Wissensnetzwerke im Grid (WisNetGrid)
– **Evaluierung der Suchfunktion** –
Bericht D3.2.6 zu Arbeitspaket 3.2

Juni 2011

Martin Junghans (KIT) und Sudhir Agarwal (KIT)

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

Die Grundlage der Diensteschicht ist die semantische Dienstbeschreibungsprache, sowie die Komponente zur Dienstsuche. Die Suche ist ein wichtiger Baustein zu jeglicher Verwendung von Diensten und befasst sich mit der Identifikation von Diensten, welche ein gewünschtes Verhalten und Qualität bieten. Die Suchkomponente hat dabei die Aufgabe eine Anfrage mit Anforderungen an einen gewünschten Dienst mit den angebotenen Dienstbeschreibungen aus einem Verzeichnis für Dienstbeschreibungen abzugleichen und diejenigen Dienste zu identifizieren, die die Anforderungen erfüllen. Dieser Abgleich von Dienstbeschreibungen zu einer gegebenen Anfrage gilt auch als die Grundlage zur Dienstekomposition mit dessen Hilfe neue komplexere Dienste aus existierenden Diensten zusammengesetzt werden können.

In den Berichten [AHM+09] und [AHJM10] wurde die semantische Beschreibungssprache für Web-Dienste zur Verwendung im Grid motiviert und eingeführt. Im Bericht [AHM10] wurde die Entwicklung eines Grid-basierten Diensteverzeichnisses für semantische Dienstbeschreibungen beschrieben, sowie ein Ausblick auf die Suchfunktionalität gegeben. In dem vorliegenden Dokument ist der Fokus die Suchfunktionalität des Diensteverzeichnisses. Wir betrachten dazu die Suche basierend auf den Metadaten von den im Grid abgelegten Dienstbeschreibungen, sowie den Abgleich funktionaler und nicht-funktionaler Eigenschaften der Dienste. Die Komposition der Dienste, die auf der Suchfunktionalität aufbaut, wird in den folgenden Berichten des Arbeitspakets 3.3 behandelt.

Die Suche nach Diensten basierend auf der formalen Beschreibung von funktionalen (Beschreibung des Dienstverhaltens) und nicht-funktionalen (qua-

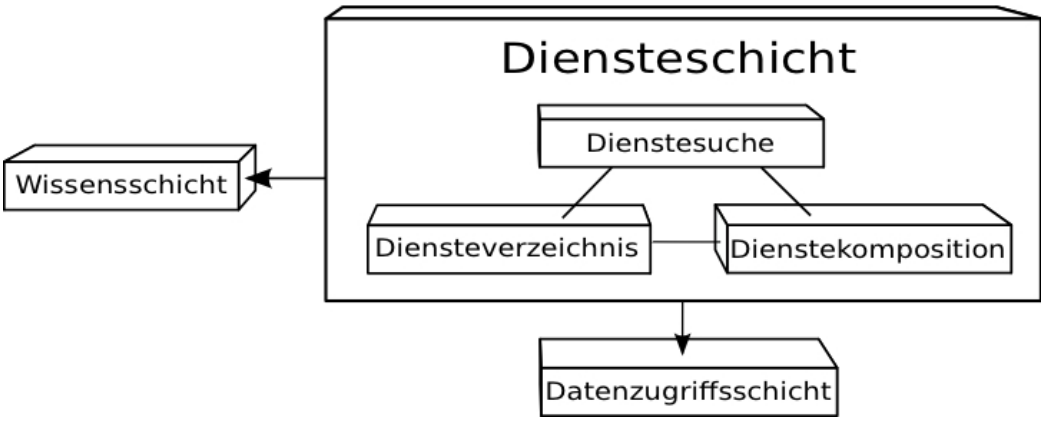

Abbildung 1.1: Diensteschicht

litativen) Eigenschaften der Dienste kann schnell rechenintensiv werden. Somit kann sich die Nützlichkeit des Suchansatzes deutlich verringern wenn ein Benutzer bei einfachen Suchanfragen bereits eine längere Zeit auf die Resultate warten muss. Unsere Studien zeigten, dass sich mit einer größer werdenden Anzahl an verfügbaren Diensten der Ressourcen und Zeitbedarf der Suchkomponente zur Bearbeitung der Anfragen rasch vergrößerte. Dieses beobachtete Problem entsteht durch die hohe Komplexität des Dienstabgleichs, welche sich aus der Komplexität und Expressivität des Beschreibungs- und Anfrageformalismuses ergibt. Die Verwendung semantischer Technologien zum logischen Schließen trägt ebenfalls nicht unerheblich zur Komplexität des Suchproblems bei. Da die Suche eine zentrale Stellung in der Diensteschicht, wie auch in jeder Service-Orientierten Architektur, einnimmt, ist eine effiziente Suche unabdingbar.

Daher stellen wir in diesem Bericht das Konzept der Dienstklassifikation vor. Es erlaubt uns die Effizienz der Suche zu steigern und verspricht gleichzeitig die Nutzbarkeit der Dienstmodellierung und -suche zu erhöhen. Dienstklassen sind benannte (Fragmente von) Anfragen, welche durch deren formale Klassendefinition viele Vorteile besitzen und bisherige verwandten Arbeiten erweitern. In diesem Bericht werden wir diesen Ansatz einführen und dessen Vorteile, z.B. Steigerung der Effizienz von Dienstsuche, aufzeigen.

