledoccupation_service.wsdl, WA(d0)

Für Konfiguration d2 und d2h kann jeweils die Datei `citycountry_skilledoccupation_service.wsdl` nicht eingelesen werden.

- Zyklus: `TimeInterval`

Query 21: medical/hospital_investigating_service.wsdl

Keine der generierten WSDL *Requirement Sets* lässt sich komplett einlesen, das bereits ab Konfiguration d0 folgende Zyklen auftauchen:

- `TimeInterval`
- `Human`

Query 22: travel/surfing_destination_service.wsdl

Die Übersetzung der benötigen Ontologien für das *Relevance Set* führte zu 15 anonymen Typen in der Datatype Knowledgebase. Klassen wie `travel.owl#FamilyDestination` liefern auch ein sehr interessantes Übersetzungsergebnis. Allerdings sind die Parameter der untersuchten Service Schnittstellen nicht von einem dieser Typen.

Candidate service	all WA sets	Cos	EJ	JS	Semantic degree
surfing_destination_Alwaysservice.wsdl	100,00	96,00	70,00	96,00	Exact
surfing_destination_AUSservice.wsdl	100,00	96,00	72,00	96,00	Exact
surfing_destination_service.wsdl	100,00	96,00	84,00	96,00	Exact
surfing_destination_SOHservice.wsdl	100,00	91,00	90,00	92,00	Exact
sports_destination_service.wsdl	77,77	83,00	81,00	84,00	Plugin
surfing_beach_service.wsdl	77,77	78,00	59,00	77,00	Plugin
surfing_farmland_service.wsdl	77,77	77,00	38,00	73,00	Subsumes
surfing_ruralarea_service.wsdl	77,77	75,00	54,00	72,00	Plugin
surfing_nationalpark_service.wsdl	66,66	50,00	37,00	45,00	Subsumes
sports_beach_service.wsdl	55,55	70,00	54,00	69,00	Plugin
sports_ruralarea_service.wsdl	55,55	78,00	62,00	74,00	Plugin
sports_town_service.wsdl	55,55	72,00	33,00	71,00	Subsumes
sports_nationalpark_service.wsdl	44,44	55,00	34,00	51,00	Subsumes

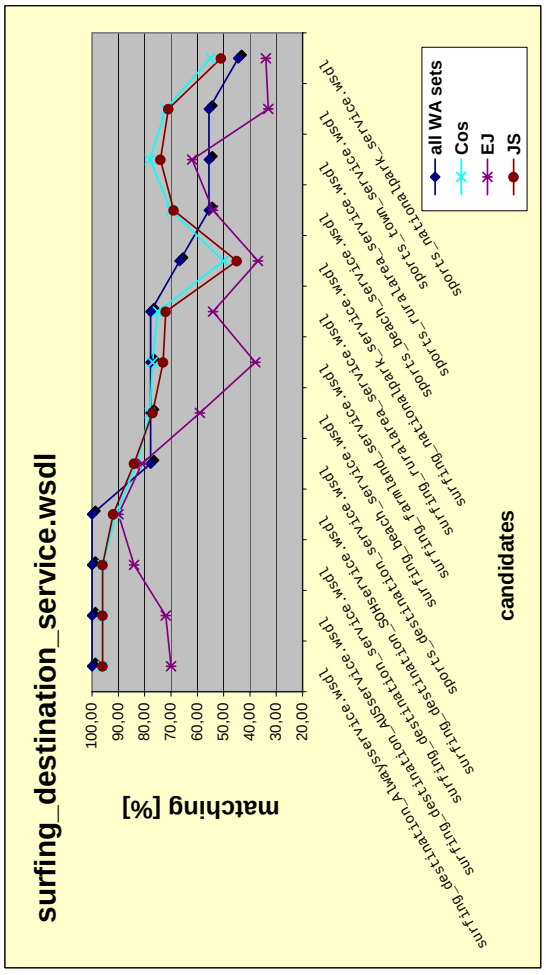
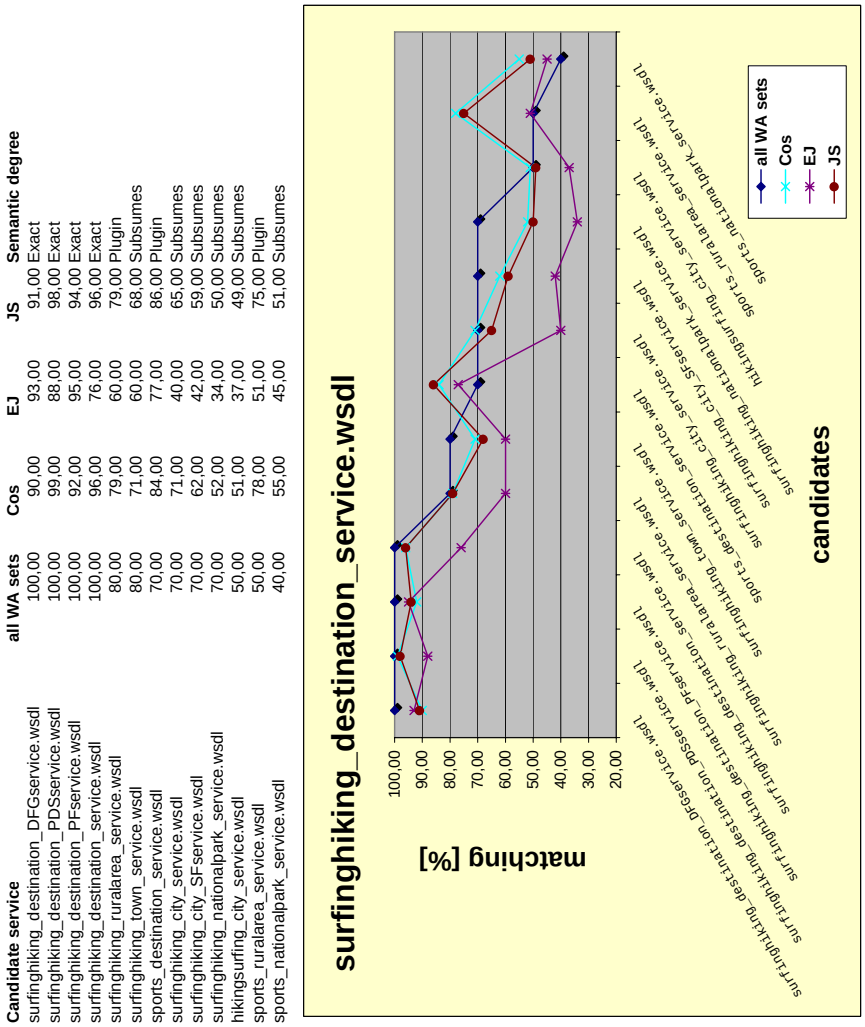


Abbildung 8.25: Query 22: travel/surfing_destination_service.wsdl

Eine Variation der Vererbungstiefe führt zu keinen Unterschieden.

Query 23: travel/surfinghiking_destination_service.wsdl



Candidate service	all WA sets	Cos	EJ	JS	Semantic degree
surfinghiking_destination_DFGservice.wsdl	100,00	90,00	93,00	91,00	Exact
surfinghiking_destination_PDSservice.wsdl	100,00	99,00	88,00	98,00	Exact
surfinghiking_destination_PFservice.wsdl	100,00	92,00	95,00	94,00	Exact
surfinghiking_destination_service.wsdl	100,00	96,00	76,00	96,00	Exact
surfinghiking_ruralarea_service.wsdl	80,00	79,00	60,00	79,00	Plugin
surfinghiking_town_service.wsdl	80,00	71,00	60,00	68,00	Subsumes
sports_destination_service.wsdl	70,00	84,00	77,00	86,00	Plugin
surfinghiking_city_service.wsdl	70,00	71,00	40,00	65,00	Subsumes
surfinghiking_city_SFservice.wsdl	70,00	62,00	42,00	59,00	Subsumes
surfinghiking_nationalpark_service.wsdl	70,00	52,00	34,00	50,00	Subsumes
hikingsurfing_city_service.wsdl	50,00	51,00	37,00	49,00	Subsumes
sports_ruralarea_service.wsdl	50,00	78,00	51,00	75,00	Plugin
sports_nationalpark_service.wsdl	40,00	55,00	45,00	51,00	Subsumes

Abbildung 8.26: Query 23: travel/surfinghiking_destination_service.wsdl

Die Ergebnisse des WA für unterschiedliche OWLS2WSDL Konfigurationen weichen nicht voneinander ab, sind aber relativ gut mit den Ergebnissen des OWLS-MX vergleichbar.

Query 24: travel/geographical-regiongeographical-region_map_service.wsdl

Ab Konfiguration d1 konnen nicht alle WSDL Beschreibungen gelesen werden.

- Nicht lesbar: `geographical-regiongeographical-region_icon_service.wsdl`
- Nicht lesbar: `locationlocation_icon_service.wsdl`
- Zyklus ab `TimeInterval`

Candidate service	WA(d0)	WA(d1h)	Cos	EJ	JS	Semantic degree
geographical-regiongeographical-region_map_Gorgservice.wsdl	100,00	100,00	97,00	100,00	100,00	Exact
geographical-regiongeographical-region_map_service.wsdl	100,00	100,00	97,00	100,00	100,00	Exact
geographical-regiongeographical-region_icon_service.wsdl	87,50		91,00	94,00	97,00	Subsumed-by
geographical-region_mapFromFrankfurt_service.wsdl	68,75	68,75	97,00	100,00	100,00	Exact
geographical-region_mapToBerlin_service.wsdl	68,75	68,75	97,00	100,00	100,00	Exact
locationlocation_map_service.wsdl	62,50	62,50	93,00	86,00	96,00	Plugin
locationlocation_map_SRIservice.wsdl	62,50	62,50	93,00	86,00	96,00	Plugin
citycity_map_service.wsdl	50,00	50,00	90,00	87,00	92,00	Failed
locationlocation_arrowfigure_service.wsdl	50,00	50,00	84,00	75,00	90,00	Failed
locationlocation_icon_service.wsdl	50,00		87,00	79,00	93,00	Subsumed-by
citycity_arrowfigure_service.wsdl	37,50	37,50	81,00	76,00	85,00	Failed
country_map_service.wsdl	37,50	37,50	92,00	89,00	93,00	Failed
_mapFrankfurtBerlin_service.wsdl	31,25	31,25	47,00	50,00	50,00	Exact
_mapGerman_service.wsdl	31,25	31,25	47,00	50,00	50,00	Exact

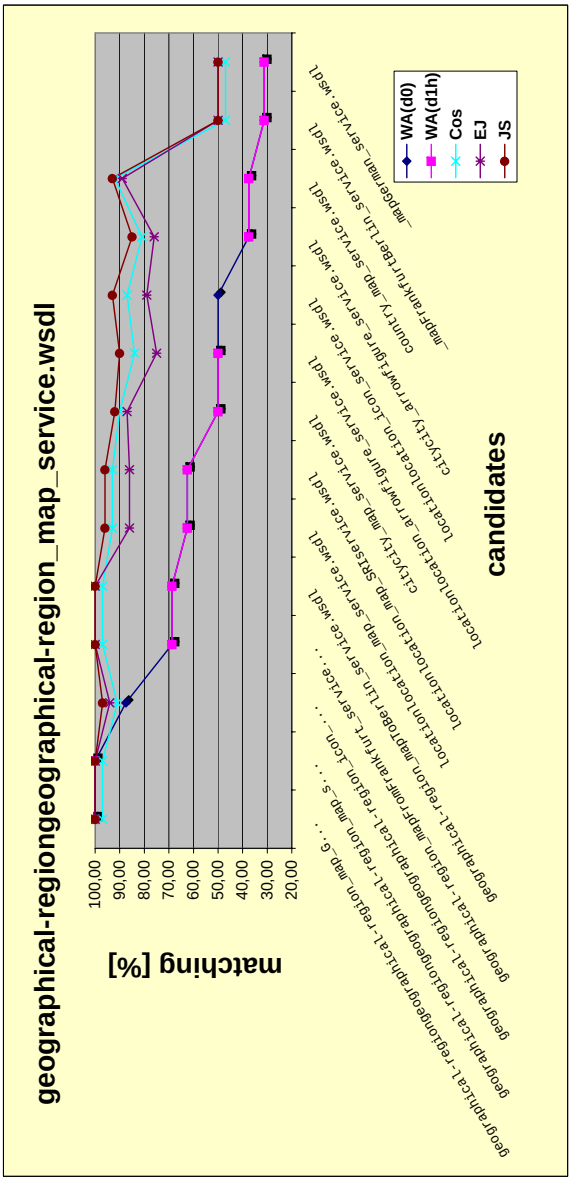


Abbildung 8.27: Query 24: travel/geographical-regiongeographical-region_map_service.wsdl

WA *Rankings* weichen stark von den Ergebnissen des OWLS-MX ab.

