

# **ConQo**

Dynamische Kontext- und Quality-of-Service-sensitive  
Dienstauswahl für Web-Services

Bastian Buder

27. April 2010



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
1.1	Motivation . . . . .	3
1.2	Zielsetzung . . . . .	3
1.3	Aufbau dieser Dokumentation . . . . .	4
<b>2</b>	<b>Grundlagen</b>	<b>5</b>
2.1	Web Services . . . . .	5
2.2	UDDI, WSDL und SOAP . . . . .	5
2.3	Semantik in Web Services . . . . .	6
2.4	Ontologien . . . . .	6
2.5	Dienstgüte / Quality of Service . . . . .	7
2.6	Kontext . . . . .	7
<b>3</b>	<b>Bestehende Ansätze</b>	<b>8</b>
3.1	OWL-S . . . . .	8
3.2	WSDL-S . . . . .	9
3.3	WSMO . . . . .	9
<b>4</b>	<b>Konzept</b>	<b>10</b>
4.1	Anforderungen . . . . .	10
4.2	Ansatz . . . . .	10
4.3	Semantische Beschreibung . . . . .	11
4.3.1	Basisontologien . . . . .	11
4.3.2	Domainontologien . . . . .	12
4.3.3	Semantische Kategorisierung . . . . .	13
4.3.4	Service-Beschreibungen . . . . .	13
4.3.5	Anfragen (Goals) . . . . .	15
4.4	Ergänzende Beschreibung . . . . .	15
4.4.1	SLA Templates . . . . .	15
4.4.2	Sonstige Dokumente . . . . .	15
4.5	Dienstverzeichnis . . . . .	15
4.5.1	Registrierung und Aktualisierung von Diensten . . . . .	15
4.5.2	Dienssuche (Discovery) . . . . .	15
4.5.3	Dienstbewertung (Ranking) . . . . .	16

<b>5</b>	<b>Implementierung</b>	<b>17</b>
5.1	Funktionsweise und Systemkomponenten . . . . .	17
5.1.1	Architektur . . . . .	17
5.1.2	Datenbanken . . . . .	17
5.2	Schnittstellen . . . . .	17
5.2.1	AdminInterface . . . . .	18
5.2.2	ProviderInterface . . . . .	18
5.2.3	ClientInterface . . . . .	19
5.2.4	GrandSlam . . . . .	19
5.2.5	Contract Wizard . . . . .	19
5.3	Client . . . . .	19
5.4	Bibliotheken . . . . .	19
5.4.1	DIP QoS-enabled Service Discovery component . . . . .	19
5.4.2	WSML2Reasoner . . . . .	20
5.4.3	WSMO4J . . . . .	20
5.5	Tools . . . . .	20
5.5.1	EclipseEE . . . . .	20
5.5.2	WSMO Studio . . . . .	21
<b>6</b>	<b>Installation</b>	<b>22</b>
6.1	Voraussetzungen . . . . .	22
6.1.1	Java SE . . . . .	22
6.1.2	Apache Tomcat . . . . .	22
6.2	Ausführung aus Eclipse . . . . .	23
6.2.1	Eclipse . . . . .	23
6.2.2	Tomcat unter Eclipse konfigurieren . . . . .	23
6.2.3	SVN Checkout . . . . .	23
6.2.4	Ausführung . . . . .	23
6.3	Stand-Alone . . . . .	23
6.3.1	SVN Checkout . . . . .	23
6.3.2	Apache Ant . . . . .	23
6.3.3	Projekte deployen . . . . .	23
6.3.4	Updates . . . . .	24
6.4	Koffiguration anpassen . . . . .	24

# Kapitel 1

## Einleitung

### 1.1 Motivation

Das Internet wächst seit seiner Einführung in rasanter Geschwindigkeit, jedes Jahr kommen Unmengen an Webseiten, Anwendungen und Nutzern hinzu. Die Bedeutung des World Wide Web wird für die Bewältigung täglicher Aufgaben, angefangen bei der Recherche nach Informationen über digitalen Postverkehr und Konferenzen bis hin zur Kommunikation komplexer Software innerhalb und zwischen Unternehmen immer wichtiger. Speziell mit dem letzten Punkt beschäftigt sich ConQo, da eine leistungsstarke und kostengünstige Kommunikationsinfrastruktur - gerade im Business-to-Business - Bereich immer wichtiger wird um eine hohe Flexibilität zu erreichen und dem hohen Innovations- und Kostendruck von Unternehmen gerecht zu werden. Die verwendeten Technologien haben sich in den letzten Jahren stark weiterentwickelt und gewandelt, so unterstützen inzwischen fast alle wichtigen Programmiersprachen Web Services und damit die Grundlage für die Service Oriented Architecture (SOA). Zahlreiche Anbieter nutzen mittlerweile die Vorteile dieser Technik und die damit verbundenen Vorzüge der Interoperabilität. Mitunter stehen inzwischen umfangreiche Frameworks und Tools bereit die den Entwicklern viel Arbeit abnehmen und damit die Realisierung von Kommunikation zwischen heterogenen Systemen fördern und erleichtern.

### 1.2 Zielsetzung

Web Services besitzen sowohl funktionale als auch nichtfunktionale Eigenschaften, die sich im Wesentlichen darin unterscheiden, dass sie einerseits die technischen Aspekte enthalten die zur Nutzung erforderlich sind und andererseits Merkmale darstellen die die Qualität und Eignung für bestimmte Zwecke widerspiegeln. Gerade die nichtfunktionalen Merkmale spielen inzwischen eine entscheidende Rolle bei der Auswahl von Diensten mit vergleichbarer Funktionalität, werden jedoch bisher kaum berücksichtigt. Ziel ist es, entgegen der meisten aktuellen Suchmechanismen, dynamisch und automatisiert geeignete Web Services anhand von Kontextinformationen und Dienstgüteparametern, also den nichtfunktionalen Eigenschaften, auszuwählen und zu nutzbar zu machen. Der Prototyp soll die Vorteile dieser Art und Weise der Dienstsuche in eine real verwendbare Umgebung umsetzen die es jeder Form von Clients und Nutzern (unabhängig von Programmiersprache und Plattform) ermöglicht diesen unabhängigen Discovery-Service zu nutzen.

### 1.3 Aufbau dieser Dokumentation

Diese Dokumentation soll sowohl für Nutzer als auch für Entwickler geeignet sein, daher sind sicher nicht für jeden alle Bestandteile relevant und sinnvoll. Im zweiten Kapitel werden in sehr kurzer Form die grundlegenden Technologien erklärt und Begriffe definiert. Neben den bereits vielfach Beschriebenen Web Service Technologien, die vermutlich den Meisten bekannt sind, sei an dieser Stelle auf die Definition von Dienstgüteparametern und Kontextdaten hingewiesen. Diese Bezeichnungen werden vielfach unterschiedlich verwendet, so das es hilfreich ist die Verwendung im Zusammenhang mit ConQo zu verstehen. Im dritten Kapitel werden die genutzten Vorarbeiten vorgestellt sowie kurz auf andere Ansätze verwiesen. Daran anschließend wird das Konzept von ConQo vorgestellt. Unter Berücksichtigung eines Gesamtsystems durch Einbeziehung weiterer Komponenten werden hier wesentliche Entscheidungen begründet. Im fünften Kapitel finden sich schließlich Details zur Implementierung gefolgt von einer Installationsanleitung für den Prototypen inklusive Bedienungshinweisen.