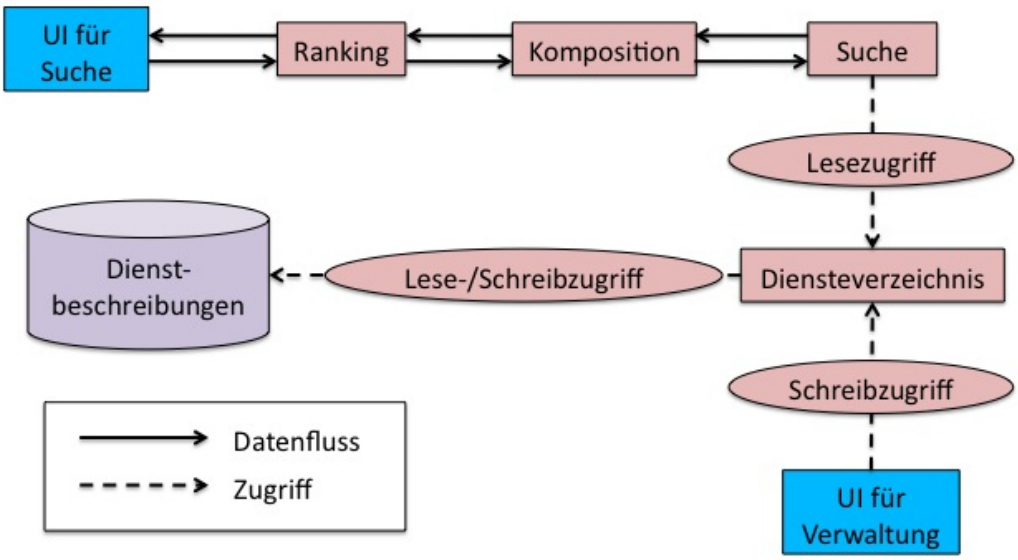

Abbildung 1.2: Verwendung der Dienstbeschreibungen für die Mehrwertdienste

Kapitel 2

Dienstsuche basierend auf Metadaten

Die Suche nach Diensten findet in der Diensteschicht der WisNetGrid Architektur statt und basiert auf den verfügbaren Dienstbeschreibungen im Diensteverzeichnis (vgl. Abbildung 1.1). Das Diensteverzeichnis speichert Dienstbeschreibungen, welche die formale Beschreibung von funktionalen sowie nicht-funktionalen Eigenschaften der Dienste beinhalten. An jede Dienstbeschreibung sind auch Metainformationen (sogenannte Metadaten) angehängt, das heißt, dass diese Metadaten selbst nicht Bestandteil der Dienstbeschreibungen sind.

Eine Ressource des Dienstverzeichnisses bezeichnet eine Entität, die eine Dienstbeschreibung sowie dessen Metadaten enthält (vgl. Abbildung 2.2). Basierend auf den Metadaten einer Dienstbeschreibung, welche im folgenden Abschnitt 2.1 zusammengefasst sind, können Kriterien zur Suche nach Diensten aufgestellt werden. In Abschnitt 2.2 stellen wir diese Technik der Dienstsuche vor.

2.1 Metadaten der Dienstbeschreibungen

Metadaten kennzeichnen eine Ressource im Diensteverzeichnis. Sie beschreiben nicht die (funktionalen und nicht-funktionalen) Eigenschaften der Dienste, welche in der Dienstbeschreibung enthalten sind (vgl. Abbildung 2.2).

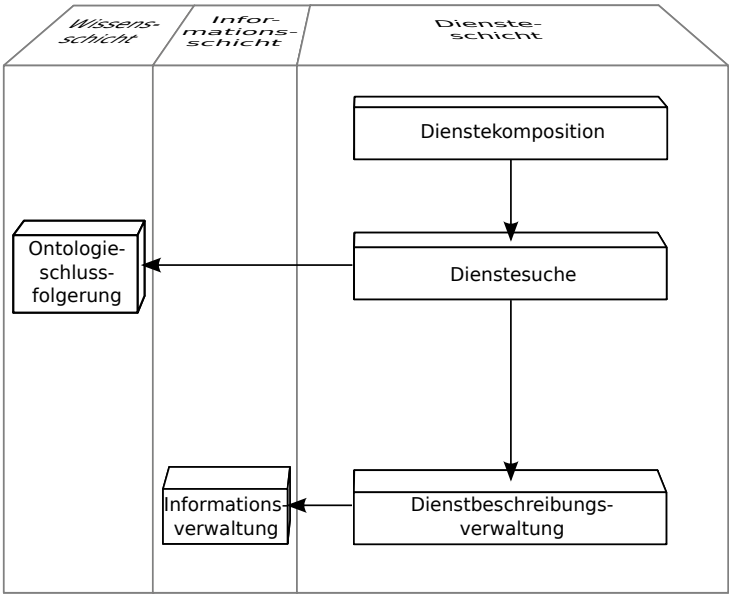

Abbildung 2.1: Komponenten der Diensteschicht und deren Relation zur Wissens- und Informationsschicht.

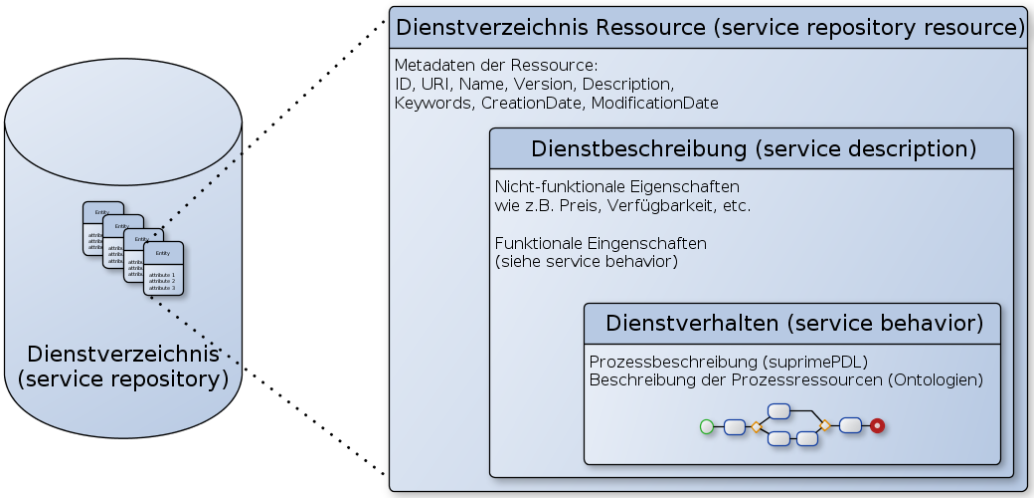

Abbildung 2.2: Zusammenhänge zwischen Ressourcen des Dienstverzeichnisses, Dienstbeschreibungen und prozessartigen Verhaltensbeschreibungen.

Tabelle 2.1: Überblick über die Metadaten von Dienstressourcen. Suchbare Metadaten können in den Suchanfragen verwendet werden.