Query 25: travel/surfingorganization_destination_service.wsdl

Der WA liest alle WSDL Beschreibungen ein und zeigt diese auch an. Es tritt jedoch folgender Fehler beim Matchen auf:

```
[SimpleTypeLookupTable.java] getMatchingScore, a:string
[SimpleTypeLookupTable.java] getMatchingScore, b:SportsType
[SimpleTypeLookupTable 9110923] ta:string, tb:string  query xsdtypes: 18423897
[SimpleTypeLookupTable 9110923] ta:string, tb:string  query xsdtypes: 18423897
[SimpleTypeLookupTable] getScore tb: 10
Exception in thread "AWT-EventQueue-0" java.lang.StackOverflowError
at java.util.regex.Pattern.family(Unknown Source)
at java.util.regex.Pattern.range(Unknown Source)
at java.util.regex.Pattern.clazz(Unknown Source)
at java.util.regex.Pattern.sequence(Unknown Source)
at java.util.regex.Pattern.expr(Unknown Source)
at java.util.regex.Pattern.group0(Unknown Source)
at java.util.regex.Pattern.sequence(Unknown Source)
at java.util.regex.Pattern.expr(Unknown Source)
at java.util.regex.Pattern.compile(Unknown Source)
at java.util.regex.Pattern.<init>(Unknown Source)
at java.util.regex.Pattern.compile(Unknown Source)
at matcher.NameTokens.<init>(NameTokens.java:26)
at matcher.WsdlServiceMatcher.matchNames(WsdlServiceMatcher.java:2589)
at matcher.WsdlServiceMatcher.matchMixedTypes(WsdlServiceMatcher.java:1081)
at matcher.WsdlServiceMatcher.matchMixedTypes(WsdlServiceMatcher.java:1134)
...
```

Query 26: travel/citycountry_hotel_service.wsdl

Der WA liest alle WSDL Beschreibungen ein und zeigt diese auch an. Es tritt jedoch folgender Fehler beim Matchen auf:

```
[WsdlServiceMatcher] match elements: has-affiliation (Organization),
                                     has-affiliation (Organization)
[info] req elem has ComplexType
[info] cand elem has ComplexType
Exception in thread "AWT-EventQueue-0" java.lang.StackOverflowError
at java.nio.CharBuffer.<init>(Unknown Source)
at java.nio.HeapCharBuffer.<init>(Unknown Source)
at java.nio.CharBuffer.wrap(Unknown Source)
at sun.nio.cs.StreamEncoder$CharsetSE.implWrite(Unknown Source)
at sun.nio.cs.StreamEncoder.write(Unknown Source)
at java.io.OutputStreamWriter.write(Unknown Source)
at java.io.BufferedWriter.flushBuffer(Unknown Source)
at java.io.PrintStream.write(Unknown Source)
at java.io.PrintStream.print(Unknown Source)
at java.io.PrintStream.println(Unknown Source)
at matcher.WsdlServiceMatcher.matchComplexTypes(WsdlServiceMatcher.java:1616)
...
```

Bei Änderung des *Requirement* funktioniert der Matcher.

Query 27: travel/geopolitical-entity_weatherprocess_service.wsdl

Eingelesen werden nur die WSDL Beschreibungen für die Konfiguration d0 und d0h. Sobald weitere Elemente dazukommen, werden die WSDL Beschreibungen sehr umfangreich. Ein Beispiel ist der Typ `WeatherProcessType`, der ohne Vererbung als `SimpleType` dargestellt wird. Sobald Elemente über Vererbung mit aufgenommen werden, werden auch bislang problematische komplexe Typen angezogen, die einen Zyklus beinhalten.

Candidate service	d0	Cos	EJ	JS	Semantic degree
geopolitical-entity_weatherprocess_service.wsdl	100,00	96,00	100,00	100,00	Exact
geopolitical-entity_weatherseasonproposition_service.wsdl	81,81	86,00	81,00	89,00	Plugin
geopolitical-entity_weatherseasontimeposition_service.wsdl	81,81	80,00	71,00	81,00	Plugin
geopolitical-entity_weatherseason_service.wsdl	81,81	92,00	92,00	96,00	Plugin
geopolitical-entity_weathersystem_service.wsdl	81,81	92,00	92,00	96,00	Plugin
geographical-region_weatherprocess_GRWservice.wsdl	63,63	92,00	92,00	95,00	Plugin
geographical-region_weatherprocess_service.wsdl	63,63	92,00	92,00	95,00	Plugin
country_weatherprocess_service.wsdl	54,54	95,00	97,00	98,00	Failed
geographical-region_lightning_service.wsdl	45,45	86,00	75,00	90,00	Subsumes
geographical-region_weatherseason_service.wsdl	45,45	88,00	84,00	91,00	Plugin
geographical-region_weathersystem_service.wsdl	45,45	88,00	84,00	91,00	Plugin
_weatherprocess_Germanservice.wsdl	45,45	47,00	50,00	50,00	Exact
_icing_Germanservice.wsdl	27,27	43,00	42,00	46,00	Plugin
_warmfront_Italyservice.wsdl	27,27	40,00	36,00	42,00	Plugin
_weatherfront_Britishservice.wsdl	27,27	43,00	42,00	46,00	Plugin
_weatherfront_Germanyservice.wsdl	27,27	22,00	8,00	15,00	Plugin

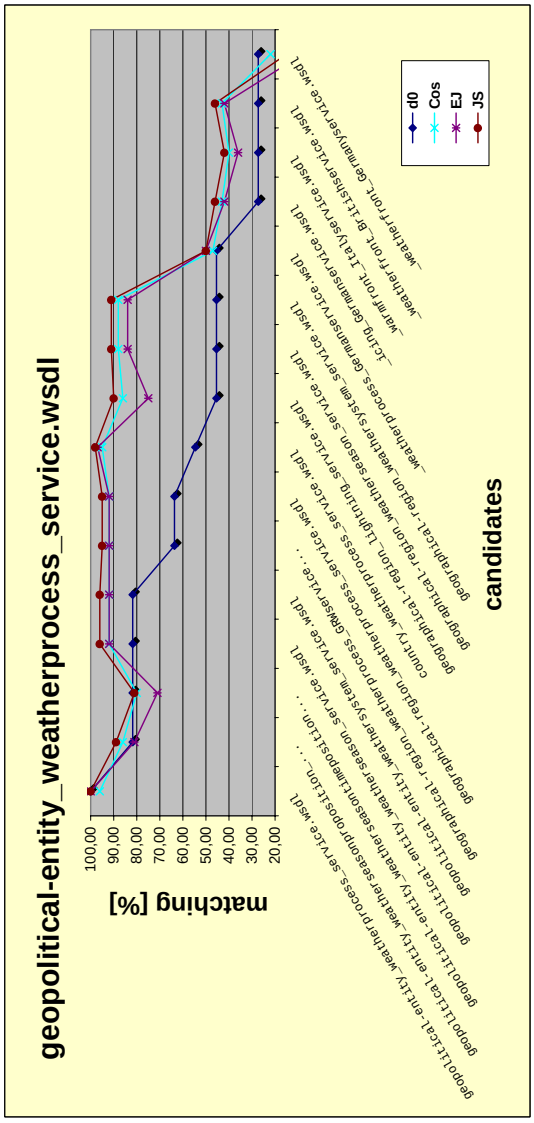


Abbildung 8.28: Query 24: travel/geographical-regiongeographical-region_map_service.wsdl

Die Interpretation von OWL-Klassen als `SimpleType` ist Ursache für die größere Abweichung der Matchmakingwerte des WA zu dem OWLS-MX. Problematisch ist aber die Übersetzung zu einem `ComplexType`, da dessen Elemente Zyklen enthalten.

Query 28: weapon-governmentmissile_funding_service.wsdl

Eingelesen werden nur die WSDL Beschreibungen für die Konfiguration d0 und d0h. Sobald ab einer Vererbungstiefe von 1 weitere Elemente dazukommen, werden die WSDL Beschreibungen sehr umfangreich. Es werden sehr viele Elemente und Subtypen mit einbezogen, die Zyklen enthalten. Die Zahl der XML Schema Typen in der Schema Definition erschwert eine Identifikation aller Zyklen. Ein gefunder Typ ist `Formula`.

Candidate service	d0	Cos	EJ	JS	Semantic degree
governmentmissile_funding_Reliableservice.wsdl	100,00	96,00	100,00	100,00	Exact
governmentmissile_funding_service.wsdl	100,00	96,00	100,00	100,00	Exact
governmentmissilesweapon_funding_service.wsdl	80,00	96,00	99,00	100,00	Plugin
governmentmissile_financing_service.wsdl	80,00	87,00	77,00	92,00	Plugin
governmentweapon_funding_service.wsdl	80,00	90,00	94,00	97,00	Plugin
missilegovernmentorganization_funding_service.wsdl	80,00	94,00	99,00	99,00	Plugin
nationalgovernmentweapon_funding_service.wsdl	80,00	89,00	93,00	96,00	Failed
projectilegovernment_funding_service.wsdl	80,00	91,00	95,00	98,00	Plugin
government_funding_BallMissileservice.wsdl	70,00	81,00	81,00	90,00	Exact
government_funding_Missileservice.wsdl	70,00	81,00	81,00	90,00	Exact
missile_funding_Asianservice.wsdl	70,00	90,00	85,00	92,00	Exact
missile_funding_Indiaservice.wsdl	70,00	90,00	85,00	92,00	Exact
missile_funding_NKoreaservice.wsdl	70,00	90,00	85,00	92,00	Exact
missile_funding_Pakservice.wsdl	70,00	90,00	85,00	92,00	Exact
ballisticmissilegovernment_fundingrange_service.wsdl	60,00	92,00	93,00	96,00	Failed
governmentmissileweapon_fundingfunding_service.wsdl	60,00	96,00	99,00	100,00	Plugin
governmentorganizationmissile_unilateralgiving_service.wsdl	60,00	87,00	88,00	93,00	Failed
missilegovernment_giving_service.wsdl	60,00	91,00	94,00	97,00	Subsume
missile_financing_Chinaservice.wsdl	50,00	82,00	62,00	83,00	Plugin
missile_financing_Russianservice.wsdl	50,00	82,00	62,00	83,00	Plugin
missile_financing_USservice.wsdl	50,00	82,00	62,00	83,00	Plugin

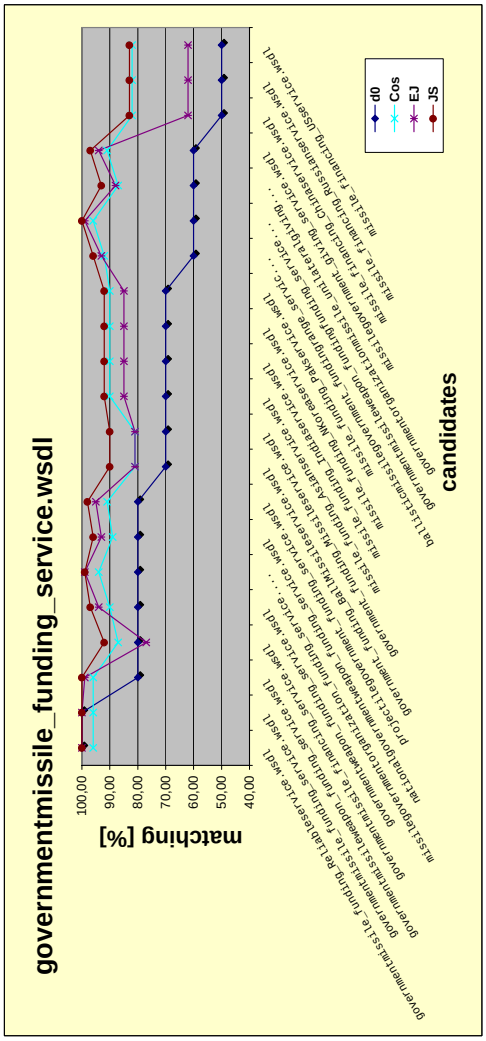


Abbildung 8.29: Query 28: weapon-governmentmissile_funding_service.wsdl

Query 29: EBookOrder1.wsdl

Candidate service	WA(d0)	WA(d0h)	WA(d2)	WA(d2h)	Cos	EJ	JS	Semantic degree
EBookOrder1.wsdl	100,00	100,00	100,00	100,00	74,00	69,00	69,00	Exact
EBookOrder2.wsdl	92,18	90,09	94,15	92,18	74,00	69,00	69,00	Exact
EBookOrder3.wsdl	92,18	90,09	94,15	92,18	74,00	69,00	69,00	Exact
bookinfo3.wsdl	76,56	70,29	82,45	76,56	82,00	81,00	81,00	Exact
bookinfo2.wsdl	57,03	45,54	67,83	57,03	82,00	81,00	81,00	Exact
BookFinder.wsdl	53,12	40,59	64,91	53,12	82,00	81,00	81,00	Exact
BookSearchService.wsdl	53,12	40,59	64,91	53,12	82,00	81,00	81,00	Exact
isbn_book_service.wsdl	19,53	24,75	14,61	19,53	16,00	12,00	12,00	Failed
publication-number_book_service.wsdl	19,53	24,75	14,61	19,53	16,00	12,00	12,00	Failed
publication_book_service.wsdl	19,53	24,75	14,61	19,53	27,00	23,00	23,00	Failed
_book_Oracleservice.wsdl	11,71	14,85	8,77	11,71	16,00	12,00	12,00	Failed