# Kapitel 2

## Grundlagen

### 2.1 Web Services

Web Services stellen eine Technologie zur Kommunikation über das Web (oder einer beliebigen anderen Netzwerkform) zwischen Applikationen von verteilten Systemen dar. Man kann sie als wiederverwendbare, gekapselte und selbstbeschreibende Software-Systeme bezeichnen. Die Schnittstellen müssen über eine eindeutige Adresse erreichbar sein und sind typischerweise als XML-Artefakte definiert. Es handelt sich hierbei um eine Client-Server-Architektur, Clients verwenden die von Servern angebotenen Informationen und Dienste. Die Web Services stellen einem Aufrufer also über das Netz eine bestimmte Funktionalität bereit, die dieser unmittelbar in seiner Applikation nutzen kann. Dieser Form des Leistungsangebotes wird vor allem im Business-to-Business-Bereich, unter anderem aufgrund der Heterogenität der Nutzeranforderungen, eine große Zukunft vorhergesagt. Ein solcher Dienst ist dabei nicht zur unmittelbaren Kommunikation mit einem Anwender gedacht sondern zur direkten Interaktion mit anderen Software-Agenten. Die Kommunikation erfolgt im Allgemeinen auf der Anwendungsebene meistens auf Basis von HTTP in Verbindung mit den auf den darunter liegenden Schichten befindlichen Protokollen TCP und IP. Die Akteure von Web Services sind im Wesentlichen der Dienstnutzer, der Dienstanbieter sowie das Dienstverzeichnis.

### 2.2 UDDI, WSDL und SOAP

Die drei auf XML aufbauenden Standards UDDI (Universal Discription, Discovery and Integration), WSDL (Web Service Description Language) und SOAP (Simple Object Access Protocoll) gelten als Hauptsäulen von Web Services.

Bei UDDI handelt es sich selbst um einen Web Service welcher ein Verzeichnis zur Registrierung von anderen Web Diensten zur Verfügung stellt und damit eine zentrale Rolle im Umfeld der dynamischen Web Services spielt. Spezifiziert sind die API zum Abfragen und Publizieren sowie die Beschreibung der verwendeten Datenstrukturen. UDDI beinhaltet globale Informationen zu den registrierten Diensten (z.B. von welchem Unternehmen der Web Service zur Verfügung gestellt wird) sowie eine technische Spezifikation (in der Regel ein Verweis auf die WSDL-Datei). Diese Service-Registry stellt einen Suchmechanismus zur Verfügung, der die Funktionalitäten der registrierten Dienste den potentiellen Nutzern of-

fenlegen soll. Leider, und damit ist einer der größten Nachteile von UDDI genannt, ist diese Suche nur stichwortbasiert.

Bei WSDL (Web Service Description Language) handelt es sich um eine auf XML basierende programmiersprachen-, protokoll und plattformunabhängige Beschreibung eines Webdienstes auf hoher Abstraktionsebene. Enthalten sind in Form einer Menge abstrakter Netzwerkpunkte im wesentlichen die unterstützten Operationen, die Datentypen, die erforderlichen Parameter und Rückgabewerte, die unterstützten Protokolle sowie die Adresse. Wählt ein Nutzer einen für ihn offenbar geeigneten Dienst aus dem UDDI-Verzeichnis, erhält er den Uniform Resource Identifier (URI) der entsprechenden WSDL-Datei und kann auf Basis des SOA-Protokolls die Dienste des Servers nutzen. Problematisch ist, dass keine semantische Beschreibung erfolgen kann. Die Bedeutung der Operationen ist lediglich in externen Textdokumenten definiert oder möglicherweise im Namen verankert. Auch Gütekriterien lassen sich nicht auf sinnvolle Art und Weise modellieren, geschweige denn Dienste anhand solcher suchen und auswählen.

SOAP stellt das Standardprotokoll dar und spezifiziert den konkreten Aufbau von Nachrichten für die Dienstanutzung. Die Basis hierfür sind Remote Procedure Calls (RPC), wobei das Transportprotokoll, mit dem die Nachrichten ausgetauscht werden, nicht festgeschrieben ist. In der Regel wird auf HTTP zurückgegriffen, denkbar wären jedoch ebenfalls FTP, SMTP sowie zahlreiche weitere Protokolle.

Diese grundlegenden Technologien sind bereits vielfach detailliert beschrieben, weshalb an dieser Stelle nicht weiter darauf eingegangen wird.

## 2.3 Semantik in Web Services

Die aktuell übliche Suche (Discovery) von geeigneten Web Services ist aufgrund der stark wachsenden Anzahl von Diensteanbietern zunehmend ineffektiv. Die ausschließlich syntaktischen Suchmechanismen liefern zahlreiche irrelevante Ergebnisse die manuell durch den Nutzer bewertet werden müssen. Beispielhaft lässt sich dies daran verdeutlichen, dass syntaktisch bei der Suche nach dem Begriff "Koch" nicht unterscheidbar ist ob der Beruf Koch, eine Person die diesen Namen trägt oder ein Unternehmen gesucht wird. Computer können zwar das Internet zur Kommunikation, jedoch nur bedingt als Informationsmedium nutzen, da die Inhalte (die Bedeutung von Informationen) nicht verarbeitet werden können und damit unbrauchbar sind. Das Semantic Web und spezieller Semantic Web Services sollen es nun auch Maschinen ermöglichen Informationen zu suchen und dank maschinenverständlicher Formalisierung zu verwerten. Da natürliche Sprache sehr ungenau ist gilt es einen Weg zu finden Informationen derart aufzubereiten, dass sie nicht nur durch den Menschen sondern auch maschinell verarbeitet werden können. Voraussetzung und wichtiger Bestandteil hierzu sind Metadaten (Daten über Daten) und eine standardisierte Sprache mit einheitlichem Vokabular die es Applikationen ermöglicht die Inhalte zu verstehen und auf verdeckte Informationen zu schließen. Dieser Schritt ist dringend notwendig um die inzwischen unüberschaubare Menge von Angeboten zu differenzieren und effizient nutzbar zu machen.

## 2.4 Ontologien

Eine Ontologie stellt eine formale Beschreibung von Gegenständen und deren Beziehungen untereinander dar. Erst hierdurch wird es Anwendungen möglich mit den darin spezifizierten Objekten eindeutig zu arbeiten. Sie stellen die Grundlage dafür dar, von den überwiegend

syntaktischen Beschreibungen über die Art und Weise des Aufrufs eines Services zu einer semantischen Beschreibung der Leistungsfähigkeit des Dienstes zu gelangen. Selbstverständlich wird es jedoch nicht eine sondern zahlreiche verschiedene Ontologien für unterschiedliche Anwendungsbereiche geben die aufeinander abgebildet und verglichen werden müssen. Mittlerweile haben sich einige Ontologiesprachen etabliert und werden stetig weiterentwickelt, häufig verwendet werden beispielsweise OWL und WSML. Die Basis dieser Sprachen sind oft XML-ähnliche Strukturen, welche jedoch für den Menschen nur sehr schwer lesbar sind. Sie stellen eine formale Spezifikation einer Begriffsbildung (Konzeptualisierung) dar und bestehen aus Daten und Regeln, welche Rückschlüsse über die Bedeutung und den Zusammenhang der enthaltenen Elemente zulassen. Später wird genauer auf die Konzepte, Instanzen, Axiome und Relationen eingegangen.

## 2.5 Dienstgüte / Quality of Service

Unter Dienstgüte eines Web Services zählen wir (in der Regel direkt messbare) Eigenschaften des Services selbst, welche unabhängig von der eigentlichen Programmlogik sind. Sie zählen zu den nichtfunktionalen Eigenschaften eines Web Services. Beispiele hierfür sind die Antwortzeit, die Verfügbarkeit oder die Kosten pro Ausführung. Quality-of-Service Parameter lassen sich zum großen Teil domänenübergreifend wiederverwenden, gleichen sich also für die verschiedensten Anwendungsgebiete von Web Services. Es macht mitunter Sinn solche Serviceeigenschaften vertraglich zuzusichern und zu überwachen.