Bezeichner	**abstrakter Datentyp**	**Erklärung**	**Suchbar**
ID	URI	Eindeutiger Bezeichner zur Identifikation	×
URI	URI	URI der Ressource im Dienstverzeichnis	
Name	Satz	Name des Dienstes	×
Version	Wort	Version der Dienstbeschreibung	×
Description	Satz	Beschreibung des Dienstes	×
Domains	Menge von Wörten	Domänen in denen der Dienst relevant sein kann	×
Keywords	Menge von Wörten	Menge von Schlüsselwörtern zur Einordnung des Dienstes	×
Owners	Menge von Nutzer IDs	Besitzer der Dienstbeschreibung	×
Creator	Nutzer ID	Ersteller der Dienstbeschreibung	×
Contributors	Menge von Nutzer IDs	Benutzer die zur Dienstbeschreibung beigetragen haben	×
CreationDate	Datum	Datum der Erstellung der Dienstbeschreibung	×
LastModification-Date	Datum	Datum der letzten Änderung der Dienstbeschreibung	×
ChangeLog	Liste von Sätzen	Liste der Änderungskommentare	×
Language	Sprachcode (RFC1766)	Sprache benutzt in Freitexten der Beschreibung	
RelatedResources	Menge von URIs	URIs in Beziehung stehender Dienstbeschreibungen	
Certificates	Menge von Zertifikaten	Zertifikate zur Bestätigung von Diensteigenschaften	

Aufbauend auf den Bericht D3.2.2 "Semantische Beschreibungssprache für Web-Dienste" [AHJM10] gibt Tabelle 2.1 einen Überblick über alle Metadaten die zur Beschreibung einer Dienstressource zur Verfügung stehen. Die als suchbar gekennzeichneten Metadaten werden für die Dienstsuche herangezogen. D.h., Anwender dieser Dienstsuche können in einer Suchanfrage zur Beschreibung eines gewünschten Dienstes die Werte dieser Metadaten eingrenzen.

Die Metadaten einer Ressource werden durch eine Menge von Schlüssel-Wert-Paaren repräsentiert. Ein Schlüssel k kommt aus der Menge der verfügbaren Bezeichner, wie in Tabelle 2.1 aufgelistet. Der einem Schlüssel k zugeordnete Wert v ist aus dem Wertebereich des angegebenen Datentyps.

Tabelle 2.2: Modellierung der Metadaten in Resource Description Framework.

Bezeichner	**Property**	**Property Range**
ID	dcterms:identifier	rdfs:Literal
URI	dcterms:source	k.A. (xsd:anyURI)
Name	dcterms:title	rdfs:Literal
Version	dcterms:hasVersion	k.A. (rdfs:Literal)
Description	dcterms:description	k.A. (rdfs:Literal)
Domains	foaf:topic	owl:Thing (rdfs:Literal)
Keywords	dcterms:subject	k.A. (rdfs:Literal)
Owners	dcterms:publisher	k.A. (dcterms:Agent)
Creator	dcterms:creator	dcterms:Agent
Contributors	dcterms:contributor	dcterms:Agent
CreationDate	dcterms:created	rdfs:Literal
LastModificationDate	dcterms:modified	rdfs:Literal
ChangeLog	dcterms:provenance	dcterms:ProvenanceStatement
Language	dcterms:language	dcterms:LinguisticSystem
RelatedResources	dcterms:relation	k.A. (rdfs:Literal)
Certificates	suprime:hasCertificate	suprime:Certificate

Jedem Schlüssel k kann maximal ein Wert v aus dem Wertebereich zugewiesen werden. Mehrfachzuweisungen sind nicht möglich.

Speicherung der Metadaten im Grid Metadaten werden, wie die Dienstbeschreibungen, durch das Dienstverzeichnis im Grid persistiert. Während die serialisierten Dienstbeschreibungen mittels WebDAV-Protokoll und iRODS Adapter im Grid gespeichert werden [AHM10], werden die Metadaten über den WisNetGrid Metadatendienst [DHH+10] im Grid gespeichert. Diese Trennung von Metadaten und Dienstbeschreibung existiert ausschließlich in der Datenverwaltungsschicht, da im Diensteverzeichnis Dienstressourcen ganzheitlich behandelt werden. D.h., im Verzeichnis werden die Ressourcen nur zur Speicherung im Grid in Metadaten und Dienstbeschreibung getrennt und beim Laden unmittelbar zu Dienstressourcen aggregiert.

Die Wiederverwendung des Metadatendienstes zur Speicherung der Metadaten der Dienstbeschreibungen hat den Vorteil, dass die bereits verfügbare Anfragefunktionalität des Metadatendienstes für die Dienstsuche verwendet werden kann. Außerdem ist diese Trennung für die Leistung der Dienstsuche vorteilhaft. Bei der Suche basierend auf Metadaten muss die Dienstbeschreibung nicht aus dem Grid geladen und die Metadaten müssen nicht von der Dienstbeschreibung getrennt werden. Letzteres kann aufgrund der Komplexi-

tät der semantischen Dienstbeschreibungen einen nicht zu vernachlässigenden Aufwand für einen Parser darstellen.

Der WisNetGrid Metadatendienst verwaltet Dateien zur Metadatenbeschreibung im Grid und akzeptiert unter anderem auch eine RDF-Repräsentation der Metadaten. RDF ist das Resource Description Framework [MM04], welches zur formalen Beschreibung von Informationen über Ressourcen verwendet wird. Wir beschreiben jeden Schlüssel durch ein sog. Property, welches eine Eigenschaft einer Ressource dargestellt. In Tabelle 2.2 sind jedem Metadaten-Schluessel k auf ein Property abgebildet. Die Property Range bezeichnet die jeweils durch das Property definierten Wertebereich eines Wertes v. Falls keine Beschränkung des Wertebereichs gegeben war, dann wird die von uns eingeführte in Klammern stehende Beschränkung bei der Metadatenbeschreibung angewendet.