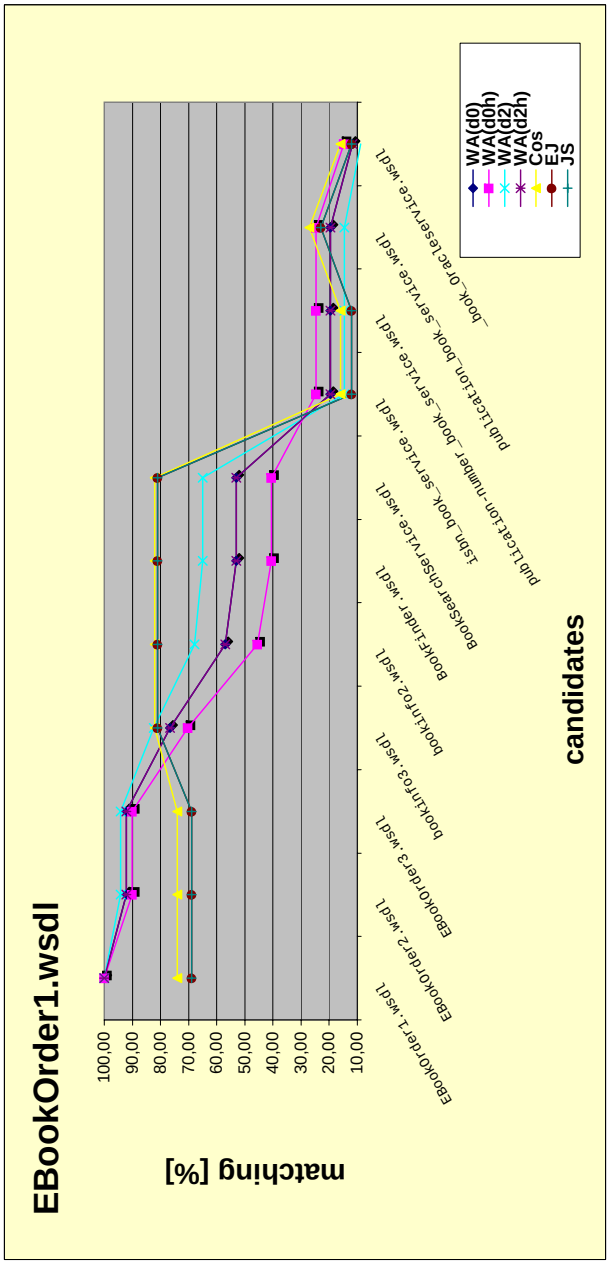


Abbildung 8.30: Query 29: EBookOrder1.wsdl

Einige WSDL Matchmaking-Ergebnisse lassen sich mit den Werten der OWLS-TC vergleichen. Weiterhin fällt auf, dass die Benutzung des *Hierarchy Pattern* bereits ohne explizite Konfiguraton der Vererbung (d0h) zu einem Unterschied in den Ergebnissen führt.

8.6 Diskussion der Ergebnisse

Man kann feststellen, dass die Matchmaking-Ergebnisse des *WSDL Analyzers* durchaus mit den Ergebnissen des *OWLS-MX* vergleichbar sind. Probleme machen WSDL Beschreibungen, deren Parametertypen auf komplexen XML Schematypen basieren, die Zyklen beinhalten. Ein Problem bei der Analyse mit dem *WSDL Analyzer* ist, dass nicht alle Zyklen stören. Dieses Verhalten muss künftig weiter untersucht werden, um den *WSDL Analyzer* zu verbessern.

Ontologien, für deren Übersetzung Zyklen auftreten, die nicht übersetzt werden können sind:

- Mid-level-ontology.owl
- SUMO.owl

Diese Ontologien beschreiben eine große Vererbungshierarchie. Dies betrifft die Tiefe sowie den Umfang des Vererbungsbaumes bzw. Vererbungsgraphen pro Vererbungsebene. Typen mit einer Elementanzahl von über 30 sind ab Vererbungstiefe 2 keine Seltenheit.

Bei Analyse mit dem *WSDL Analyzer* kann man feststellen, dass die Konfiguration des XML Schema Generators entscheidend für das Matchmakingverhalten der generierten WSDL Beschreibungen ist.

8.7 Arbeit mit dem WSDL Analyzer

Um die generierten WSDL Beschreibungen lesen und verarbeiten zu können, musste der *WSDL Analyzer* untersucht und aufgrund folgender Probleme angepasst werden:

- Bei XML Schema Definitionen führen definierte Zyklus (zwei Datentypen beinhalten Elemente des jeweils anderen Typs) zu einer Unendlichschleife beim Aufbau des internen Datenmodells.
- Nicht-terminierendes Verhalten bei der Verabeitung des internen Datenmodells.
- Bug bei der Verarbeitung von `anyURI`
- Verwendung der Sprachelemente `restriction base` sowie `extension base` um den Basistyp eines `ComplexType` festzulegen führte zu Fehlern beim Aufbau des internen Datenmodells.
- WordNet Abfragen zur Zeit nicht möglich *(broken)*

Kapitel 9

Zusammenfassung

9.1 Vergleich semantisches und syntaktisches Matchmaking

Die Frage, ob die generierten WSDL Beschreibungen zu den gleichen Matchmaking Ergebnissen führen (WA) wie die übersetzten OWL-S Definitionen (OWLS-MX), lässt sich nicht mit ja oder nein beantworten, da für das syntaktische Matchmaking ein anderes Ähnlichkeitsmaß benutzt wird, als bei dem semantischen Matchmaking von Services. Für ein syntaktisches Matchmaking lässt sich die Ähnlichkeit zwischen zwei Diensten (bzw. ihrer Schnittstellen) in Prozent ausdrücken, bei einem semantischen Matchmaking wird die Ähnlichkeit über **Exact, Plugin, Subsumes und Fail** und ein Ranking ausgedrückt.

Die vorgestellte Translation bildet über das *Hierarchy Pattern* eine Klassenhierarchie der Ontologie ab (Generalisierung). Eine Einschränkung der aktuellen Version ist, dass die Implementierung des *Hierarchy Pattern* nur eine Superklasse betrachtet und so lediglich eine Taxonomie abbilden kann. Weiterhin werden Terme aus der verwendeten Beschreibungslogik (OWL-DL) ausgewertet und anonyme Klassen zu entsprechenden Schema Elementen und Schema Typen übersetzt. Diese haben prinzipiell die gleiche Aussage wie die OWL-DL Beschreibung (z.B. *hasIntersection, hasUnion, hasComplement*). Ein Unterschied des Sinngehalts besteht allerdings, da die Translation zur Erstellung des Metamodells die OWL-DL Beschreibung “nur möglichst gut” interpretiert. Die Frage, ob ein Matchmaking der generierten WSDL Beschreibungen vergleichbar zu dem Matchmaking der OWL-S Definitionen ist, hängt also von der *Matchmaking Engine* und dem übersetzten Informationsgehalt (anonyme Klassen) ab. Wird eine *Inference Engine* bzw. ein *Reasoner* eingesetzt, der die Signatur des WSDL Dienstes und alle Typen der XML Schema Definition versteht, lässt sich auch hier ein gutes Matchingverhalten schlußfolgern.

Die Evaluierung der Translation durch die Analyse der generierten WSDL Beschreibungen mit dem *WSDL Analyzer* zeigt, dass der *WSDL Analyzer* ähnliche Ergebnisse erzielt wie die syntaktische *Matchmaking Engine* des OWLS-MX, die auf IR-Metriken basiert.
Man kann so feststellen, dass die Übersetzung von OWL-Klassen zu XML Schema Typen und die Übersetzung von OWL Eigenschaften zu XML Schema Elementen unter Betrachtung der Struktur und der Vererbungshierachie von Eigenschaften an Unterklassen eine Strategie dargestellt, mit der beide Matchmaking Technologien verglichen werden können. Abhängig ist das Resultat von der

Arbeitsweise des WSDL Matchmakers (z.B. Betrachtung der *substitutionGroup* eines Elements mit komplexem Typ oder des Basistyps eines komplexen Typs). Durch Betrachtung der zusätzlich übersetzten anonymen Klassen wird das Matchmakingergebnis weiter verbessert.

9.2 Bearbeitung des Themas

Für die Konzeptionierung der Translation und das Design der Anwendung konnten anfangs nur wenige Vorgaben gemacht werden. Während der Entwicklung und der späteren Testphase zusammen mit dem *WSDL Analyzer* wurden weitere Anforderungen an die Translation spezifiziert. Dieser Sachverhalt wird in Kapitel 4 durch die Präsentation des Vorgehensmodells verdeutlicht.

9.3 Fazit

Die Konzeption der implementierten Translation *OWL-S nach WSDL* machte einen Großteil dieser Arbeit aus. Gründe waren das Fehlen von Referenzen, die sich mit diesem *top dow* Ansatz beschäftigen und das Fehlen von OWL-S Anwendungsbeispielen, die WSDL Groundings vorstellen, in denen OWL Individuals vollständig über **eine** WSDL Schnittstelle instanziert werden. In der Regel werden nur wenige Eigenschaften einer OWL-Klasse in einer OWL Instanz genutzt (ZipCode Beispiel mindswap, siehe B.1).

Ziel dieser Arbeit war es, die semantische Bedeutung eines Begriffs bzw. einer OWL-Klasse möglichst gut zu interpretieren und anhand der gesammelten Information einen entsprechenden XML Schema Typ zu generieren. Dadurch wird es ermöglicht, alle interpretierten Eigenschaften über **eine** WSDL Schnittstelle zu nutzen.

Übersetzt wurden die OWL-S Definitionen der OWLS-MX Testkollektion (OWLS-TC). Die generierten WSDL Beschreibungen werden benutzt, um OWL-S Definitionen der Testszenarien jeweils um ein passendes WSDL Grounding zu erweitern. Der *WSDL Analyzer* wurde für die Evaluation der Translation benutzt. Die so ermittelten Ähnlichkeitswerte der generierten WSDL Beschreibungen bestätigen in etwa die Ergebnisse des OWLS-MX (Cos, EJ, JS Metriken).

Bei dieser Form der Übersetzung von OWL-Klassen kann man feststellen, dass die Definition von komplexen Schema Typen durch die Auswertung der Ontologie sehr umfangreich werden kann. Aktuell in der Praxis genutzte XML Schema Definitionen sind zwar auch sehr umfangreich, allerdings sollte immer untersucht werden, ob alle Eigenschaften der Schnittstelle, die über Abfrage des Ontologie-Modells und durch semantisches Schlußfolgern gefunden werden, auch benötigt werden bzw. für den Anwendungsfall Sinn machen (z.B. Beschreibung mathematischer Formeln).

Anhang A

Beispiele Translation OWL nach XSD

A.1 Generierung von XML Schema Typen und Sub-Typen

Mit Hilfe von Protégé wurde eine Ontologie erstellt, mit der man Studenten beschreiben kann. Anschließend wurde die Klasse MasterStudent nach XML Schema übersetzt.