## 2.6 Kontext

Kontext stellt die zweite Form der nichtfunktionalen Eigenschaften dar. Er bezieht sich nach unserer Definition bei Diensten auf Dinge, die durch den Service repräsentiert werden. Bei einem Filehostingdienst können dies zum Beispiel statische Eigenschaften wie die unterstützten Dateiformate oder die maximale Dateigröße sein. Ein Druckdienst hingegen könnte die Auflösung und das Papierformat angeben. An diesem kleinen Beispiel wird schon deutlich das Kontext von Domäne zu Domäne sehr unterschiedlich ausfallen wird und separat definiert werden muß. Diese Daten sind im Unterschied zu Dienstgüteeigenschaften schwer vorhersehbar. Die Herausforderung liegt in einer generischen Definition der Konzepte, so das diese für beliebige Anwendungsfälle erweitert werden können. Ein weiteres Problem stellt die Dynamik einiger Parameter dar. Soll beispielsweise die Länge einer Warteschlange ausgedrückt werden ist eine regelmäßige aktualisierung unabdingbar. Folglich können wir eine weitere Unterteilung in statische und dynamische Kontextdaten vornehmen.

# Kapitel 3

## Bestehende Ansätze

Folgend werden einige Ansätze zur semantischen Beschreibung von Web Services betrachtet und vorgestellt. In der Regel finden hierzu Ontologiesprachen als Grundlage Verwendung. Keiner der folgenden Lösungen vereint bislang ausreichend die parallele Definition und Suche von Quality-of-Service Parametern und Kontextdaten. Für eine detaillierte Beschreibung weiterer Lösungsansätze sei an dieser Stelle auf den parallel entstandenen Beleg [1] von Gergana Stoyanova verwiesen.

### 3.1 OWL-S

OWL-S bzw. sein Vorgänger DAMLS zählt zu einer der ersten ernstzunehmenden Initiativen zur semantischen Annotation von Web Services. Wesentlich ist die Stützung auf den etablierten Web Service Standard WSDL sowie den Semantic Web Standard OWL.

Die Beschreibung eines Dienstes auf Basis von OWL-S setzt sich aus drei Hauptprofilen zusammen:

- **Service Profil**  
Das Service Profile beschreibt in abstrakter Form was ein Service tut, seine Funktionalität und nichtfunktionale Aspekte die zur Auffindung nützlich sein können. Die Ein- und Ausgabeparameter sind als OWL-Konzepte definiert. Weiterhin gibt es eine Zuordnung zu einer Service-Kategorie sowie nichtfunktionale Attribute zur Beschreibung des Anbieters und der Servicequalität.
- **Service Model**  
Im Service Model wird beschrieben WIE ein Dienst seine Funktionalität erreicht. Der Service wird hierbei als Prozess betrachtet und in verschiedene Prozesstypen unterteilt. Es stellt damit eine Modell dar, das zur Komposition von Services verwendet werden kann.
- **Service Grounding**  
Im Service Grounding wird festgehalten wie ein Dienst genutzt und aufgerufen werden kann. Die einzige derzeit definierte Methode ist die Anbindung an WSDL.

Die Dienstsuche kann auf Basis des Service Profile durch Erweiterung des UDDI Verzeichnisdienstes realisiert werden. Einen solchen Ansatz verfolgt beispielsweise der OWL-S Matchmaker.

## 3.2 WSDL-S

WSDL-S ist der wohl leichtgewichtige Ansatz in diesem Vergleich. Er wurde "Bottom-Up", ausgehend von WSDL, entwickelt und bereits im November 2005 von IBM und der University of Georgia beim W3C eingereicht.

Hierbei besteht keine Beschränkung auf eine bestimmte Sprache zur Annotation. Die semantischen Modelle (z.B. OWL Ontologien) bzw. deren Konzepte sind lediglich in der WSDL per URI referenziert. Des Weiteren ist der WSDL Standard um einige zusätzliche Elemente ergänzt worden, die es erlauben die Operationen sowie die Ein- und Ausgabeparameter zu annotieren.

## 3.3 WSMO

WSMO ist ein weiterer leistungsfähiger Ansatz und wurde 2005 als Standard-Vorschlag beim W3C eingereicht. Die WSMO Working Group stellt ein konzeptionelles Modell – die Web Service Modeling Ontology (WSMO), die dazugehörige Semantik und eine formale Sprache für die Beschreibung dieser Semantik – die Web Service Modeling Language (WSML), sowie die Ausführungsumgebung – das Web Service Execution Environment (WSMX) bereit. Die vorwiegend europäische Initiative der EU-Projekte SEKT, DIP und KnowledgeWeb verfolgt ähnliche Grundsätze wie OWL-S, jedoch mit einem anderem Fokus. Wie bei OWL-S stellen auch hier die Ontologien ein wichtiges Element dar. WSMO folgt dem Prinzip der strikten Entkopplung von Entitäten: Servicebeschreibungen, Ontologien und Nutzeranfragen sind unabhängig.

Zur Beschreibung sind vier Elemente relevant:

- **Ontologien**  
Die Ontologien definieren das Vokabular um Services und Nutzeranforderungen eindeutig beschreiben zu können. Sie werden von diesen Beschreibungen referenziert und verwendet. Genutzt wird hierbei die Sprache WSML, welche speziell auf die Anforderungen von WSMO abgestimmt ist und diese formalisiert. Es existieren aktuell fünf verschiedene, unterschiedliche mächtige, Varianten von WSML.
- **Servicebeschreibungen**  
Dienstanbieter definieren die Leistungen Ihrer Services detailliert in Form der Servicebeschreibungen.
- **Goals (Nutzeranforderungen)**  
Der Nutzer definiert, ähnlich einer Servicebeschreibung, seine Anforderungen an den Dienst.
- **Mediatoren**  
Mediatoren dienen der Behandlung von Interoperabilitätsproblemen.

Als Referenzimplementierung dieses konzeptionellen Modells steht WSMX zur Verfügung.

# Kapitel 4

## Konzept

Der folgende Abschnitt stellt ein Konzept vor, welches die Vorteile der kontextsensitiven Dienstausswahl mit denen der Dienstausswahl anhand von QoS-Parametern vereinen soll.

Um ein entsprechendes semantisches Discovery zu erreichen muss ein Dienstverzeichnis geschaffen werden, das neben den funktionalen Eigenschaften der Dienste auch die Metadaten in geeigneter, maschinenlesbarer Form verarbeiten kann und eine entsprechende Schnittstelle zur Suche danach zur Verfügung stellt. Es besteht folglich entweder die Möglichkeit UDDI zu erweitern oder aber gänzlich zu ersetzen. Nutzer (in der Regel Applikationen) können dann ihre Anforderungen mit Dienstgüteparametern und domainspezifischen Kontextdaten übermitteln und erhalten aus einem Repository die für ihn am besten geeigneten Dienste.

### 4.1 Anforderungen

Eine der zentralen Anforderungen bestand darin, möglichst auf vorhandene Standards aufzusetzen um eine einfache Nutzung zu gewährleisten und die Akzeptanz zu steigern. Zusätzlich sollte das System möglichst generisch realisiert werden um eine Erweiterung und Anpassung auf möglichst viele Anwendungsbereiche (Domänen) zu ermöglichen.

Um eine real verwendbare System zu schaffen sind öffentlich zugängliche Schnittstellen nötig. Weiterhin sollte die Deckung der Nutzeranforderungen gewichtet erfolgen können. Ziel ist, eine Liste von potentiell geeigneten Diensten zur Verfügung zu stellen und den Zugang zu den Services zu erleichtern.