Die Properties wurden weitestgehend dem bestehenden Dublin Core Vokabular [WKLW98] zur Beschreibung von Metadaten entliehen. Die Wiederverwendung des bestehenden RDF Vokabulars ist u.a. aus Gründen der Interoperabilität empfehlenswert. In Tabelle 2.2 werden die Abkürzungen der Ressourcen angegeben. So steht dcterms: abkürzend für den Namensraum `http://purl.org/dc/terms/` der von Dublin Core definierten Terme. Falls das Dublin Core Vokabular nicht ausreichend war, wurden Terme aus anderen Projekten verwendet: Friend of a Friend [BM10] (abgekürzt durch foaf:), RDF Schema [BG04] (RDF Vocabulary Description Language, rdfs:), XML Schema (xsd:). Lediglich die Terme suprime:hasCertificate und suprime:Certificate zur Beschreibung der verwendeten Zertifikate wurden neu definiert. Die Metadaten eines Beispieldienstes werden in Auflistung 2.1 gezeigt.

```
1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4  @prefix sup: <http://suprime.aifb.kit.edu/> .
5  @prefix supSvc: <http://suprime.aifb.kit.edu/service/> .
6  @prefix supProc: <http://suprime.aifb.kit.edu/process/> .
7  @prefix supOnto: <http://suprime.aifb.kit.edu/ontology/> .
8  @prefix dc: <http://purl.org/dc/elements/1.1/> .
9  @prefix dcterms: <http://purl.org/dc/terms/> .
10 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
11
12 <http://suprime.aifb.kit.edu/service/4682691685906> a supSvc:Service ;
13     dcterms:identifier "http://suprime.aifb.kit.edu/service/4682691685906"^^rdfs:Literal ;
14     dcterms:source "http://194.94.143.4/webdav/irods-test/repository/services/http___suprime.aifb.
           kit.edu_service_4682691685906"^^xsd:anyURI ;
15     dcterms:title "String_Tokenizer_Service"^^rdfs:Literal ;
16     dcterms:hasVersion "0.98alpha"^^rdfs:Literal ;
17     dcterms:description "Allows_an_application_to_break_a_string_into_tokens."^^rdfs:Literal ;
```

```
18      foaf:topic "Text_Analysis"^^rdfs:Literal , "Text_Processing"^^rdfs:Literal ;
19      dcterms:subject "string"^^rdfs:Literal , "tokenizer"^^rdfs:Literal ;
20      dcterms:publisher "sag"^^rdfs:Literal , "mju"^^rdfs:Literal ;
21      dcterms:creator "mju"^^rdfs:Literal ;
22      dcterms:contributor "sag"^^rdfs:Literal ;
23      dcterms:created "2011-07-07T06:41:12"^^xsd:dateTime ;
24      dcterms:modified "2011-07-07T06:41:12"^^xsd:dateTime .
25      dcterms:provenance _:node1642niv9fx1 ;
26      dcterms:language "en"^^xsd:language ;
27      dcterms:relation "http://suprime.aifb.kit.edu/ontologies#467849395309403"^^xsd:anyURI ;
28      supSvc:hasCertificate "HighReliability"^^rdfs:Literal , "FreeLibreOpen"^^rdfs:Literal .
29
30  _:node1642niv9fx1 a rdf:Seq ;
31      rdf:_0 "Inital_Import"^^dcterms:ProvenanceStatement ;
32      rdf:_1 "All_bugs_patched"^^dcterms:ProvenanceStatement .
```

Auflistung 2.1: RDF Serialisierung (Turtle Syntax) der Metadaten eines Beispieldienstes.

2.2 Dienstsuche basierend auf Metadaten

Die in einem RDF-Syntax serialisierten Metadaten der Dienstressourcen sind durch die Anfragesprache SPARQL maschinell durchsuchbar und auswertbar. "SPARQL Protocol And RDF Query Language" (SPARQL) [PS08] ist eine Anfragesprache für RDF, die vom WisNetGrid Metadatendienst unterstützt wird. Der Metadatendienst stellt die Funktionalität der Suche über Metadaten bereit und akzeptiert dafür SPARQL Anfragen an einer öffentlichen Schnittstelle. Ausgehend von den verschiedenen Datentypen der einzelnen Werte der Metadaten können verschiedene Anfragetypen bzw -operationen angewendet werden.

In einer Anfrage kann zu jeder Metainformation entweder ein gewünschter Wert vom jeweiligen Datentyp, oder ein Bereich gewünschter Werte angegeben werden (vgl. Spalte 'Datentyp in Suchanfrage' in Tabelle 2.3). Die Auswertung der Anfrage bestimmt ob ein Angebot (die Beschreibung der Metadaten eines Dienstes) die Anforderungen einer Suchanfrage erfüllt. Für verschiedene Datentypen werden verschiedene Methoden des Abgleichs implementiert (vgl. Spalte 'Auswertung der Suchanfrage').

Bei der Auswertung der Anfrage wird ermittelt ob für jede Metainformation der Wert eines Angebots in den gewünschten Werten einer Anfrage enthalten ist. Wie in Tabelle 2.3 dokumentiert, soll zum Beispiel die Zeichenkette des angefragten Namens im Namen des angebotenen Dienstes vorkommen. Dagegen wird bei den Schlüsselwörtern geprüft, ob jedes in der

Tabelle 2.3: Überblick über mögliche Suchanfragen basierend auf Metadaten.

Bezeichner	**Datentyp in Suchanfrage**	**Auswertung der Suchanfrage**
ID	ID	Gleichheit der Zeichenketten
Name	Satz	Anfrage ist ein Teil der Zeichenkette des Angebots
Version	Wort	Gleichheit der Zeichenketten
Description	Satz	Anfrage ist ein Teil der Zeichenkette des Angebots
Domains	Menge von Wörtern	Angebot enthält (mindestens) alle Wörter der Anfrage
Keywords	Menge von Wörtern	Angebot enthält (mindestens) alle Wörter der Anfrage
Owners	Menge von Nutzern	Angebot enthält (mindestens) alle Nutzer der Anfrage
Creator	Nutzer	Gleichheit der Nutzer
Contributors	Menge von Nutzern	Angebot enthält (mindestens) alle Nutzer der Anfrage
CreationDate	Start- und Enddatum	Angebot liegt im Intervall der Nachfrage
LastModification-Date	Start- und Enddatum	Angebot liegt im Intervall der Nachfrage
ChangeLog	Menge von Wörtern	Angebot enthält (mindestens) alle Wörter der Anfrage

Anfrage spezifizierte Schlüsselwort vom Angebot erbracht wird. Dieses Verhältnis zwischen Angebot und Anfrage wird häufig plug-in match [PKPS02] genannt und bedeutet dass das Angebot mindestens die geforderten Eigenschaften besitzt und daher auch noch weitere Eigenschaften besitzen kann. Bei der ID und dem Ersteller des Dienstes ist ein solches Ähnlichkeitsmaß nicht anwendbar und es wird daher nur die Gleichheit überprüft. Bei der Suche nach Diensten basierend auf dem Datum der Erstellung und der letzten Änderung ist es möglich einen Zeitraum anzugeben, in der das gewünschte Datum liegen soll.