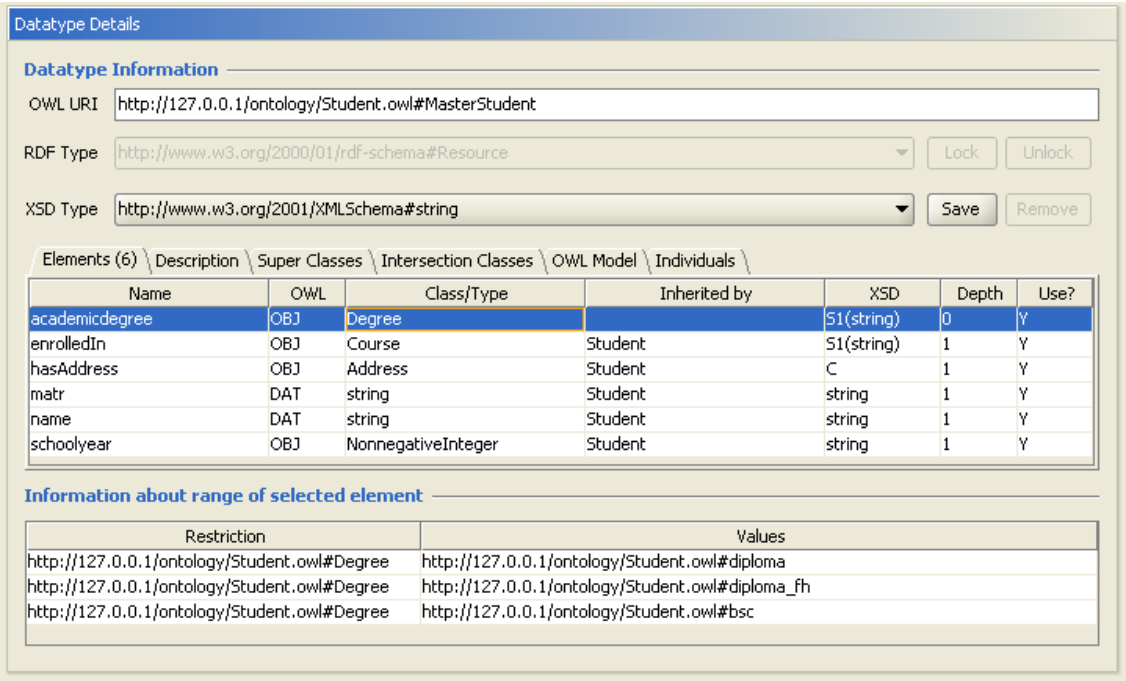


Abbildung A.1: Beispiel. OWL-Klasse MasterStudent. OWLS2WSDL Ansicht.

Die grafische Benutzeroberfläche von OWLS2WSDL zeigt in den Datentyp-Details an, aus welchen Elementen der Typ besteht. Für jedes Element zeigt die **Spalte XSD** der Tabelle an, welcher Typ für das Element generiert wird.

```
...
  <xsd:element name="MasterStudent" type="MasterStudentType"/>
  <xsd:complexType name="MasterStudentType">
4   <xsd:annotation>
      <xsd:documentation>ComplexType</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string" />
9     <xsd:element name="matr" type="xsd:string" />
      <xsd:element name="academicdegree" type="Degree" />
      <xsd:element name="schoolyear" type="NonnegativeInteger" />
      <xsd:element name="enrolledIn" type="Course" />
      <xsd:element name="hasAddress" type="Address" />
14  </xsd:sequence>
  </xsd:complexType>
...
```

Listing A.1: OWL2XSD: Translation der OWL-Klasse MasterStudent

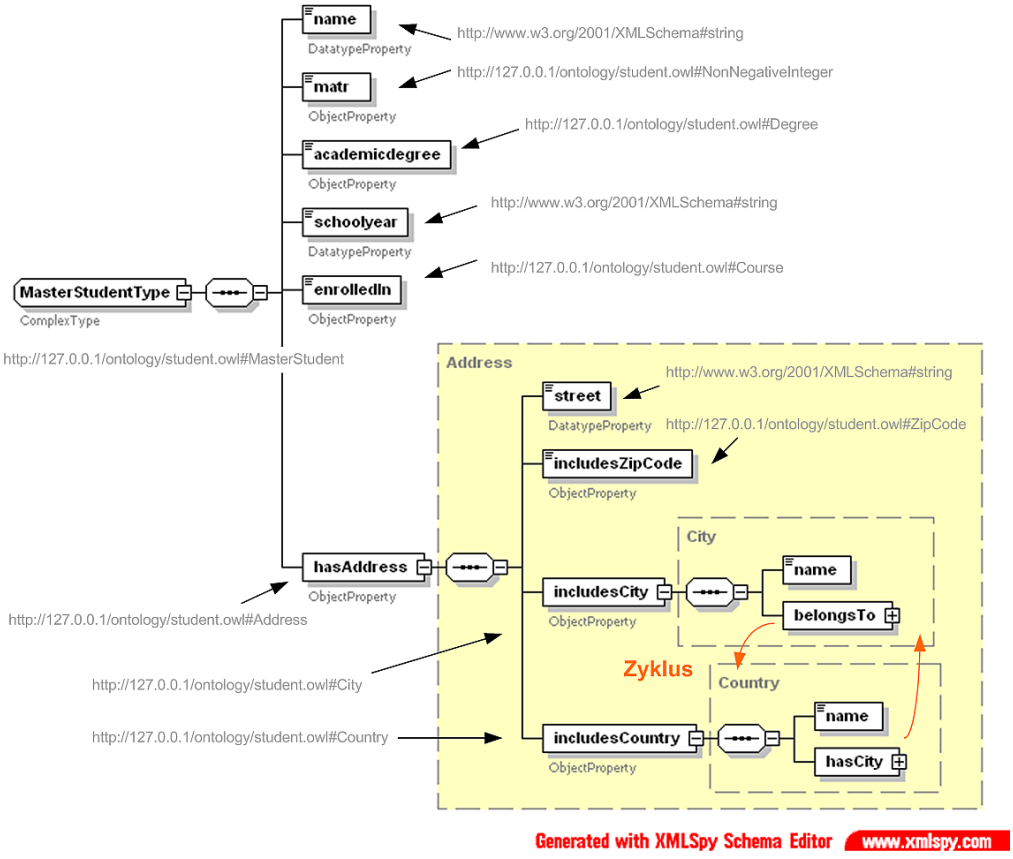


Abbildung A.2: Beispiel. XML Schema MasterStudentType

Der generierte XML Schema Typ MasterStudentType besteht aus dem Element `hasAddress`, das einen weiteren komplexem Typ Address besitzt. Weitere Elemente von MasterStudent sind `academicdegree`, `schoolyear` und `enrolledIn` mit simplen Typ und den Elementen `name` und `matr` mit primitiven Datentyp `xsd:string`.

XML Schema Instanzen

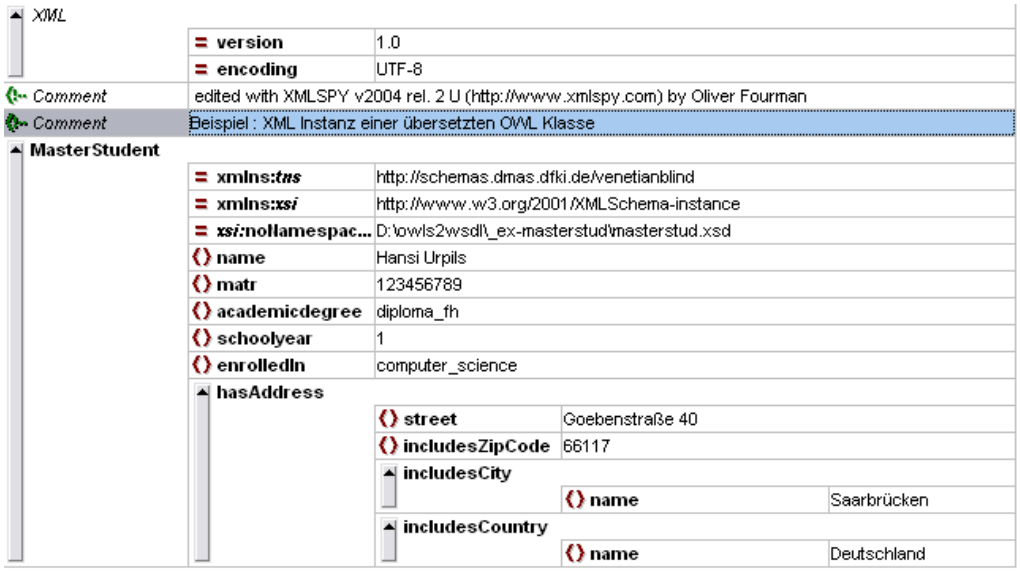


Abbildung A.3: Beispiel. Instanz MasterStudentType (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 2 U (http://www.xmlspy.com) by Oliver Fourman -->
<!--Beispiel : XML Instanz einer übersetzten OWL Klasse-->
<MasterStudent xmlns:tns="http://schemas.dmas.dfki.de/venetianblind"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="masterstud.xsd">
    <name>Hansi Urpils</name>
    <matr>123456789</matr>
    <academicdegree>diploma_fh</academicdegree>
    <schoolyear>1</schoolyear>
    <enrolledIn>computer_science</enrolledIn>
    <hasAddress>
        <street>Goebenstraße 40</street>
        <includesZipCode>66117</includesZipCode>
        <includesCity>
            <name>Saarbrücken</name>
        </includesCity>
        <includesCountry>
            <name>Deutschland</name>
        </includesCountry>
    </hasAddress>
</MasterStudent>
```

Abbildung A.4: Beispiel. Instanz MasterStudentType (2)

A.2 Übersetzung der Wine Ontologie (wine.owl)

Die *Wine Ontologie* (wine.owl) wird innerhalb des W3 OWL Guides[1] vorgestellt und ist aufgrund der Tatsache, dass sie sehr viele OWL Sprachelemente enthält, sehr interessant für die Evaluation der durchzuführenden Translation. Inhalt der Ontologie ist die Beschreibung von verschiedenen Weinsorten, die klassifiziert werden. Durch Restriktierung der Klasse Wein werden immer neue Weinsorten und Weinarten definiert (siehe Abb. A.5).

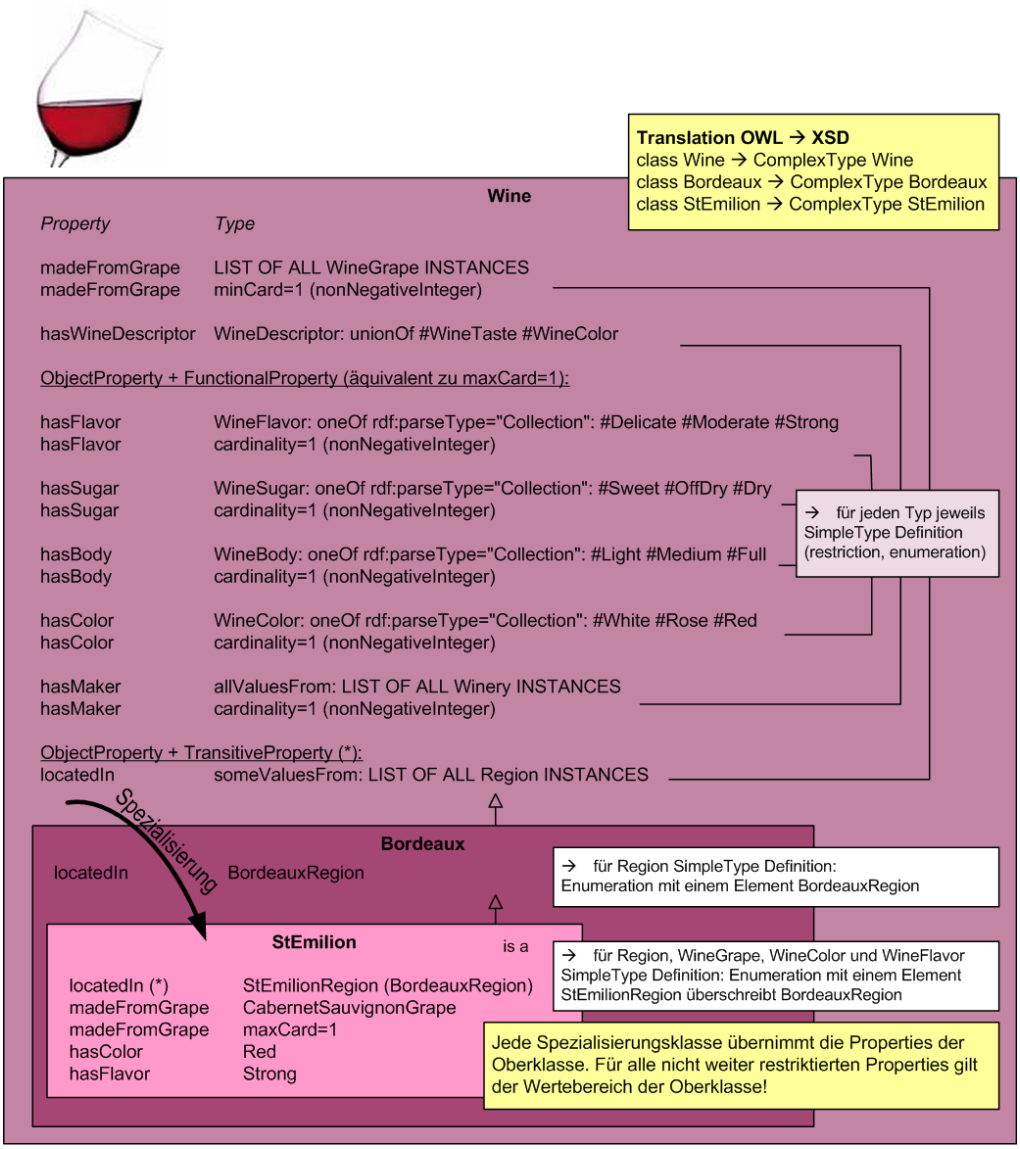


Abbildung A.5: Wine Ontologie, Eigenschaften der Klasse StEmilion

[1] http://www.w3.org/TR/owl-guide/

Anhand der Wine Ontologie wird die Einschränkung des Wertebereichs in XML Schema gezeigt. Übersetzt wurde die Klasse StEmilion deren Wertebereich z.B. durch den Gebrauch von `hasValue` rerstriktiert wurde.

http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#StEmilion