### 4.2 Ansatz

Als vielversprechender Ansatz hat sich eine auf WSMO basierende Komponente herauskristallisiert: die "QoS-enabled Service Discovery Component" aus dem DIP-Projekt [4]. Es handelt sich hierbei um einen Prototyp unter der LGPL-Lizenz, welcher als Plugin für die WSMX Ausführungsumgebung konzipiert ist. Es handelt sich im Wesentlichen um eine Java-Bibliothek, einigen Referenz-Ontologien, Service-Beschreibungen und Goals. Die auf WSML basierenden Ontologien dienen primär der Beschreibung von Dienstgüteparametern. Mittels des auf der Konsole testbaren Prototypen lassen sich Anforderungen und Angebote von

QoS-Parametern miteinander vergleichen und Bewerten.

Diese auf Dienstgüte beschränkte Vorarbeit soll um Kontextdaten erweitert werden und in eine real nutzbare Umgebung eingebunden und zugänglich gemacht werden.

### 4.3 Semantische Beschreibung

Wie bereits in Kapitel 2 beschrieben wird zur eindeutigen, maschinenlesbaren Beschreibung ein einheitliches Vokabular benötigt. Um dies zu gewährleisten werden Ontologien verwendet, welche die Definition von Konzepten, Relationen und Instanzen ermöglichen. Die Wahl fiel hierbei aus mehreren Gründen auf WSMML: einerseits ist die Sprache mächtig genug um aktuelle und zukünftige Anforderungen an Service-Beschreibungen zu erfüllen und zusätzlich ist es damit möglich die gut geeigneten Vorarbeit aus dem DIP-Projekt zu nutzen.

Bei der Analyse der potentiellen Suchkriterien für unterschiedliche Anwendungsdomänen hat sich herauskristalisiert, dass sowohl Parameter existieren die für fast jeden Typ von Web Service relevant sind, als auch spezielle Eigenschaften die nur domänenspezifisch eine Rolle spielen.

Es macht folglich Sinn sowohl globale Eigenschaften zu definieren, als auch (aus Sicht der Anwendungsdomäne) lokale. Zur ersten Gruppe lassen sich primär Dienstgütekriterien zählen, darunter beispielsweise die Antwortzeit oder die Kosten pro Ausführung eines Services. Die anwendungsspezifischen Eigenschaften sind hingegen vorwiegend Kontextparameter, beispielsweise die Kategorie eines Hotels oder die verfügbare Auflösung eines Druckdienstes.

Dieser Umstand wird mit Hilfe verschiedener Basisontologien realisiert, die je nach Eignung einzeln oder kombiniert in einer Domäne genutzt werden können.

#### 4.3.1 Basisontologien

Als Basisontologie steht die QoSBase aus dem DIP-Projekt zur Verfügung, hierin befinden sich bereits einige sinnvolle Dienstgüteeigenschaften und Datentypen. Unter Berücksichtigung des Umstandes, dass ConQo durch Kopplung mit einem Monitor auch die Überwachung von Dienstgütekriterien unterstützen soll, wurden die enthaltenen Parameter in zwei Kategorien unterteilt und in neuen Ontologien untergebracht. Auf der einen Seite werden jene Parameter gebündelt, die aus der Ferne von einem Monitor überwachbar sind (z.B. Availability, MaxDownTime) und auf der anderen Seite die Parameter, deren Einhaltung nur dann überprüft werden kann, wenn auf dem System des Anbieters eine entsprechende Installation des Monitors existiert (z.B. CPU- oder Systemlast). Für diese beiden Typen von Parametern wurden die Ontologien RemoteQoSBase und SystemQoSBase geschaffen, die nun bedarfsgerecht eingesetzt werden können.

Ergänzend zur Dienstgüte ist im Rahmen von [1] eine Ontologie entstanden, die grundlegende Konzepte für Kontextparameter legt sowie die ReputationBase zur Kennzeichnung von reputationsrelevanten Eigenschaften. Zur entsprechenden Modellierung der Eigenschaften in den verschiedenen Anwendungsdomänen werden diese Basen in den domänenspezifischen Ontologien, den Servicebeschreibungen sowie den Goals importiert und genutzt.

Listing 4.1: QoSBase.wsml: Definition von Einheiten

```

1 // A measurement unit
2 concept MeasurementUnit

```

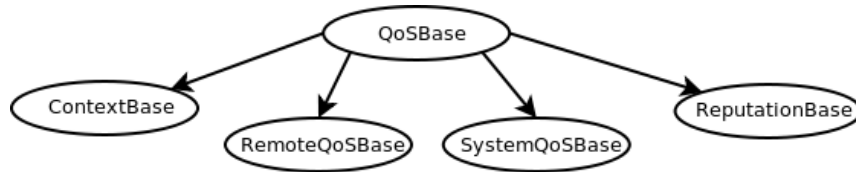


Abbildung 4.1: Basisontologien

```

3     conversionFactor impliesType _double
4
5 concept PercentageUnit subConceptOf MeasurementUnit
6
7 concept CurrencyUnit subConceptOf MeasurementUnit
8
9 instance Euro memberOf CurrencyUnit
10    conversionFactor hasValue 1.5
11
12 instance CHF memberOf CurrencyUnit
13    conversionFactor hasValue 1
  
```

In diesem Auszug der QoSBase wird ein Konzept (vergleichbar mit einer Klasse in der objektorientierten Programmierung) "MeasurementUnit" definiert, dessen Instanzen (vergleichbar mit der Instanz einer Klasse) einen "conversionFactor" vom Typ Double erwarten. Darunter wurden die Einheiten "PercentageUnit" sowie "CurrencyUnit" angelegt, welche Subkonzepte von "MeasurementUnit" sind, also auch die Eigenschaften erben. In den Zeilen 9 und 12 werden wiederum Instanzen von "CurrencyUnit" erstellt, welche einen "conversionFactor" angeben müssen. Dieser Faktor dient später zum Umrechnen der Einheiten untereinander. Es ist damit zum Beispiel möglich das ein Dienstanbieter, der seinen Preis z.B. in Schweizer Franken angegeben hat, trotzdem von einem Nutzer gefunden wird, der nach einem maximalen Ausführungspreis in Euro sucht.

### 4.3.2 Domainontologien

Als Domainontologien werden im Rahmen von ConQo die Ontologien bezeichnet, die lediglich für einen bestimmten Anwendungsbereich relevant sind. Eine derartige Ontologie ist beispielsweise speziell auf die Bedürfnisse von Druckdiensten oder Filehostingdiensten zugeschnitten. Am Beispiel von Hotels und Druckdiensten wird hierbei auch die Relevanz von der verschiedenen Basisontologien deutlich: Während bei Hotelbuchungen Kontexteigenschaften (z.B. Hotelkategorie, Raucherzimmer, ...) wesentliche Parameter darstellen erscheint es nicht sinnvoll über einen Monitor QoS-Parameter zu ermitteln. Im Gegensatz hierzu kann bei Druckdiensten die Antwortzeit oder die Länge der Warteschlange durchaus relevant sein. Entsprechend dieser unterschiedlichen Anforderungen finden in den Domänen auch nur einige der zur Verfügung stehenden Basisontologien Verwendung.

Folgend ist ein Auszug aus der Domainontologie für Druckdienste dargestellt. Im gelisteten Abschnitt ist die Definition des Kontextparameters "Resolution" und der zugehörigen Einheit sichtbar, welche zum Ausdruck gebotener und benötigter Auflösungen dient.