Formulierung der Anfragen. Wie bereits beschrieben, werden die Anfragen in SPARQL formuliert. Nach der Auswertung der Anfrage durch den im Metadatendienst verwendeten Triplestore mit SPARQL Endpoint werden die IDs der Dienste zurückgegeben, die die Anforderungen bezüglich der Metadaten in der Anfrage erfüllen. Zu den Details der Anfragesprache sowie deren Syntax verweisen wir an die W3C Spezifikation [PS08]. Wir geben im Folgenden einige Beispiele, wie Suchanfragen in SPARQL ausgedrückt werden. In einem vom Endanwender zu nutzenden System wird natürlich nicht

erwartet, dass SPARQL Anfragen eingegeben werden. Formulare zur Eingabe der Suchkriterien können leicht vom festen Schema der Metadaten und deren Wertebereichen abgeleitet werden. Solch eine Suchmaske kann dann von der darunter liegenden Anfragesprache SPARQL abstrahieren, so dass ein Endanwender diese Sprache nicht kennen muss. Somit können fehlerhafte Eingaben durch die Nutzer leichter vermieden werden.

Anfrage 1 (ID) Um einen Dienst basierend auf dessen ID im Dienstverzeichnis auffinden zu können, soll die ID in der Suche angegeben werden. In diesem Beispiel wird nach Diensten mit der ID

`http://suprime.aifb.kit.edu/service/2608862837723`

gesucht. In Auflistung 2.2 ist die entsprechende SPARQL Anfrage abgebildet. Diese wird an den Metadatendienst gesendet. Basierend auf den Metadatenbeschreibungen werden die Entitäten (?svc) vom Typ supSvc:Service identifiziert und als Ergebnis zurückgegeben, die genau diese gewünschte ID besitzen. Diese Anfrage wird maximal einen Dienst zurückgeben können, da die ID der Dienste eindeutig ist.

```
1 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2 PREFIX dcterms: <http://purl.org/dc/terms/>
3 PREFIX supSvc: <http://suprime.aifb.kit.edu/service/>
4
5 SELECT ?svc
6 WHERE { ?svc a supSvc:Service ;
7               dcterms:identifier ?id .
8         FILTER (?id = "http://suprime.aifb.kit.edu/service/2608862837723"
              ^^rdfs:Literal) . }
```

Auflistung 2.2: SPARQL Anfrage zur Suche nach Diensten mit gewünschter ID.

Anfrage 2 (LastModificationDate) Das Beispiel in Auflistung 2.3 zeigt eine SPARQL Anfrage die nach (allen) Diensten sucht, welche nach dem 15. August 2007 letztmalig geändert wurden.

```
1 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
3 PREFIX dcterms: <http://purl.org/dc/terms/>
4 PREFIX supSvc: <http://suprime.aifb.kit.edu/service/>
5
```

```
SELECT ?svc
WHERE { ?svc a supSvc:Service ;
           dcterms:modified ?lastModif .
        FILTER (?lastModif > "2007-08-15T00:00:00Z"^^xsd:dateTime ) . }
```

Auflistung 2.3: SPARQL Anfrage zur Suche nach Diensten mit Beschränkung der letzten Änderung.

Anfrage 3 (Keywords, Owners) Die Anfrage in Auflistung 2.4 sucht nach Diensten von dem Besitzer mit der Nutzeridentifikation ”mju” und die mit einem Schlüsselwort annotiert wurden, welches ”token” enthält.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX supSvc: <http://suprime.aifb.kit.edu/service/>