```
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor" />
    <owl:oneOf rdf:parseType="Collection">
      <owl:Thing rdf:about="#White" />
      <owl:Thing rdf:about="#Rose" />
      <owl:Thing rdf:about="#Red" />
    </owl:oneOf>
</owl:Class>
...
<owl:ObjectProperty rdf:ID="hasColor">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range rdf:resource="#WineColor" />
</owl:ObjectProperty>
...
<owl:Class rdf:ID="StEmilion">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor" />
        <owl:hasValue rdf:resource="#Red" />
      </owl:Restriction>
    </rdfs:subClassOf>
    ...
...
```

Listing A.2: OWL2XSD: Beispiel owl:hasValue

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.dmas.dfki.de/venetianblind"
    version="OWLS2WSDL␣Thu␣May␣10␣19:04:26␣CEST␣2007">
    <xsd:annotation>
        <xsd:documentation source="Translation␣(OWL2XSD-ComplexType)␣of
http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#StEmilion"/>
        <xsd:documentation source="Translation␣(OWL2XSD-ComplexType)␣of
http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Bordeaux"/>
        <xsd:documentation source="Translation␣(OWL2XSD-ComplexType)␣of
http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Wine"/>
    </xsd:annotation>

<xsd:element name="StEmilion" type="StEmilionType" substitutionGroup="Bordeaux"/>
<xsd:element name="Bordeaux" type="BordeauxType" abstract="true"
        substitutionGroup="Wine"/>
<xsd:element name="Wine" type="WineType" abstract="true"
        substitutionGroup="PotableLiquid"/>
<xsd:element name="PotableLiquid" type="PotableLiquid" abstract="true"/>

    <xsd:complexType name="StEmilionType">
    <xsd:complexContent>
        <xsd:restriction base="BordeauxType">
            <xsd:choice>
```

```
                <xsd:element name="stEmilionID" type="StEmilionDATA"/>
                <xsd:sequence>
                  <xsd:element name="name" minOccurs="0" type="xsd:string"/>
                  <xsd:element name="locatedIn" type="StEmilionRegion"/>
                  <xsd:element name="madeFromGrape" type="StEmilionWineGrape"/>
                  <xsd:element name="hasColor" minOccurs="0" type="StEmilionWineColor"/>
                  <xsd:element name="hasMaker" type="Winery"/>
                  <xsd:element name="hasFlavor" type="StEmilionWineFlavor"/>
                  <xsd:element name="hasSugar" type="WineSugar"/>
                  <xsd:element name="hasBody" type="WineBody"/>
                </xsd:sequence>
            </xsd:choice>
        </xsd:restriction>
    </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="BordeauxType">
    <xsd:complexContent>
        <xsd:restriction base="WineType">
            <xsd:sequence>
                <xsd:element name="locatedIn" type="BordeauxRegion"/>
                <xsd:element name="madeFromGrape" type="WineGrape"/>
                <xsd:element name="hasColor" minOccurs="0" type="WineColor"/>
                <xsd:element name="hasMaker" type="Winery"/>
                <xsd:element name="hasFlavor" type="WineFlavor"/>
                <xsd:element name="hasSugar" type="WineSugar"/>
                <xsd:element name="hasBody" type="WineBody"/>
            </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="WineType">
        <xsd:sequence>
            <xsd:element name="madeFromGrape" type="WineGrape"/>
            <xsd:element name="hasColor" minOccurs="0" type="WineColor"/>
            <xsd:element name="hasMaker" type="Winery"/>
            <xsd:element name="locatedIn" type="Region"/>
            <xsd:element name="hasFlavor" type="WineFlavor"/>
            <xsd:element name="hasSugar" type="WineSugar"/>
            <xsd:element name="hasBody" type="WineBody"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:simpleType name="StEmilionDATA">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="ChateauChevalBlancStEmilion"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="PotableLiquid">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="WineBody">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Light"/>
            <xsd:enumeration value="Full"/>
            <xsd:enumeration value="Medium"/>
        </xsd:restriction>
```

```
    </xsd:simpleType>
    <xsd:simpleType name="WineGrape">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="GamayGrape"/>
            <xsd:enumeration value="SemillonGrape"/>
            <xsd:enumeration value="MalbecGrape"/>
            <xsd:enumeration value="ZinfandelGrape"/>
            <xsd:enumeration value="CabernetFrancGrape"/>
            <xsd:enumeration value="SangioveseGrape"/>
            <xsd:enumeration value="CheninBlancGrape"/>
            <xsd:enumeration value="RieslingGrape"/>
            <xsd:enumeration value="PinotNoirGrape"/>
            <xsd:enumeration value="MerlotGrape"/>
            <xsd:enumeration value="ChardonnayGrape"/>
            <xsd:enumeration value="CabernetSauvignonGrape"/>
            <xsd:enumeration value="SauvignonBlancGrape"/>
            <xsd:enumeration value="PinotBlancGrape"/>
            <xsd:enumeration value="PetiteVerdotGrape"/>
            <xsd:enumeration value="PetiteSyrahGrape"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="StEmilionRegion">
        <xsd:restriction base="Region">
            <xsd:enumeration value="StEmilionRegion"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="WineSugar">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="OffDry"/>
            <xsd:enumeration value="Sweet"/>
            <xsd:enumeration value="Dry"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="WineColor">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Red"/>
            <xsd:enumeration value="Rose"/>
            <xsd:enumeration value="White"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="StEmilionWineGrape">
        <xsd:restriction base="WineGrape">
            <xsd:enumeration value="CabernetSauvignonGrape"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="StEmilionWineFlavor">
        <xsd:restriction base="WineFlavor">
            <xsd:enumeration value="Strong"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="Winery">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="KathrynKennedy"/>
            <xsd:enumeration value="ChateauDeMeursault"/>
            <xsd:enumeration value="SeanThackrey"/>
            <xsd:enumeration value="Ventana"/>
            <xsd:enumeration value="Cotturi"/>
            <xsd:enumeration value="Handley"/>
```

```
            <xsd:enumeration value="ChateauLafiteRothschild"/>
            <xsd:enumeration value="ChateauMargauxWinery"/>
            <xsd:enumeration value="ChateauMorgon"/>
            <xsd:enumeration value="PeterMccoy"/>
            <xsd:enumeration value="Forman"/>
            <xsd:enumeration value="SaucelitoCanyon"/>
            <xsd:enumeration value="ChateauChevalBlanc"/>
            <xsd:enumeration value="SchlossVolrad"/>
            <xsd:enumeration value="Marietta"/>
            <xsd:enumeration value="SevreEtMaine"/>
            <xsd:enumeration value="Elyse"/>
            <xsd:enumeration value="Longridge"/>
            <xsd:enumeration value="Bancroft"/>
            <xsd:enumeration value="SchlossRothermel"/>
            <xsd:enumeration value="DAnjou"/>
            <xsd:enumeration value="Stonleigh"/>
            <xsd:enumeration value="Taylor"/>
            <xsd:enumeration value="ClosDeVougeot"/>
            <xsd:enumeration value="Corbans"/>
            <xsd:enumeration value="Foxen"/>
            <xsd:enumeration value="PageMillWinery"/>
            <xsd:enumeration value="StGenevieve"/>
            <xsd:enumeration value="MountEdenVineyard"/>
            <xsd:enumeration value="CongressSprings"/>
            <xsd:enumeration value="CortonMontrachet"/>
            <xsd:enumeration value="Selaks"/>
            <xsd:enumeration value="GaryFarrell"/>
            <xsd:enumeration value="WhitehallLane"/>
            <xsd:enumeration value="ChateauDYchem"/>
            <xsd:enumeration value="ClosDeLaPoussie"/>
            <xsd:enumeration value="LaneTanner"/>
            <xsd:enumeration value="Beringer"/>
            <xsd:enumeration value="McGuinnesso"/>
            <xsd:enumeration value="Mountadam"/>
            <xsd:enumeration value="PulignyMontrachet"/>
            <xsd:enumeration value="KalinCellars"/>
            <xsd:enumeration value="SantaCruzMountainVineyard"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="BordeauxRegion">
        <xsd:restriction base="Region">
            <xsd:enumeration value="BordeauxRegion"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="WineFlavor">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Delicate"/>
            <xsd:enumeration value="Strong"/>
            <xsd:enumeration value="Moderate"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="Region">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="CaliforniaRegion"/>
            <xsd:enumeration value="FrenchRegion"/>
            <xsd:enumeration value="SancerreRegion"/>
            <xsd:enumeration value="SauterneRegion"/>
            <xsd:enumeration value="BordeauxRegion"/>
```

```
                <xsd:enumeration value="TexasRegion"/>
                <xsd:enumeration value="SonomaRegion"/>
                <xsd:enumeration value="MedocRegion"/>
                <xsd:enumeration value="NapaRegion"/>
                <xsd:enumeration value="ToursRegion"/>
                <xsd:enumeration value="ChiantiRegion"/>
                <xsd:enumeration value="EdnaValleyRegion"/>
                <xsd:enumeration value="ArroyoGrandeRegion"/>
                <xsd:enumeration value="AnjouRegion"/>
                <xsd:enumeration value="BeaujolaisRegion"/>
                <xsd:enumeration value="AlsaceRegion"/>
                <xsd:enumeration value="SouthAustraliaRegion"/>
                <xsd:enumeration value="LoireRegion"/>
                <xsd:enumeration value="CentralTexasRegion"/>
                <xsd:enumeration value="BourgogneRegion"/>
                <xsd:enumeration value="USRegion"/>
                <xsd:enumeration value="GermanyRegion"/>
                <xsd:enumeration value="MuscadetRegion"/>
                <xsd:enumeration value="NewZealandRegion"/>
                <xsd:enumeration value="SantaBarbaraRegion"/>
                <xsd:enumeration value="PauillacRegion"/>
                <xsd:enumeration value="CotesDOrRegion"/>
                <xsd:enumeration value="AustralianRegion"/>
                <xsd:enumeration value="ItalianRegion"/>
                <xsd:enumeration value="MeursaultRegion"/>
                <xsd:enumeration value="StEmilionRegion"/>
                <xsd:enumeration value="MendocinoRegion"/>
                <xsd:enumeration value="SantaCruzMountainsRegion"/>
                <xsd:enumeration value="CentralCoastRegion"/>
                <xsd:enumeration value="PortugalRegion"/>
                <xsd:enumeration value="MargauxRegion"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="StEmilionWineColor">
            <xsd:restriction base="WineColor">
                <xsd:enumeration value="Red"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:schema>
```

Listing A.3: OWL2XSD Beispiel: StEmilionType

A.3 Übersetzung von anonymen Typen (Process.owl)

`http://www.daml.org/services/owl-s/1.1/Process.owl#Process`
Folgendes XML Schema zeigt eine Translation der OWL Klasse Process. Der in OWL anonyme Typ, der die Vereinigungsmenge von AtomicProcess, SimpleProcess und CompositeProcess beschreibt, wird in XML Schema als `SufficientProcessUnion` bezeichnet.