Listing 4.2: Auszug PrinterContextQoSBase.wsm1

```

1 concept DpiUnit subConceptOf qos#MeasurementUnit
2
  
```

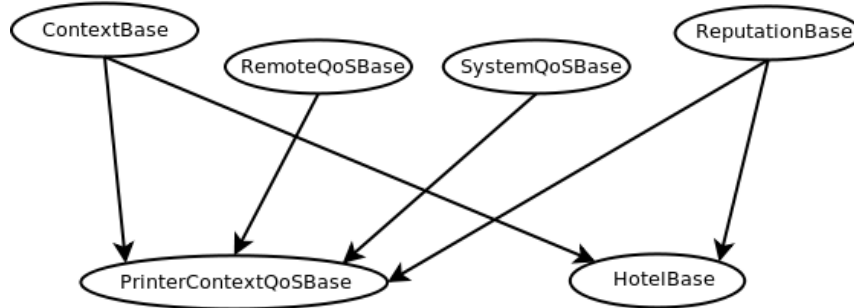


Abbildung 4.2: Basisontologien

```

3 instance Dpi memberOf DpiUnit
4     qos#conversionFactor hasValue 1.0
5
6 concept Resolution subConceptOf {qos#Quality, qos#HigherBetter}
7     qos#unit impliesType DpiUnit
8
9 instance ResolutionRange memberOf qos#QualityRange
10    qos#item hasValue Resolution
11    qos#min hasValue 0
12    qos#max hasValue 3000
13    qos#defaultUnit hasValue Dpi
  
```

### 4.3.3 Semantische Kategorisierung

Aufgrund der Definition der Anforderungen spezieller Web Service - Typen in Domänen lässt sich auf dieser Basis auch eine grobe funktionale Zuordnung der Dienste vornehmen. Hintergrund dieser Zuordnung ist dabei nicht auf welche Art und Weise eine entsprechend eingeordneter Dienst verwendet werden kann, sondern was er im Wesentlichen leistet.

Um die Dienstsuche auch bei vielen, auf mehrere Domänen verteilten, registrierten Services zügig durchführen zu können sind die Dienste entsprechend kategorisiert. Die jeweilige Kategorie wird sowohl in den Domainontologien, als auch in den Servicebeschreibungen und Goals angegeben. Auf Basis dieser Vorsortierung kann ein großteil irrelevanter Dienste bei einer Dienstsuche bereits ausgeschlossen werden. Der daraus resultierende Vorteil besteht darin, dass der verhältnismäßig aufwendige Vergleich der nichtfunktionalen Parameter sowie das Ranking nur für Services erfolgen muss, die auch die gewünschte Grundfunktionalität bieten.

### 4.3.4 Service-Beschreibungen

Dienstanbieter müssen ihr Angebot möglichst detailliert beschreiben damit die Eigenschaften des Dienstes mit den Nutzeranforderungen gematcht werden kann. Hierzu werden in einer Dienstkategorie ausschließlich die Konzepte der der Basis- und entsprechenden Domainontologie verwendet. Neben den Angeboten des Services ist es ebenfalls möglich Anforderungen an den Client zu stellen um die zugesicherten Qualitätsmerkmale einhalten zu können. So ist beispielsweise bei einem Filehostingdienst denkbar, dass er eine gewisse Bandbreite

vom Nutzer fordert um die angebotene Downloadrate überhaupt gewährleisten zu können. Unterschieden werden Angebot und Forderung über die Konzepte `serviceRequirement` und `serviceSpec`. Die Instanzen einer Parameter erben von mindestens einer dieser beiden Konzepte.

Das folgende Listing zeigt eine vollständige Servicebeschreibung. Der aufgeführte Dienst bietet genau ein Interface an, für welches zwei nichtfunktionale Parameter beschrieben sind. Einerseits ist eine maximale Zeitspanne für die Nichterreichbarkeit angegeben ("MaxDowNTime") und andererseits ein Preis je Ausführung von 2 EUR gefordert. Gut erkennbar ist hierbei die unterschiedliche Interpretation der Parameter gesteuert durch den Einsatz der Konzepte "ServiceSpec" und "ServiceRequirement".

Listing 4.3: Servicebeschreibung

```

1 wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
2
3 namespace { _"http://localhost:8080/Matchmaker/ontologies/Generic/Service0.wsml#",
4             qos _"http://localhost:8080/Matchmaker/ontologies/QoSBase.wsml#",
5             gen _"http://localhost:8080/Matchmaker/ontologies/Generic/GenericQoSBase.wsml#",
6             remoteqos _"http://localhost:8080/Matchmaker/ontologies/RemoteQoSBase.wsml#",
7             systemqos _"http://localhost:8080/Matchmaker/ontologies/SystemQoSBase.wsml#" }
8     dc _"http://purl.org/dc/elements/1.1#",
9     wsml _"http://www.wsmo.org/wsml/wsml-syntax#" }
10
11 webService GenericService1
12     importsOntology { _"http://localhost:8080/Matchmaker/ontologies/Generic/GenericQoSBase.wsml#" }
13
14 /* Define Service Class */
15
16
17 capability ServiceCapability
18     postcondition definedBy ?serviceType memberOf gen#Generic.
19
20
21 /* Define Interfaces */
22
23 interface A_Param1
24     importsOntology { Param1 }
25
26     nfp
27         // monitor connection settings
28         monitorendpoint hasValue "http://localhost:8060/axis/services/ConQoAccess"
29     endnfp
30
31
32 /* Define the Ontologies */
33
34 ontology Param1
35     importsOntology { _"http://localhost:8080/Matchmaker/ontologies/Generic/GenericQoSBase.wsml#" }
36
37     instance c1 memberOf { remoteqos#MaxDowNTime, qos#ServiceSpec }
38         qos#value hasValue 10
39         qos#unit hasValue qos#Second
40
41     instance e1 memberOf { gen#Price, qos#ServiceRequirement }
42         qos#value hasValue 2
43         qos#unit hasValue qos#Euro

```

### 4.3.5 Anfragen (Goals)

Analog zur Modellierung von Angeboten werden auch die Anforderungen der Nutzer definiert. Bei den sogenannten Goals entfällt jedoch die Angabe verschiedener Interfaces.

## 4.4 Ergänzende Beschreibung

Da in der beschriebenen semantischen Beschreibung neben nichtfunktionalen Eigenschaften lediglich eine grobe funktionale Zuordnung erfolgt sind zur vollständigen Spezifikation eines Dienstes weitere Dokumente notwendig. Es erscheint an dieser Stelle sinnvoll die Dokumententypen nicht explizit vorzugeben. Speziell für die wichtigsten funktionalen Eigenschaften wird jedoch i.d.R. zumindest eine WSDL beigefügt. Diese zusätzlichen Beschreibungen werden hierbei sinnvollerweise den Interfaces der Servicebeschreibungen zugeordnet und nicht den Services selbst. Dies hat zur Folge das für z.B. für jede "Qualitätsstufe" eines Dienstes eine separate WSDL zugeordnet werden kann.

### 4.4.1 SLA Templates

Eine Sonderrolle nehmen Service Level Agreement Templates ein. Sie dienen als Grundlage für eine vertragliche Zusicherung offerierter Güteeigenschaften. Die entsprechenden Dokumententypen werden explizit definiert und erhalten eine Sonderbehandlung. Wird ein solches Dokument einem Interface zugeordnet können Parameter aus WSMML, die auf die Parameter im SLA Template gemappt werden können durch einen Monitor überwacht werden. Zusätzlich besteht die Möglichkeit auf Basis eines solchen Templates (durch eine externe Komponente) einen Vertrag auszuhandeln.

### 4.4.2 Sonstige Dokumente

Neben den SLA Templates sind quasi beliebige weitere Dokumente denkbar und können dem Dienst zugeordnet werden. Denkbar sind beispielsweise GUIDD, RDF und viele mehr. Nutzern kann später die Gelegenheit gegeben werden Dokumententypen vorauszusetzen.

## 4.5 Dienstverzeichnis

### 4.5.1 Registrierung und Aktualisierung von Diensten

Die Registrierung von Diensten erfolgt allein auf Basis von WSMML Servicebeschreibungen in denen eine Domainontologie referenziert wird. Für Dienste die keine der vorgegebenen Domänen zuzuordnen sind ist eine generische Domain definiert, die lediglich grundlegende Eigenschaften enthält. Die Aktualisierung eines Dienstes kommt der Neuregistrierung gleich. Anhand des eindeutigen Namensraumes wird der bereits registrierte Dienst überschrieben und durch den neuen Ersetzt. Hierbei werden alle registrierten Interfaces berücksichtigt. Die vollständige Löschung erfolgt ebenfalls anhand des Namespaces.