SELECT ?svc
WHERE { ?svc a supSvc:Service ;
           dcterms:publisher ?owner ;
           dcterms:subject ?keyword .
         FILTER (?owner = "mju"^^rdfs:Literal) .
         FILTER (REGEX(STR(?keyword), "token")) . }
```

Auflistung 2.4: SPARQL Anfrage zur Suche nach Diensten mit bestimmten Schlüsselwort von einem bestimmten Nutzer.

Kapitel 3

Dienstsuche basierend auf Funktionalen und Nicht-funktionalen Eigenschaften

Das Projekt WisNetGrid gliedert sich in 3 Hauptbereiche: Die Verwaltung und Verbreitung von Information, Wissen und Diensten (vgl. Bericht D3.1.1. "Architektur des Gesamtsystems" [ADH+09]). Die Diensteschicht ist hierbei wiederum in ihre 3 Hauptaspekte unterteilt (vgl. Abbildung 1.1):

- Diensteverzeichnis,
- Dienstesuche und
- Dienstekomposition.

Abbildung 2.1 stellt diese Aufteilung und die Kommunikation mit anderen Schichten von WisNetGrid schematisch dar. Das Verzeichnis zur Speicherung und Verwaltung der Dienstbeschreibungen wurde im Bericht D3.2.3 "Diensteverzeichnis" [AHM10] vorgestellt. Die Dienstsuche dient der Bereitstellung von verfügbaren Diensten (d.h. im Diensteverzeichnis abgelegt), welche bestimmten Anforderungen einer Anfrage genügen. Die Dienstsuche basierend auf den Metadaten der Ressourcen im Dienstverzeichnis ermöglicht eine effiziente und intuitive Abfrage. Jedoch deckt diese Art der Dienstsuche die Anforderungen an funktionalen sowie nicht-funktionalen Diensteigenschaften nicht ab.

Funktionale Eigenschaften der im WisNetGrid betrachteten Dienste werden mit Hilfe einer Prozessalgebra sowie Beschreibungslogiken modeliert (vgl. Bericht D3.2.2. "Semantische Beschreibungssprache für Web-Dienste" [AHJM10]). Eine derartige Beschreibung des Verhaltens eines Dienstes ermöglicht die automatisierte Suche nach Funktionalitäten, welche dann entweder direkt oder in einer Komposition (vgl. Abb. 1.1) mit weiteren Diensten verwendet werden kann. Die nicht-funktionalen Eigenschaften der Dienste beschreiben die Güte eines Dienstes und werden zusätzlich zu den funktionalen Eigenschaften bei der Suche zur Eingrenzung der Ergebnismenge eingesetzt.

In diesem Kapitel wird die Dienstsuche im Detail beschrieben. In Abschnitt 3.1 stellen wir die Grundlagen der formalen Beschreibungen, der Anfragesprache zur Formulierung von Anforderungen, sowie die Methode zum Dienstabgleich vorgestellt. Um die Nutzbarkeit des Ansatzes zur Dienstsuche zu erhöhen, wird in Abschnitt 3.2 eine Erweiterung des Ansatzes basierend auf einer Dienstklassifikation vorgestellt. Zunächst wird der Formalismus zur Dienstklassenmodellierung vorgestellt. Danach werden die aus der Verwendung von Dienstklassen gewonnenen Vorteile bezüglich der Dienstmodellierung, Anfragemodellierung und der WisNetGrid Dienstsuche aufgezeigt. Dieses Kapitel schließt mit der Evaluierung der Effizienz der Dienstsuche unter Verwendung der Dienstklassen.

3.1 Grundlagen

3.1.1 Dienstmodellierung

Wie bereits in Abbildung 2.2 dargestellt, umfasst die Beschreibung D_w eines Dienstes w funktionale als auch nicht-funktionale Eigenschaften des Dienstes. Wir betrachten eine endliche Menge von unabhängigen Eigenschaften $\mathcal{P}$. Darunter befindet sich maximal eine Eigenschaft $P \in \mathcal{P}$, welche durch ein Prozessmodell die Funktionalität und das Verhalten des Dienstes w beschreibt. Die restlichen Eigenschaften $\mathcal{P} - \{P\}$ beschreiben nicht-funktionale Eigenschaften des Dienstes w. Jeder Eigenschaft $P \in \mathcal{P}$ ist ein konkreter Wert $v_P \in V_P$ aus dem Wertebereich V_P zugewiesen.

Dienstbeschreibungen werden durch die Verwendung einer Beschreibungslogik semantisch modelliert. Dies ermöglicht das logische Schließen basierend auf den Beschreibungen der Diensteigenschaften. Das Vokabular zur Beschrei-

bung in Angebot und Nachfrage muss daher nicht a priori festgelegt werden, wodurch die in dienstorientierten Anwendungen notwendige Interoperabilität erreicht wird.

In einer Ontologie O_D zur Dienstbeschreibung D fügen wir ein Individuum w für einen Dienst w, sowie eine Relation (Ontologie Property) p für jede Eigenschaft $P \in \mathcal{P}$, die den Dienst w charakterisiert, hinzu. Das Ontologie-Property p setzt das Individuum w in Beziehung mit dem Wert v_P der Eigenschaft. Konzepte zur Modellierung der Datentypen der Werte v_P können durch die verwendete Ontologiesprache direkt angeboten werden. Alternativ werden diese Werte durch Literale oder Individuen modelliert.

Das Dienstmodell (vgl. Auflistung 3.1.1) zur semantischen Beschreibung dieser Informationen enthält Konzepte (z.B. sm:Service, sm:BehavioralProperty) und Relationen (z.B. sm:hasBehavior mit Domain sm:Service und Range sm:BehavioralProperty) die zur Beschreibung der Dienste in einer Ontologie verwendet werden. D.h., das Individuum w wird als Element der Klasse sm:Service modelliert. Eine Eigenschaft wie der Preis des Dienstes w wird durch die Relation ex:hasPrice(w,"1EUR") ausgedrückt, wobei ex:hasPrice als Teilrelation von ex:hasNonFuntionalProperty definiert ist. Die Funktionalität hingegen ist durch einen komplexeren Wert beschrieben. Im Folgenden geben wir einen kurzen Überblick über die Modellierung der Funktionalität durch Prozessausdrücke.

```
1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3  @prefix sm: <http://suprime.aifb.kit.edu/service/> .
4
5  # classes
6  sm:Service a rdfs:Class .
7
8  sm:Property a rdfs:Class .
9
10 sm:NonFuntionalProperty rdfs:subClassOf sm:Property .
11
12 sm:FunctionalProperty rdfs:subClassOf sm:Property .
13
14 sm:BehavioralProperty rdfs:subClassOf sm:FunctionalProperty .
15
16 # properties
17 sm:hasProperty rdfs:domain sm:Service ;
18                rdfs:range sm:Property .
19
20 sm:hasNonFuntionalProperty rdfs:domain sm:Service ;
21                            rdfs:range sm:NonFuntionalProperty ;
22                            rdfs:subPropertyOf sm:hasProperty .
23
24 sm:hasFunctionalProperty rdfs:domain sm:Service ;
25                          rdfs:range sm:FunctionalProperty ;
26                          rdfs:subPropertyOf sm:hasProperty .
```