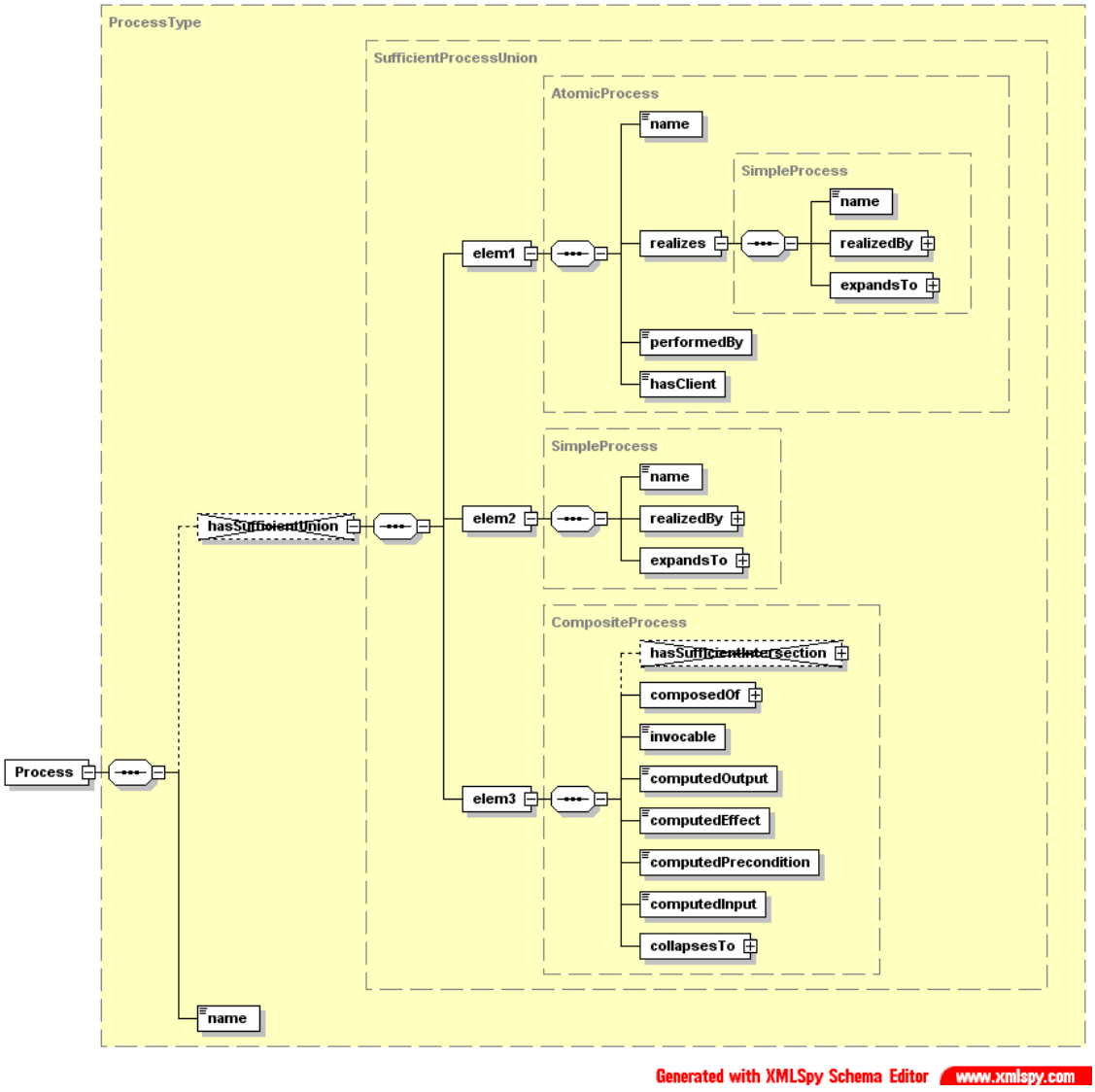


Abbildung A.6: OWL2XSD: Generierter Typ ProcessType (aus der OWL-S Ontologie)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:tns="http://schemas.dmas.dfki.de/venetianblind"
            version="OWLS2WSDL␣Mon␣Apr␣30␣10:02:41␣CEST␣2007">
  <xsd:annotation>
    <xsd:documentation source="Translation␣(OWL2XSD-ComplexType)␣of
http://www.daml.org/services/owl-s/1.1/Process.owl#Process"/>
  </xsd:annotation>
  <xsd:element name="Process" type="ProcessType"/>
  <xsd:complexType name="ProcessType">
    <xsd:sequence>
      <xsd:element name="hasSufficientUnion" type="SufficientProcessUnion"
      minOccurs="0" maxOccurs="0"/>
      <xsd:element name="name" type="xsd:anyURI"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SufficientProcessUnion">
    <xsd:complexContent>
      <xsd:extension base="Union">
        <xsd:sequence>
          <xsd:element name="elem1" type="AtomicProcess"/>
          <xsd:element name="elem2" type="SimpleProcess"/>
          <xsd:element name="elem3" type="CompositeProcess"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="AtomicProcess">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:anyURI"/>
      <xsd:element name="realizes" type="SimpleProcess"/>
      <xsd:element name="performedBy" type="AtomicProcessParticipant"/>
      <xsd:element name="hasClient" type="AtomicProcessParticipant"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="AtomicProcessParticipant">
    <xsd:restriction base="Participant">
      <xsd:enumeration value="TheServer"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="Participant">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="TheServer"/>
      <xsd:enumeration value="TheClient"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="SimpleProcess">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:anyURI"/>
      <xsd:element name="realizedBy" type="AtomicProcess"/>
      <xsd:element name="expandsTo" type="CompositeProcess"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="CompositeProcess">
    <xsd:sequence>
      <xsd:element name="hasSufficientIntersection"
       type="SufficientCompositeProcessIntersection" minOccurs="0" maxOccurs="0"/>
      <xsd:element name="composedOf" type="ControlConstruct"/>
```

```
        <xsd:element name="invocable" type="xsd:boolean"/>
        <xsd:element name="computedOutput" type="Thing"/>
        <xsd:element name="computedEffect" type="Thing"/>
        <xsd:element name="computedPrecondition" type="Thing"/>
63      <xsd:element name="computedInput" type="Thing"/>
        <xsd:element name="collapsesTo" type="SimpleProcess"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Union" abstract="true"/>
68  <xsd:complexType name="Intersection" abstract="true"/>
    <xsd:complexType name="SufficientCompositeProcessIntersection">
      <xsd:complexContent>
        <xsd:extension base="Intersection">
          <xsd:sequence>
73          <xsd:element name="elem1" type="ProcessType"/>
            <xsd:element name="elem2"
                         type="composedOfCompositeProcessCardinalityRestriction"/>
          </xsd:sequence>
        </xsd:extension>
78    </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="composedOfCompositeProcessCardinalityRestriction">
      <xsd:complexContent>
        <xsd:restriction base="Restriction">
83        <xsd:all>
            <xsd:element name="onProperty"
  type="xsd:anyURI" fixed="http://www.daml.org/services/owl-s/1.1/Process.owl#composedOf"/>
            <xsd:element name="cardinality" type="Cardinality" fixed="1"/>
          </xsd:all>
88      </xsd:restriction>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Restriction" abstract="true">
      <xsd:sequence>
93      <xsd:element name="onProperty" type="OntProperty"/>
        <xsd:element name="restrictedBy" type="RestrictionComponent" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="OntProperty">
98    <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>
    <xsd:complexType name="RestrictionComponent">
      <xsd:choice>
        <xsd:element name="allValuesFrom" type="AllValuesFrom"/>
103     <xsd:element name="someValuesFrom" type="SomeValuesFrom"/>
        <xsd:element name="hasValue" type="HasValue"/>
        <xsd:element name="cardinality" type="Cardinality"/>
        <xsd:element name="minCardinality" type="MinCardinality"/>
        <xsd:element name="maxCardinality" type="MaxCardinality"/>
108   </xsd:choice>
    </xsd:complexType>
    <xsd:complexType name="AllValuesFrom">
      <xsd:sequence>
        <xsd:element name="range" type="DataRange" maxOccurs="unbounded"/>
113   </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="SomeValuesFrom">
      <xsd:sequence>
```

```
        <xsd:element name="range" type="DataRange" maxOccurs="unbounded"/>
118   </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="HasValue">
      <xsd:sequence>
        <xsd:element name="range" type="DataLiteral"/>
123   </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="Cardinality">
      <xsd:restriction base="xsd:nonNegativeInteger"/>
    </xsd:simpleType>
128 <xsd:simpleType name="MaxCardinality">
      <xsd:restriction base="xsd:nonNegativeInteger"/>
    </xsd:simpleType>
    <xsd:simpleType name="MinCardinality">
      <xsd:restriction base="xsd:nonNegativeInteger"/>
133 </xsd:simpleType>
    <xsd:complexType name="DataRange">
      <xsd:choice>
        <xsd:element name="datatypeID" type="xsd:anyURI"/>
        <xsd:element name="rdfsLiteral" type="xsd:anyURI">
138       <xsd:annotation>
            <xsd:documentation source="..."/>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="oneOfDataLiteral" type="DataLiteral" maxOccurs="unbounded"/>
143   </xsd:choice>
    </xsd:complexType>
    <xsd:simpleType name="DataLiteral">
      <xsd:restriction base="xsd:anyURI"/>
    </xsd:simpleType>
148 <xsd:complexType name="ControlConstruct">
      <xsd:sequence>
        <xsd:element name="timeout" type="IntervalThing"/>
      </xsd:sequence>
    </xsd:complexType>
153 <xsd:simpleType name="IntervalThing">
      <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="Thing">
      <xsd:restriction base="xsd:string"/>
158 </xsd:simpleType>
  </xsd:schema>
```

Listing A.4: OWL2XSD: Translation der OWL Klasse Process

A.4 Abbildung des OWL Sprachumfangs mit Hilfe des Hierarchy Pattern

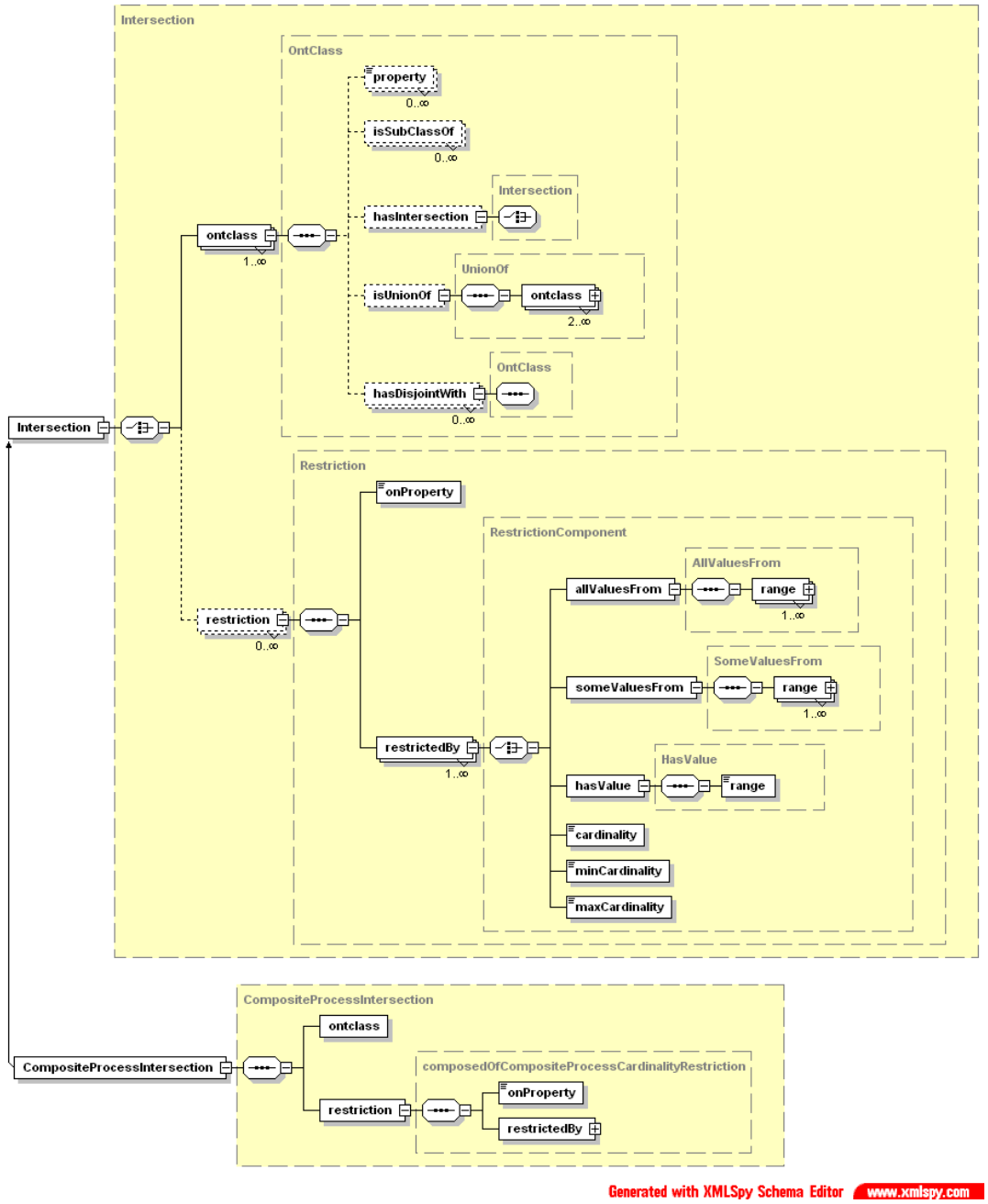


Abbildung A.7: Abbildung: Übersetzte OWL Intersection

Anhang B

OWL-S WSDL Grounding

B.1 mindswap, ZipCodeFinder Service

Das Beispiel bezieht sich auf den in Abschnitt 3.3 vorgestellten ZipCodeFinder Service, der anhand von zwei Eingabeparametern ein OWL ZipCode Individual erstellt. Über das WSDL Grounding, das die Eigenschaften der OWL Klasse ZipCode mit Elementen des XML Schema Typs ZipCodeInfo mappt, muss die zurückgelieferte SOAP Nachricht transformiert werden, um das OWL Individual instanzieren zu können (Prozess Input).

B.1.1 XML Schema