### 4.5.2 Dienssuche (Discovery)

Die Dienstsuche ist der Komplexeste Teil von ConQo. Die gewünschten Kriterien werden wie im Abschnitt Ontologien erklärt in Form eines Goals auf Basis der hinterlegten Domai-

nontologien beschrieben und als Anforderung übermittelt. Um eine bessere Skalierbarkeit zu gewährleisten erscheint es sinnvoll die Suche in drei Schritten durchzuführen. Registriert sind eine Menge von Interfaces, die jeweils genau einer Servicebeschreibung untergeordnet sind. Diese Servicebeschreibungen sind semantisch klassifiziert (siehe Abschnitt "Semantische Kategorisierung"). Die erste Filterung dieser Menge erfolgt auf Basis dieser semantischen Kategorien und stellt die funktionale Filterung dar. Nach diesem ersten Schritt verbleiben nur noch Services die sich in der richtigen Domäne befinden (z.B. nur Filehosting oder nur Druckerdienste). Nachdem die Menge durch diesen Schritt auf ein Minimum reduziert wurde, wird zunächst geprüft welche der Interfaces die "Mandatory"-Parameter beschreiben und vom Wert erfüllen. Es handelt sich dabei um Dienstgüte oder Kontexteigenschaften, von denen der Service-Requester explizit voraussetzt das sie angegeben sind und erfüllt werden. Die Interfaces die dies nicht gewährleisten werden bereits hier aussortiert und nicht weiter betrachtet da sie für den Nutzer ohnehin nicht relevant sind. Da es mitunter hilfreich sein kann zu wissen warum ein Suche kaum oder gar kein Ergebnis hervorbringt werden diese Dienste jedoch ebenfalls unter Angabe des nichterfüllten Parameters gelistet und ausgegeben. Im nächsten Schritt erfolgt die Prüfung der Environment-Parameter. Dies sind Forderungen des Dienstanbieters an den Nutzer, die benötigt werden um die offerierten Qualitätseigenschaften zu gewährleisten. Die nun verbleibende Menge ist in der Regel sehr klein und kann dem Ranking zugeführt werden. Das Ranking ist der Ressourcenintensivste Part des Discovery. Es ist daher wichtig das alle ungeeigneten Dienste bereits vorher möglichst vollständig aussortiert wurden.

### 4.5.3 Dienstbewertung (Ranking)

Ein Ranking ist auf zahlreiche Arten realisierbar. Da es sich bei der Menge der Dienste, die das Ranking durchlaufen lediglich um eine Restmenge von Angeboten handelt, die für den Nutzer geeignet sind und zumindest seinen minimalen Anforderungen entsprechen geht es primär darum die Web Services untereinander zu vergleichen und entsprechend der Nutzerbewertungen zu sortieren.

Hierzu gibt es bereits in der DIP-Komponente eine Art "Punktsystem". Je mehr "Punkte" ein Dienst, oder genauer ein Interface erreicht desto besser ist es aus Sicht der aktuellen Anfrage. Zur Gewichtung wird im Goal jedem nichtfunktionalen Parameter eine Gewichtung in Form eines Integerwertes zugewiesen.

Im Ranking wird nun jeder Parameter eines Interfaces mit den Äquivalenten der anderen Interfaces verglichen. Ist der Parameter des aktuellen Interfaces "besser" als die der Konkurrenz wird ein Punkt vergeben. "Besser" kann, je nach Art des Parameters, sowohl kleiner als auch größer bedeuten. Im Zuge der Monitoring-Erweiterung wurde dieser Algorithmus dahingehend erweitert, dass weitere Punkte vergeben werden wenn die angegebenen Parameter überwachbar sind und eingehalten werden. Die aufwendigen Vergleiche jedes Parameters mit jedem Interface in der Menge führt erwartungsgemäß zu einer beschränkten Skalierbarkeit. Sofern in einem späteren Produktiveinsatz große Ergebnismengen zu erwarten sind wird es unter Umständen erforderlich an dieser Stelle effizientere Algorithmen einzusetzen.

# Kapitel 5

## Implementierung

Die Arbeit besteht im Wesentlichen aus zwei Teilen. Zum Einen sind die semantischen Beschreibungen der Web Services und der Anfragen sowie der dazu benötigten Ontologien für die Dienstgüteeigenschaften und Kontextdaten zu erstellen und erweitern. Zum Anderen ist dies eine Ausführungsumgebung, welche als zentrale Komponente zwischen Web Service Providern und Requestern steht. Die Implementierung stützt sich in weiten Teilen auf die "QoS-enabled Service Discovery Component" des DIP-Projektes. Aufgrund der sehr eingeschränkten Dokumentation und des beachtlichen Umfangs stellt die Einarbeitung durchaus eine Herausforderung dar. Für die Nutzung und Erweiterung der Ontologien um Kontextdaten ist ein exaktes Verständnis der einzelnen Matching-Schritte notwendig war. Weiterhin sind die Anfragen an den Matchmaker relativ komplex, so dass eine Lösung zur automatischen Generierung dieser geschaffen werden muss.

### 5.1 Funktionsweise und Systemkomponenten

#### 5.1.1 Architektur

ConQo besteht aus zwei wesentlichen Elementen. Dies ist einerseits das ServiceManagement und andererseits die DiscoveryComponent. Beide Komponenten arbeiten auf einer Datenbank in der alle relevanten Daten gesammelt verwaltet werden. Für die Kommunikation nach außen stehen SOAP-Schnittstellen zur Verfügung die an die jeweilige Anforderungen von Dienstanutzern- und Dienst Anbietern angepasst sind.

#### 5.1.2 Datenbanken

ConQo ist dank JDBC für verschiedene Datenbanksysteme ausgelegt. Die Entwicklung erfolgte auf MySQL 5, eine Umstellung auf PostgreSQL oder andere DBMS ist via JDBC mit geringem Aufwand realisierbar.

### 5.2 Schnittstellen

Zur Nutzung von ConQo sind mehrere Schnittstellen vorgesehen. Die Unterteilung erfolgte dabei nach Nutzergruppen, da Administratoren, Dienstanbieter und potenzielle Dienstnutzer

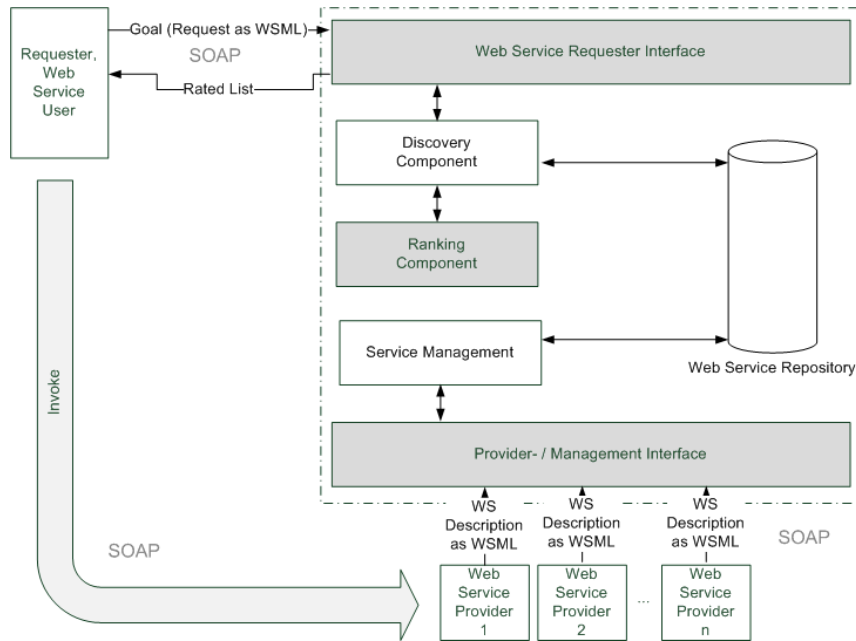


Abbildung 5.1: Architektur ConQo

unterschiedliche Anforderungen an das Dienstverzeichnis haben. Für jede dieser Nutzergruppen ist eine Web Service - Schnittstelle mit spezialisierten Methoden verfügbar. Durch diese Trennung wird es für real eingesetzte Systeme einfacher die Funktionalitäten vor unbefugtem Zugriff zu schützen.