```
27
28 sm:hasBehavioralProperty rdfs:domain sm:Service ;
29                          rdfs:range sm:BehavioralProperty ;
30                          rdfs:subPropertyOf sm:hasFunctionalProperty .
```

Auflistung 3.1: Das Dienstmodell in der RDF/S Serialisierung (Turtle Syntax).

Semantische Modellierung des Verhaltens von Diensten Wir beschreiben das Verhalten eines Dienstes durch (zustandsbasierte) Transitionssysteme $L = (\mathcal{S}, T, \rightarrow)$. $\mathcal{S}$ sei eine endliche Menge von Zuständen, T eine Menge von Markierungen und $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$ die Menge der markierten Zustandsübergänge (Transitionen).

Ein Zustand $S \in \mathcal{S}$ in einem Transitionssystem L ist gekennzeichnet durch die darin geltenden Fakten (Aussagen über die Welt in diesem Zustand). Eine Veränderung der Gültigkeit dieser Fakten führt zur Änderung des Zustands. Im modellierten Prozess wird dies durch eine Eingabe- oder Ausgabeaktivität oder durch den Aufruf einer lokalen Operation hervorgerufen. Eine entsprechende Transition $r \in \rightarrow$ ist mit der Beschreibung dieser zustandsändernden Aktivität $a \in T$ im Transitionssystem L markiert.

Zur Beschreibung eines Transitionssystems wird der π-Kalkül in Verbindung mit der Beschreibungslogik $\mathcal{SHROIQ}(\mathbf{D})$ verwendet. Der π-Kalkül ist eine Prozess-Algebra zur Modellierung des Verhaltens der Dienste. Mittels Beschreibungslogik werden statische und dynamische Ressourcen der Prozessbeschreibung semantisch beschrieben. Analog zur Modellierung der nicht-funktionalen Eigenschaften wird eine Ontologiesprache, wie z.B. OWL 2 DL [HKP+09], verwendet.

Die Syntax der Prozessmodellierungssprache ist in Tabelle 3.1 zusammengefasst. Für detaillierte Informationen zur Modellierung des Dienstverhaltens, sowie ein auf dem InterLogGrid Szenario basiertes Beispiel, sei abermals auf Bericht D3.2.2. [AHJM10] verwiesen.

3.1.2 Anfragemodellierung

Die Verfügbarkeit vieler Dienstbeschreibungen im Dienstverzeichnis ermöglicht deren Verwendung zur automatischen Dienstsuche, Dienstkomposition oder auch deren Ausführung. Bei der Suche nach Diensten ist eine Spra-

Tabelle 3.1: π-Kalkül Syntax and Semantik

Name	**Syntax**	**Semantik**
Null	$\mathbf{0}$	Aktivität ohne Effekt, zur Terminierung benutzt
Eingabe	$\mathsf{c}[\mathbf{x}].P$	Akzeptiert Eingaben vom Kanal c, bindet diese an die Variablen $\mathbf{x}$ und verhält sich danach wie P
Ausgabe	$\mathsf{c}\langle\mathbf{y}\rangle.P$	Gibt die Werte $\mathbf{y}$ über Kanal c aus und verhält sich danach wie P
Lokale Operation	$\Delta.P$	Nimmt die Änderungen Δ vor und verhält sich danach wie P
Bedingung	$\omega?P$	Verhält sich wie P wenn die Bedingung ω gilt, ansonsten wie $\mathbf{0}$
Komposition	$\prod_{1\leq i\leq n} P_i$	Parallele Komposition von n Komponenten P_i
Auswahl	$\sum_{1\leq i\leq n} P_i$	Verhält sich wie genau einer der n alternativen Prozesse P_i
Aufruf	$@A\{\mathbf{x}\}$	Aufruf eines Agenten A mit den Argumenten $\mathbf{x}$. Ein Agent A mit Argumenten $\mathbf{x}$ ist durch einen Prozessausdruck mit $\mathbf{x}$ als einzige freie Variablen definiert.

che zur Formulierung der Anforderungen an gewünschte Dienste notwendig. Im Kontrast zu vielen existierenden Ansätzen zum Abgleich semantischer (Web) Dienste, welche den Formalismus zur Dienstbeschreibung auch zur Formulierung von Anfragen verwenden, benutzt dieser Ansatz eine dedizierte Anfragesprache zur Spezifikation der Anforderungen. Solch eine Trennung zwischen Beschreibungs- und Anfragesprache wurde im Kontext atomarer Dienste in [JAS10] motiviert und angewendet. In [JA10] wurden zusätzlich nicht-funktionale Diensteigenschaften berücksichtigt.

Eine Dienstanfrage beschränkt die Menge der gewünschten Werte der jeweiligen Diensteigenschaften. Dadurch wird ein Raum gewünschter Dienste aufgespannt; eine spezifische Permutation gewünschter Werte jeweiliger Diensteigenschaften ist ein Suchtreffer, wenn es äquivalent zu der Beschreibung eines angebotenen Dienstes ist. In Abbildung 3.1 ist in der obersten Schicht beispielsweise die Verfügbarkeit eines Dienstes mit $\mathsf{availability} > 0.95$ in einer Anforderung begrenzt worden.

Syntax einer Anfrage $\mathcal{R}$ Eine Anfrage $\mathcal{R}$ kombiniert nicht-funktionale Anforderungen (NFR) $\mathcal{N}$ mit Anforderungen Ψ bzgl. des Dienstverhaltens.

$$\mathcal{R} := \mathcal{R} \wedge \mathcal{R} \mid \mathcal{R} \vee \mathcal{R} \mid \neg\mathcal{R} \mid \mathcal{N} \mid \Psi$$

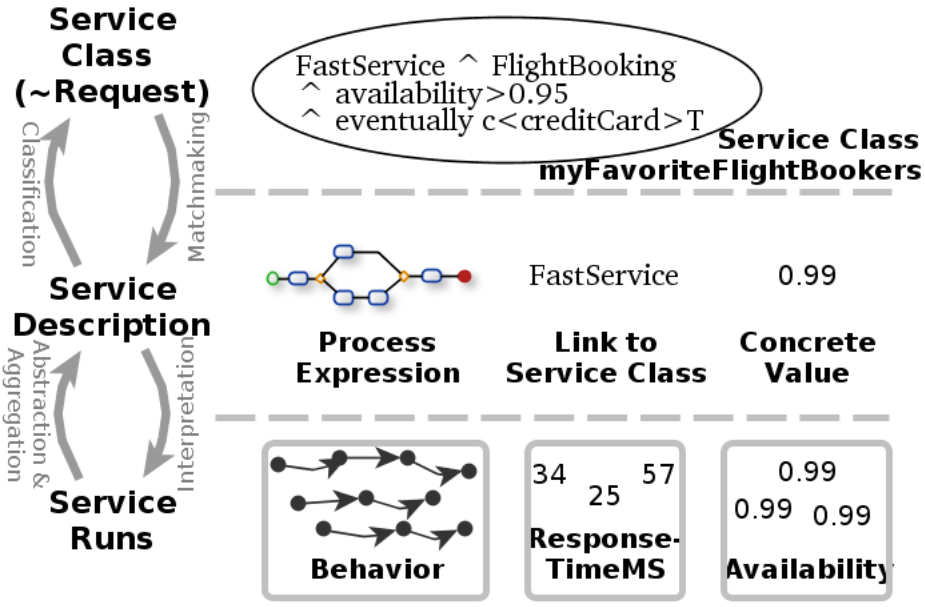

Abbildung 3.1: Zusammenhang von Dienstbeschreibung und Anfrage

Im Gegensatz zu den Schlüssel-Wert-Paaren zur Beschreibung der Dienst-eigenschaften werden die Anforderung durch eine Menge von Schlüssel-Wertemenge-Paaren beschrieben. NFRs sind im Fall einer Teilordnung des Wertebereichs V durch ein Intervall gewünschter Werte oder alternativ durch eine Aufzählung gewünschter Werte beschrieben. Die gewünschten Werte stammen dabei aus dem Wertebereich V_P einer jeweiligen Eigenschaft $P \in \mathcal{P}$. Analog zur Dienstbeschreibung werden NFRs durch Ontologien semantisch modelliert und vordefinierte Datentypen und Operatoren können zur Definition der Intervalle verwendet werden. Alternativ werden die gewünschten Wertemengen mit Hilfe von Individuen, Konzepten und Nominalen definiert.