```
<xs:schema id="ZipCodeData"
    targetNamespace="http://www.tilisoft.com/ws/LocInfo/DataObjects/ZipCodeData.xsd"
    ... >
<xs:element name="ZipCodeData" msdata:IsDataSet="true">
  <xs:complexType>
  <xs:choice maxOccurs="unbounded">
    <xs:element name="ZipCodeInfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ZIPCODE" type="xs:string" minOccurs="0"/>
        <xs:element name="CITY" type="xs:string" minOccurs="0"/>
        <xs:element name="STATE" type="xs:string" minOccurs="0"/>
        <xs:element name="FIPS" type="xs:string" minOccurs="0"/>
        <xs:element name="COUNTY" type="xs:string" minOccurs="0"/>
        <xs:element name="COUNTYAREA_SQMI" type="xs:double" minOccurs="0"/>
        <xs:element name="LATITUDE" type="xs:double" minOccurs="0"/>
        <xs:element name="LONGITUDE" type="xs:double" minOccurs="0"/>
        <xs:element name="AREACODE" type="xs:string" minOccurs="0"/>
        <xs:element name="TIMEZONECODE" type="xs:int" minOccurs="0"/>
        <xs:element name="TIMEZONENAME" type="xs:string" minOccurs="0"/>
        <xs:element name="MILES" type="xs:int" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
    </xs:element>
  </xs:choice>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Listing B.1: XML Schema. Komplexer Typ ZipCodeInfo

Das XML Schema ist sehr umfangreich und beinhaltet neben dem durch das Mapping referenzierte Element `ZIPCODE` einige weitere Informationen.

B.1.2 SOAP Nachricht

Der WSDL Service wird über die zwei Parameter City und State aufgerufen und liefert mehrere XML Instanzen mit allen Elementen des Schema Typs zurück.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="ZipCodeData.xsl" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<soap:Body>
  <ListByCityResponse xmlns="http://www.tilisoft.com/ws/LocInfo/literalTypes">
  <ListByCityResult>

  <!-- XML SCHEMA (s.o.) -->

    <diffgr:diffgram>
      <ZipCodeData xmlns="http://www.tilisoft.com/ws/LocInfo/DataObjects/ZipCodeData.xsd">
        <ZipCodeInfo diffgr:id="ZipCodeInfo1" msdata:rowOrder="0">
          <ZIPCODE>20740</ZIPCODE>
          <CITY>COLLEGE PARK</CITY>
          <STATE>MD</STATE>
          <FIPS>033</FIPS>
          <COUNTY>PRINCE GEORGES</COUNTY>
          <COUNTYAREA_SQMI>486</COUNTYAREA_SQMI>
          <LATITUDE>38.9976</LATITUDE>
          <LONGITUDE>76.9258</LONGITUDE>
          <AREACODE>301</AREACODE>
          <TIMEZONECODE>5</TIMEZONECODE>
          <TIMEZONENAME>Eastern</TIMEZONENAME>
          <MILES>0</MILES>
        </ZipCodeInfo>
        <ZipCodeInfo diffgr:id="ZipCodeInfo2" msdata:rowOrder="1">
          <ZIPCODE>20741</ZIPCODE>
          <CITY>COLLEGE PARK</CITY>
          <STATE>MD</STATE>
          <FIPS>033</FIPS>
          <COUNTY>PRINCE GEORGES</COUNTY>
          <COUNTYAREA_SQMI>486</COUNTYAREA_SQMI>
          <LATITUDE>38.9806</LATITUDE>
          <LONGITUDE>76.9372</LONGITUDE>
          <AREACODE>301</AREACODE>
          <TIMEZONECODE>5</TIMEZONECODE>
          <TIMEZONENAME>Eastern</TIMEZONENAME>
          <MILES>0</MILES>
        </ZipCodeInfo>
        <ZipCodeInfo diffgr:id="ZipCodeInfo3" msdata:rowOrder="2">
          <ZIPCODE>20742</ZIPCODE>
          <CITY>COLLEGE PARK</CITY>
          <STATE>MD</STATE>
          <FIPS>033</FIPS>
          <COUNTY>PRINCE GEORGES</COUNTY>
          <COUNTYAREA_SQMI>486</COUNTYAREA_SQMI>
```

```
            <LATITUDE>38.9806</LATITUDE>
            <LONGITUDE>76.9372</LONGITUDE>
            <AREACODE>301</AREACODE>
            <TIMEZONECODE>5</TIMEZONECODE>
            <TIMEZONENAME>Eastern</TIMEZONENAME>
            <MILES>0</MILES>
          </ZipCodeInfo>
        </ZipCodeData>
      </diffgr:diffgram>
    </ListByCityResult>
    </ListByCityResponse>
</soap:Body>

</soap:Envelope>
```

Listing B.2: SOAP Nachricht der Service Antwort

B.1.3 OWL-S Grounding

Das OWL-S Grounding hat die Aufgabe, die Rückgabe Nachricht (SOAP bzw. XML Instanz) des implementierten Web Service auf die Instanz des durch den AtomicProcess definierten Rückgabeparameter zu mappen. Der ensprechende Aufruf der OWL-S API und das benötigte WSDL Grounding sehen Sie in diesem Abschnitt.

```
    public void runZipCode() throws Exception {
      service = reader.read(URI.create(
        "http://www.mindswap.org/2004/owl-s/1.0/ZipCodeFinder.owl"));
      process = service.getProcess();

      // initialize the input values to be empty
      values = OWLSFactory.createValueMap();

      values.setValue(process.getInputs().getParameter("City"), "College Park");
      values.setValue(process.getInputs().getParameter("State"), "MD");
      values = exec.execute(process, values);

      // get the output param using the index
      outValue = values.getValue(process.getOutputs().outputAt(0)).toString();

      // display the results
      System.out.println("Executed service '" + service + "'");
      System.out.println("City   = " + "College Park");
      System.out.println("State  = " + "MD");
      System.out.println("Output = ");
      System.out.println(Utils.formatRDF(outValue));
      System.out.println();
    }
```

Listing B.3: ZipCodeFinder. OWL-S Aufruf.

XSL Transformation

```
<grounding:WsdlAtomicProcessGrounding rdf:ID="ZipCodeFinderProcessGrounding">

    <!-- snip -->

    <grounding:wsdlOutputMessage rdf:datatype="&xsd;#anyURI">
        http://www.tilisoft.com/ws/LocInfo/ListByCitySoapOut
    </grounding:wsdlOutputMessage>

    <grounding:wsdlOutput>
        <grounding:WsdlOutputMessageMap>
            <grounding:owlsParameter rdf:resource="#ZipCode"/>
            <grounding:wsdlMessagePart rdf:datatype="&xsd;#anyURI">
                http://www.tilisoft.com/ws/LocInfo/ListByCityResult
            </grounding:wsdlMessagePart>
            <grounding:xsltTransformationString>
            <![CDATA[
<xsl:stylesheet version="1.0"       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                                    xmlns:ns="http://www.tilisoft.com/ws/LocInfo/DataObjects/ZipCodeData.xsd">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/">
        <xsl:variable name="X1" select="//ns:ZipCodeData/ns:ZipCodeInfo /ns:ZIPCODE"/>
        <rdf:RDF    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                    xmlns:zipcode-ont="http://www.daml.org/2001/10/html/zipcode-ont#">
            <zipcode-ont:ZipCode>
                <zipcode-ont:zip><xsl:value-of select="$X1"/></zipcode-ont:zip>
            </zipcode-ont:ZipCode>
        </rdf:RDF>
    </xsl:template>
</xsl:stylesheet>
            ]]>
            </grounding:xsltTransformationString>
        </grounding:WsdlOutputMessageMap>
    </grounding:wsdlOutput>

</grounding:WsdlAtomicProcessGrounding>
```

Abbildung B.1: WSDL AtomicProcess-Grounding für den Output Parameter

Anhand der Grounding Transformation in Abbildung B.1 und des Java Aufrufs wird genau ein OWL Individual (`outValue`) erzeugt.

Beispiel einer OWL Instanz:

```
<rdf:RDF
 xmlns:ns="http://www.tilisoft.com/ws/LocInfo/DataObjects/ZipCodeData.xsd"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:zipcode-ont="http://www.daml.org/2001/10/html/zipcode-ont#">
<zipcode-ont:ZipCode>
<zipcode-ont:zip>20741</zipcode-ont:zip>
</zipcode-ont:ZipCode>
</rdf:RDF>
```

Anhang C

Die OWLS-TC Testcollection

Die *Semantic Web Services* aus der Testcollection referenzieren ein Set von Ontologien, die aus mehreren unterschiedlichen Quellen stammen:

- SUMO ontology, Projekt: `http://reliant.teknowledge.com`
- Larflast ontology, Projekt: `http://www-it.fmi.uni-sofia.bg/larflast/`
- Wine und food ontology (W3C), Projekt: `http://www.w3.org`
- AKT Reference ontology v2, Projekt: `http://www.aktors.org`
- University ontology, `http://www.lehigh.edu/~zhp2/`
- Travel ontology, tourism tutorial, `http://protege.standford.edu`
- Health ontology, `http://www.dfki.de/scallops/`
- Simple Book ontology, DFKI
- Simplified SUMO ontology

Für die Entwicklung des OWL-Parsers im Rahmen der Translation *OWL nach XML Schema* war es Voraussetzung, kompatibel zu diesen Ontologien und den dort definierten OWL-Klassen zu sein (OWL-DL). Sollten künftig weitere Ontologien in die Testkollektion aufgenommen werden, die nicht direkt gelesen werden können, muss der OWL-Parser gemäß dem Vorgehensmodell 4.1 erweitert werden.

Untersuchung der OWLS-TC

Im Rahmen dieser Arbeit mussten alle Ontologien und OWL-S Service Beschreibungen der Testkollektion betrachtet werden. Bei Untersuchung der Ontologien fielen folgende Dinge auf:

- geoCoordinateSystems20040307.owl und geoFeatures20040307.owl: Fehler bei der Referenzierung von Eigenschaften (Namespace Prefix)
- Mit *Protégé* ist es nicht möglich ist, ein DatatypeProperty anzulegen, das über rdfs:range eine OWL Klasse (wie z.B. NonnegativeInteger) referenziert.

Anhang D

Abkürzungsverzeichnis

ATHENA	Advandced Technologies for interoperability of Heterogenous Enterprise Networks and their Applications (IST-507849)
BPEL	Business Process Execution Language
DAML	DARPA Agent Markup Language (2000)
ERP	Enterprise Resource Planning
IST	Information Society Technologies
MDA	Model Driven Architecture
OIL	Ontology Inference Layer (1997)
OMG	Object Management Group
OWL	Web Ontology Language (2004)
RDF	Resource Description Framework
RDFS	RDF Schema
SHOE	Simple HTML Ontology Extensions (1995)
SWRL	Semantic Web Rule Language (2004)
SAWSDL	Semantic Annotations for WSDL
SOA	Service-Orientierte Architektur
SOAP	Simple Object Access Protocol
SUMO	Suggested Upper Merged Ontology
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WSDL	Web Service Description Language

Anhang E

Literaturverzeichnis

[AH04] ANTONIOU, Grigoris ; VAN HARMELEN, Frank: *A Semantic Web Primer (Cooperative Information Systems)*. The MIT Press, April 2004. – ISBN 0262012103

[AK03] ATKINSON, Colin ; KÜHNE, Thomas: Model-Driven Development: A Metamodeling Foundation. In: *IEEE Softw.* 20 (2003), Nr. 5, S. 36–41. – ISSN 0740–7459

[AMM+04] ANKOLEKAR, Anupriya ; MARTIN, David ; MCGUINNESS, Deborah ; MCILRAITH, Sheila ; PAOLUCCI, Massimo ; PARSIA, Bijan. *OWL-S' Relationship to Selected Other Technologies*. http://www.w3.org/Submission/2004/SUBM-OWL-S-related-20041122/. 2004

[AS05] ALESSO, H.Peter ; SMITH, Craig F.: *Developing Semantic Web Services*. A K Peters, Ltd., 2005

[BF06] BENEDIKT FRIES, Patrick K.: *OWLS-MX Hybrid Semantic Web Service Matchmaker For Services in OWL-S*. Saarbrücken, Germany: DFKI GmbH, 2006

[BL98] BERNERS-LEE, Tim: Semantic Web Road map / W3C. 1998. – Forschungsbericht

[BL04] BALZER, Steffen ; LIEBIG, Thorsten: Bridging the Gap Between Abstract and Concrete Services - A Semantic Approach for Grounding OWL-S. In: *Proceedings of the Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications*. Hiroshima, Japan : CEUR Workshop Proceedings, November 2004, S. 16–30

[BLHL01] BERNERS-LEE, T. ; HENDLER, J. ; LASSILA, O.: The Semantic Web. In: *Scientific American* (2001), May

[Bou01] BOURRET, Ronald: Mapping W3C Schemas to Object Schemas to Relational Schemas. In: *http://www.rpbourret.com* (2001)

[CDM+04] CABRAL, Liliana ; DOMINGUE, John ; MOTTA, Enrico ; HAKIMPOUR, Farshad ; PAYNE, Terry: Approaches to Semantic Web Services: An Overview and Comparisons. In: *Proceedings of European Semantic Web Conference* (2004)

[Dän05] DÄNZER, Michael: *NExT - The NMR EXperiment Toolbox*, University of Zürich, Diplomarbeit, 2005

[DOS03] DACONTA, Michael C. ; OBRST, Leo J. ; SMITH, Kevin T.: *The Semantic Web*. Wiley Publishing, Inc., 2003

[EP04] EVRENSIRIN ; PARSIA, Bijan: The OWL-S Java API / MINDSWAP Research Group, University of Maryland, College Park, MD. 2004. – Forschungsbericht

[Erl05] ERL, Thomas: *Service-Oriented Architecture*. Crawfordsville, IN : Prentice Hall, 2005. – ISBN 0–13–185858–0

[FKG+06] FRIESEN, Andreas ; KRAUTH, Péter ; GOUVAS, Panagiotis ; KEREKES, József ; BOURAS, Athanasios. *Business process fusion based on Semantically-enabled Service-oriented Business Applications*. http://www.fusionweb.org. 2006

[FP07] FRICKE, Olaf ; PROTT, Karl: Castors neuer Halbbruder. In: *Javamagazin 4.07* (2007)

[FZT04] FERDINAND, Matthias ; ZIRPINS, Christian ; TRASTOUR, D.: Lifting XML Schema to OWL. In: KOCH, Nora (Hrsg.) ; FRATERNALI, Piero (Hrsg.) ; WIRSING, Martin (Hrsg.): *Web Engineering - 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004, Proceedings*, Springer Heidelberg, 2004, S. 354–358

[Gru94] GRUBER, Thomas R.: A Translation Approach to Portable Ontology Specifications / Knowledge System Laboratory, Stanford University. 1994. – Forschungsbericht

[GT04] GANNOD, Gerald C. ; TIMM, John T.: An MDA-based Approach for Facilitating Adoption of Semantic Web Service Technology. In: *Proceeding of the IEEE EDOC Workshop on Model-Driven Semantic Web (MDSW 04)*, 2004

[HSH03] HORROCKS, I. ; SCHNEIDER, Patel P. ; VAN HARMELEN, F.: From SHIQ and RDF to OWL: The making of a web ontology language. In: *Journal of Web Semantics* 1 (2003), Nr. 1, S. 7–26

[KFHH] KLEIN, M. ; FENSEL, D. ; VAN HARMELEN, F. ; HORROCKS, I. *The Relation between Ontologies and Schema-languages: Translating OIL-specifications in XML-Schema*. In Proceedings of the Workshop on Applications of Ontologies and Problem-Solving Methods, 2000. http://delicias.dia.fi.upm.es/WORKSHOP/ECAI00/7.pdf.

[KFNM04] KNUBLAUCH, Holger ; FERGERSON, Ray W. ; NOY, Natalya F. ; MUSEN, Mark A.: The Protégé OWL plugin: An Open Development Environment for Semantic Web Applications. In: *in press* (2004)

[KG03] KEVIN GIBBS, Elias T. *Create Web services using Apache Axis and Castor*. http://www-128.ibm.com/developerworks/webservices/library/ws-castor/. 2003

[KG06] KLUSCH, Matthias ; GERBER, Andreas: Fast Composition Planning of OWL-S Services and Application. In: *ecows* 0 (2006), S. 181–190. ISBN 0–7695–2737–X

[KK06] KAUFER, Frank (Hrsg.) ; KLUSCH, Matthias (Hrsg.): *OWL-S Matchmaker, OWL-S/UDDI, OWLS-MX*. 2006

[KPBP04] KALYANPUR, Aditya ; PASTOR, Daniel J. ; BATTLE, Steve ; PADGET, Julian A.: Automatic Mapping of OWL Ontologies into Java. In: MAURER, Frank (Hrsg.) ; RUHE, Günther (Hrsg.): *SEKE*, 2004. – ISBN 1–891706–14–4, S. 98–103

[KS96] KHAN, Ayub ; SUM, Marina: Introducing Design Patterns in XML Schemas. In: *Sun Developer Network(SDN)* (1996)

[MBL+03] MARTIN, David ; BURSTEIN, Mark ; LASSILA, Ora ; PAOLUCCI, Massimo ; MCILRAITH, Terry Payne S. *Describing Web Services using OWL-S and WSDL*. http://www.daml.org/services/owl-s/1.0/owl-s-wsdl.html. 2003

[MPM+04] MARTIN, David ; PAOLUCCI, Massimo ; MCILRAITH, Sheila ; BURSTEIN, Mark ; MCDERMOTT, Drew ; MCGUINNESS, Deborah ; PARSIA, Bijan ; PAYNE, Terry ; SABOU, Marta ; SRINIVASAN, Monika Solanki N. ; SYCARA, Katia: Bringing Semantics to Web Services: The OWL-S Approach / various institutions. 2004. – Forschungsbericht

[PNL05] PANTSCHENKO, Konstantin ; NOPPENS, Olaf ; LIEBIG, Thorsten. *Grounding Web Services Semantically: Why and How?* W3C Workshop on Frameworks for Semantics in Web Services, Innsbruck, Austria. 2005

[Pol05] POLLERES, Axel. *Current Efforts towards Semantic Web Services (SWS): OWL-S and WSMO*. WSMO working group, http://www.wsmo.org. 2005

[POSV04] PATIL, Abhijit ; OUNDHAKAR, Swapna ; SHETH, Amit ; VERMA, Kunal: METEOR-S Web Service Annotation Framework / LSDIS Lab, Department of CS, University of Georgia, 415 GSRC, Athens, GA 30602. 2004. – Forschungsbericht

[PS04] PELTZ, Chris (Hrsg.) ; SECRIST, Mark (Hrsg.): *Using XML schemas effectively in WSDL design*. HP, 2004

[PSH04] PATEL-SCHNEIDER, Peter F. ; HORROCKS, Ian. *OWL Web Ontology Language, Semantics and Abstract Syntax, Section 2. Abstract Syntax*. http://www.w3.org/TR/2004/REC-owl-semantics-20040210/syntax.html. 2004

[PSSN03] PAOLUCCI, Massimo ; SRINIVASAN, Naveen ; SYCARA, Katia ; NISHIMURA, Takuya: Towards a Semantic Choreography of Web Services: from WSDL to DAML-S. In: *First International Conference on Web Services, ICWS '03*, CSREA Press, 2003, S. 22–26

[Sch05] SCHATTAT, Christian: *Konzepte zur Strukturierung von Inhalten von Lehrveranstaltungen*, Otto-von-Guericke-Universität Magdeburg, Diplomarbeit, 2005

[SPAS03] SYCARA, Katia ; PAOLUCCI, Massimo ; ANKOLEKAR, Anupriya ; SRINIVASAN, Naveen: Automated discovery, interaction and composition of Semantic Web services. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 1 (2003), December, Nr. 1, S. 27–46

[SPH04] SIRIN, Evren ; PARSIA, Bijan ; HENDLER, James: Filtering and Selecting Semantic Web Services with Interactive Composition Techniques. In: *IEEE Intelligent Systems* 19 (2004), Nr. 4, S. 42–49. – ISSN 1541–1672

[Syc06] SYCARA, Katia. *Tools and Technologies for Semantic Web Services: An OWL-S Perspective*. http://www.ai.sri.com/daml/services/owl-s/1.2/OWL-S-walkthru.html. 2006

[SYWZ05] SHEN, Jun ; YANG, Yun ; WAN, Chengang ; ZHU, Chuan: From BPEL4WS to OWL-S: Integrating E-Business Process Descriptions. In: *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*. Washington, DC, USA : IEEE Computer Society, 2005. – ISBN 0–7695–2408–7–01, S. 181–190

[TG05] TIMM, John T. ; GANNOD, Gerald C.: A Model-Driven Approach for Specifying Semantic Web Services. In: *3rd IEE International Conference on Web Services (ICWS); ASU, IEEE; Arizona State University* (2005)

[VBE+] VAYSSIÈRE, Julien ; BENGURIA, Gorka ; ELVESÆTER, Brian ; FISCHER, Klaus ; ZINNIKUS, Ingo. *Rapid Prototyping for Service-Oriented Architectures*

[VBE+06] VAYSSIÈRE, Julien ; BENGURIA, Gorka ; ELVESÆTER, Brian ; FISCHER, Klaus ; ZINNIKUS, Ingo: Rapid Prototyping for Service-Oriented Architectures. In: *2nd Workshop on Web Services Interoperability, Bordeaux, France, 2006; proceedings* (2006)

[YDY+07] YUE, P. ; DI, L. ; YANG, W. ; YU, G. ; ZHAO, P: Semantics-based automatic composition of geospatial Web service chains. In: *Computers and Geosciences* 33 (2007), S. 649–665

[ZGBFV06] ZINNIKUS, Ingo ; GORKA BENGURIA, Brian E. ; FISCHER, Klaus ; VAYSSIERE, Julien: *Multiagent System Technologies (4th German Conference, Mates 2006, Erfurt, Germany, September 19-20)*. Springer, 2006. – ISBN 978–3–540–45376–5

[ZRF05] ZINNIKUS, Ingo ; RUPP, Hans J. ; FISCHER, Klaus: Detecting Similarities between Web Service Interfaces: The WSDL Analyzer / German Research Center for Artificial Intelligence.Deduction and Multiagent Systems Group. 2005. – Forschungsbericht

Index