Neben diesen drei grundlegenden Schnittstellen sind im Folgenden auch die Anbindungen an GrandSlam (Monitoring) und den ContractWizard (Vertragsaushandlung) beschrieben. Aus Kompatibilitätsgründen besitzen einige der in den nachfolgenden Schnittstellen enthaltenen Methoden XML-Dokumente als Rückgabewerte. Zum Parsen dieser Dokumente und Rücktransformation in ein Objektnetz steht die Bibliothek "conqo-xml-parser.jar" zur Verfügung.

### 5.2.1 AdminInterface

Die Schnittstelle zur Administration umfasst eine Methode zur Initialisierung von ConQo (u.a. Anlage der Datenbank) sowie vorkehrungen zur Verwaltung der Anbindung an das Monitoring.

### 5.2.2 ProviderInterface

Das ProviderInterface ist auf die Bedürfnisse von Dienst Anbietern abgestimmt. Hier finden sich Methoden zur Registrierung, Manipulation und Löschung von Services.

### 5.2.3 ClientInterface

Für Nutzer des Dienstverzeichnisses sind in dieser Schnittstelle Methoden gesammelt um Services aufzulisten. Zusätzlich kann das Discovery auf Basis eines WSML-Goals verwendet werden.

### 5.2.4 GrandSlam

Der Monitor "GrandSlam" ist ebenfalls über Web Services angebunden. Die entsprechenden Methoden befinden sich im Package "monitor" in den Klassen "MonitorAccess" und "MonitorCall". Im Falle einer Registrierung eines Services bei ConQo mit SLA-Templates wird die Vorlage automatisch beim entsprechenden Monitor durch Aufruf der Methode "registerSLA" registriert. Intervallartig werden nun durch Aufruf der Methode "getWSUpdate()" die neuesten Messdaten gesammelt abgeholt und zur Nutzung bei der Dienstsuche in der Datenbank abgelegt. Im Falle eines Vertragsbruches steht dem Monitor die Methode "breachOfSLA()" zur Verfügung.

### 5.2.5 ContractWizard

Der ContractWizard dient der Aushandlung von Verträgen zur Zusicherung von Parametern. Hierzu werden dem Nutzer in den Discovery-Ergebnissen zusätzlich die verfügbaren SLA-Templates aufgeführt. Diese Templates dienen dem ContractWizard als Grundlage zur Aushandlung konkreter Verträge (Service Level Agreements).

Der Client übermittelt dem ContractWizard hierzu URLs, unter denen die entsprechenden Vertragsvorlagen heruntergeladen werden können.

## 5.3 Client

Für den Test der Schnittstellen und zur Illustration der Funktionalität wurde ein einfacher JSP-Client implementiert. Dieser Client nutzt ausschließlich die SOAP-Schnittstellen AdminInterface, ProviderInterface und ClientInterface und demonstriert damit die Verwendbarkeit von ConQo in beliebigen plattformunabhängigen Komponenten.

Die Funktionalität ist weitestgehend selbsterklärend und nach den Möglichkeiten der einzelnen Schnittstellen sortiert. Die wichtigsten Elemente stellen jedoch sicherlich die Registrierung von Services sowie die Dienstsuche dar. Da es wenig praktikabel ist für jede Dienstsuche ein WSML-Goal manuell zu erstellen, wird für jede Domain ein Formular mit Dienstgüteparametern generiert. Mit dessen Auswertung kann der integrierte Goalgenerator die entsprechende maschinenlesbare WSML-Beschreibung erstellen.

Zusätzlich lassen sich unter Anderem die registrierten Ontologien und Services abrufen und deren Beschreibungen erweitern und manipulieren.

## 5.4 Bibliotheken

### 5.4.1 DIP QoS-enabled Service Discovery component

ConQo basiert zu großen Teilen auf einer Arbeit des DIP<sup>1</sup> - Projektes (Data, Information, and Process Integration with Semantic Web Services). Dieses Projekt hat zum Ziel Quality-

---

<sup>1</sup><http://lsirpeople.epfl.ch/lhvu/download/qosdisc/factsheet/qosdiscovery.html>

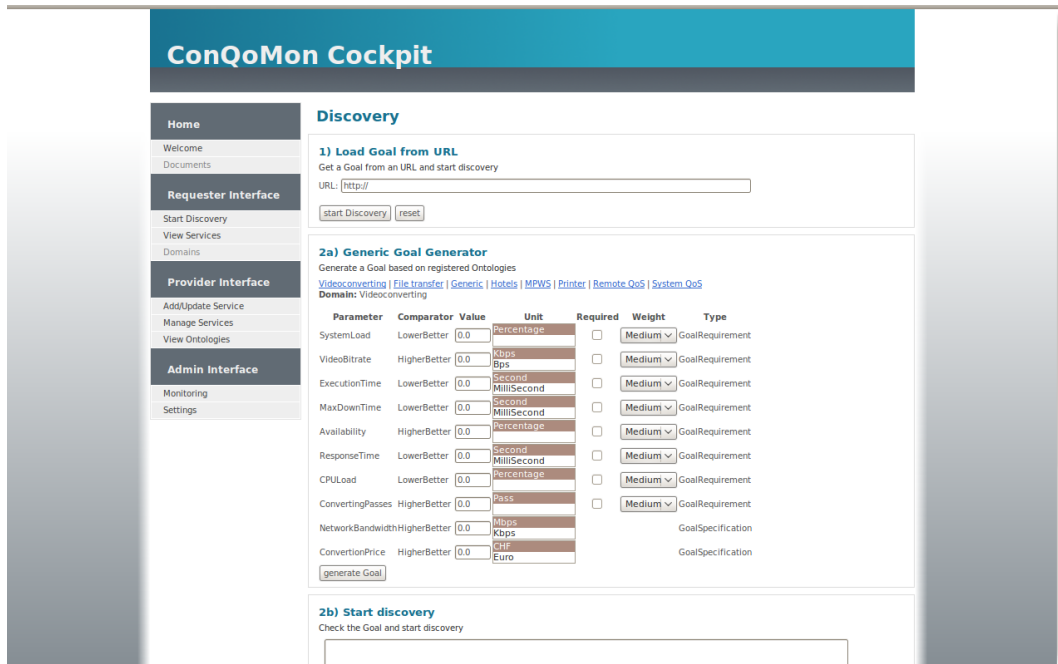


Abbildung 5.2: JSP-Client

of-Service Parameter von Web Services semantisch zu Beschreiben und realisiert mit dieser Komponente einen Prototyp.

#### 5.4.2 WSML2Reasoner

Modulares Framework zur Überprüfung, Normalisierung und Transformation von WSML Ontologien in die Syntax verschiedener Reasoning-Engines.

#### 5.4.3 WSMO4J

Bei WSMO4J handelt es sich um eine API und eine Referenzimplementierung zum Erstellen von Semantic Web Services und Semantic Business Process Applications basierend auf WSMO.

### 5.5 Tools

#### 5.5.1 EclipseEE

Bei Eclipse handelt es sich um eine Open Source Entwicklungsumgebung. Mit der speziell auf JavaEE abgestimmten Version von Eclipse Galileo steht eine komfortable IDE für die Entwicklung von Web Services und Java-Anwendungen zur Verfügung.

### 5.5.2 WSMO Studio

WSMO Studio 0.7.2 ist ein grafischer Editor für WSML, welcher als Eclipse-Plugins zur Verfügung gestellt wird. Enthalten sind u.a. ein Ontologie-Editor mit integriertem Reasoner, ein Editor für WSMO-Elemente (Web Services, Goals, Mediators, ...). WSMO Studio eignet sich hervorragend um die Eigenheiten von WSML kennenzulernen, Ontologien zu erstellen und zu visualisieren und mittels eines Reasoners Anfragen zu stellen. Die Erstellung von Basis-Ontologien, Service-Beschreibungen und Goals setzt dennoch recht genaue Kenntnis vom notwendigen Aufbau dieser Dokumente für ConQo voraus. In ähnlicher Art und Weise wie XML quasi unzählige Möglichkeiten bietet Informationen darzustellen gibt es auch in WSML mehrere Varianten Eigenschaften zu formulieren.

# Kapitel 6

## Installation

Die folgende Anleitung ist auf eine auf Debian basierende Umgebung abgestimmt und auf wurde auf Debian "Lenny" und Ubuntu "Karmic Koala" getestet. Bei analoger Konfiguration der benötigten Komponenten ist das System auch auf Windows lauffähig, wenn auch folgend nicht im Einzelnen beschrieben.

Neben der Wahl des Betriebssystems sind im Folgenden zwei verschiedene Installationsvarianten beschrieben. Einmal besteht die Möglichkeit ConQo direkt aus Eclipse zu starten und auszuführen und zum Anderen unabhängig von einer IDE zu nutzen. Die erste Variante ist für eine Weiterentwicklung gut geeignet, die zweite Variante ist hingegen zum schnellen Test und für Produktivsysteme zu bevorzugen.

### 6.1 Voraussetzungen

#### 6.1.1 Java SE

Java SE bildet die Grundlage zur Entwicklung von Java basierten Applikationen und Web Services. Getestet wurde die Konfiguration mit dem Java JDK6. In diesem Paket ist bereits das JRE, welches zur Ausführung von Java-Applikationen notwendig ist, enthalten. Die jeweils aktuelle Version ist unter <http://java.sun.com> erhältlich.

#### 6.1.2 Apache Tomcat

Als Umgebung für die Ausführung von Java-Code auf Webservern kam Apache Tomcat in der Version 6.0.20 zum Einsatz. Dank des integrierten JSP-Compilers "Jasper" können hiermit neben den web-basierten Schnittstellen auch Java Server Pages, und damit der Test-Client, übersetzt und ausgeführt werden. Die getestete Version 6.0.20 ist unter <http://tomcat.apache.org/> frei zum Download verfügbar. Bei einer Standardinstallation ist der Tomcat-Server auf Port 8080 konfiguriert und kann daher ohne Schwierigkeiten parallel mit einem Apache HTTP-Server betrieben werden. Die Installation ist mit dem Entpacken des Archives abgeschlossen. Der Start und der Stop erfolgen entweder mittels der Scripte "bin/startup.sh" bzw. "bin/shutdown.sh" oder nach dem Einbinden in Eclipse direkt aus der Entwicklungsumgebung (siehe unten). Die erforderliche Umgebungsvariable "CATALINA\_HOME" wird beim Start automatisch gesetzt.

## 6.2 Ausführung aus Eclipse

### 6.2.1 Eclipse

### 6.2.2 Tomcat unter Eclipse konfigurieren

Nach dem Start von Eclipse kann die Serveransicht hinzugefügt werden. Dies geschieht durch Klick auf

“Window“ ⇒ “Show View“ ⇒ “Other...“ ⇒ “Server“ ⇒ “Servers“ ⇒ “OK“.

In der neuen Ansicht ist es nun möglich via Rechtsklick “New“ ⇒ “Server“ den Apache Tomcat in Eclipse einzubinden. Im folgenden Dialog muss hierzu das Verzeichnis ausgewählt werden, in welches der Tomcat zuvor entpackt wurde.

Im Gegensatz zum manuell entpackten Tomcat ist die Einbindung der in den Paketquellen von Ubuntu enthaltenen Version problematisch und erfordert die manuelle Anpassung einiger Pfade.

### 6.2.3 SVN Checkout

Für den SVN Checkout aus Eclipse heraus sind mehrere Plugins (z.B. Subversive) verfügbar. Da sich die Vorgehensweise je nach Plugin geringfügig unterscheidet sei an dieser Stelle auf die zugehörige Dokumentation verwiesen.

### 6.2.4 Ausführung

Nach dem Hinzufügen der Projekte zum zuvor installierten Server kann dieser durch Rechtsklick gestartet werden. Je nach gewählter Datenbank und Port kann die Anpassung der Konfigurationsdateien notwendig sein.

## 6.3 Stand-Alone

Alternativ zur Ausführung unter Eclipse kann ConQo auch separat betrieben werden.

### 6.3.1 SVN Checkout

Für den Stand-Alone-Betrieb eignet sich zum Checkout z.B. ein Kommandozeilen-Client.

### 6.3.2 Apache Ant

Dem Projekt liegen Build-files bei. Nach der Installation von Apache Ant können die Projekte ”Matchmaker“ und ”ConQoCockpit“ durch Aufruf von ”ant“ in den entsprechenden Projektordnern kompiliert werden.

### 6.3.3 Projekte deployen

Zum Deployen der Projekte in Tomcat sind die folgenden Kopieroperationen notwendig:

Listing 6.1: Deploy

```
1 cp -R Matchmaker/WebContent $CATALINA_HOME/webapps/Matchmaker
2 cp -R ConQoCockpit/WebContent $CATALINA_HOME/webapps/ConQoCockpit
3 cp -R Matchmaker/build/classes $CATALINA_HOME/webapps/Matchmaker/WEB-INF/
```

```
4 cp -R ConQoCockpit/build/classes $CATALINA_HOME/webapps/ConQoCockpit/WEB-INF/  
5 cp -R Matchmaker/dbinit $CATALINA_HOME/webapps/Matchmaker/
```

### 6.3.4 Updates

Für automatisierte Updates aus dem SVN-Repository steht das Script "update\_conqo.sh" zur Verfügung. Je nach Installationsort ist ggf. eine Anpassung der Pfade erforderlich.

## 6.4 Konfiguration anpassen

Sowohl ConQo selbst als auch der JSP-Client besitzen Konfigurationsdateien. Im Falle von ConQo sind die entsprechenden Einstellungen unter "src/qosdisc.properties" vorzunehmen. Wichtig ist eine korrekte Konfiguration der Datenbank. Im Client werden die Einstellungen in der Datei "src/ConQoCockpit.properties" vorgenommen.

# Literaturverzeichnis

- [1] *Gergana Stoyanova*, Entwicklung einer dynamischen kontext- und Quality-of-Service sensitiven Dienstausswahl für Web Services, 2008
- [2] *Matthias Neumann*, Konzeption und prototypische Implementierung einer Lösung zur dynamischen Dienstausswahl anhand von Dienstgtparametern (QoS) Diplomarbeit, 2007
- [3] *Matthias Wauer*, Entwicklung und prototypische Implementierung einer kontextsensitiven Dienstausswahl in serviceorientierten Architekturen, Belegarbeit, 2007
- [4] *Le-Hung Vu, Sebastian Gerlach, Fabio Porto, Othman Tajmouati, Manfred Hauswirth*, QoS-enabled Service Discovery Component Prototype 1 Report
- [5] *Krzysztof Brzostowski, Witold Rekuc, Janusz Sobiecki, Leopold Szczurowski*, Service Discovery in the SOA System