Beschreibung der Anforderungen an das Dienstverhalten

Die formale und prozessbasierte Beschreibungen von komplexem Dienstverhalten ermöglichen das logische Schließen über deren Choreographie, insbesondere über den Datenfluss (Nachrichtenaustausch) und den Kontrollfluss (z.B. wann werden welche Nachrichten ausgetauscht). Da lokale Operationen es erlauben in den Dienstbeschreibungen von den konkreten Implementierungsdetails der Operationen zu abstrahieren, kann man bei der Anfrage und deren Auswertung auch nur in begrenztem Umfang über die Struktur

der Implementierung Schlüsse ziehen. Nichtsdestotrotz können die Auswirkungen der lokalen Operationen vollständig beschrieben werden, wodurch von der angebotenen Funktionalität nicht abstrahiert werden muss.

Eine Suchanfrage beschreibt Anforderungen über das Auftreten und die Abfolge gewünschter Aktivitäten und deren Effekten sowie bezüglich des durch einen Prozess initiierten Nachrichtenaustausches. Ein gewünschter Nachrichtenaustausch zwischen den verschiedenen Akteuren oder Aktivitäten wird durch die Modellierung der Eingabe- bzw. Ausgabeparameter in einer Anfrage-Ontologie $\mathcal{O}_{\mathcal{R}}$ beschrieben. Darin werden Typen und Zusammenhänge zwischen den Parametern formal beschrieben.

Das gewünschte Verhalten eines Dienstes wird durch μ-Kalkül-Ausdrücke beschrieben. Das μ-Kalkül ist eine Erweiterung der modalen Logik und wird zur Abfrage modaler und temporaler Eigenschaften von Prozessausdrücken verwendet. Es zeichnet sich durch eine kompakte Syntax, einer formalen Semantik, sowie einer großen Expressivität durch die Fixpunkt-Operatoren aus [Sti01, BS01]. Die Syntax ist wie folgt definiert.

μ-Kalkül Syntax

$$\Psi := \Psi \wedge \Psi \mid \neg\Psi \mid \mu X.\Psi(X) \mid \langle a \rangle \Psi \mid P \mid \top \mid \bot$$

Die logische Konjunktion ($\wedge$) und die Negation ($\neg$) ermöglichen die Konstruktion komplexerer Ausdrücke über das Vorhandensein und den Ausschluss gewünschter Eigenschaften Ψ. Die Terminalsymbole der oben gezeigten Grammatik der Anfragesprache sind die Proposition P, die Existenz ($\langle a \rangle \Psi$) einer Aktion a gefolgt von der Bedingung Ψ, die im Folgezustand nach der Aktion a gelten soll, sowie $\top$ und $\bot$, die in allen bzw. keinen Zuständen gelten. Der minimale Fixpunkt-Operator ($\mu X.\Psi(X)$) erlaubt die rekursive Spezifikation von Formeln. Disjunktion ($\vee$), Allquantor ($[a]\Psi$) auf das Vorkommen von Aktionen a, sowie der maximale Fixpunkt-Operator ($\nu X.\Psi(X)$) können mit obigen Konstrukten ausgedrückt werden.

Definition Erweiterter μ-Kalkül Syntax

$$\begin{aligned} \Psi \quad := \quad & \Psi \wedge \Psi \mid \neg\Psi \mid \mu X.\Psi(X) \mid \langle a \rangle \Psi \mid P \mid \top \mid \bot \mid \\ & \Psi \vee \Psi \mid [a]\Psi \mid \nu X.\Psi(X) \mid \Psi \textbf{ until } \Psi \mid \textbf{eventually } \Psi \mid \textbf{always } \Psi \end{aligned}$$

Zusätzlich zur oben definierten grundlegenden Anfragesyntax auf der Basis des μ-Kalkül werden die folgenden drei häufig verwendeten Muster in vereinfachter Form angeboten. Diese können auf der obigen Syntax mit Hilfe der Fixpunkt-Operatoren reduziert werden und bieten eine Vereinfachung in der Anfrageformulierung für den Nutzer der Dienstsuche.

- Ψ_1 **until** Ψ_2: Der gewünschte Prozess soll einen Zustand, in dem Ψ_2 gilt, erreichen und bis dahin gilt Ψ_1 in allen vorherig eingenommenen Zuständen.
- **eventually** Ψ: Der gewünschte Prozess soll einen Zustand erreichen, in dem Ψ gilt.
- **always** Ψ: In jedem Ausführungspfad eines gewünschten Prozesses gilt Ψ in allen Zuständen. Dabei kann es weitere Beschränkungen bzgl. des Pfades geben.

Zu weiteren Details zur Syntax, sowie der formalen Semantik des μ-Kalkül, wird an [Sti01, BS01] verwiesen. Weitere Informationen zur (semantischen) Modellierung der Prozessressourcen, wie z.B. der Modellierung der Eigenschaften einer Aktivität a, kann in [Aga07, ALS09] gefunden werden. Tabelle 3.2 gibt einen Überblick über die formale Semantik die auf einem Transitionssystem, d.h. für eine Prozessbeschreibung, definiert ist. Da die Zustände eines Transitionssystems in Form einer Ontologie beschrieben werden, stellt eine Proposition P ein Axiom in der Beschreibungslogik dar. Diese Proposition kann dann in einen Zustand evaluiert werden, indem bestimmt wird, ob diese Proposition in der entsprechenden Zustandsontologie gilt. Angewendet auf ein Transitionssystem eines Prozesses liefert die Evaluation einer Proposition P dann die Menge $\mathcal{V}_{Prop}(P)$ aller Zustände des Transitionssystems, in denen P gilt. Die Interpretation $\mathcal{V}_{Prop} : Prop \rightarrow \mathcal{P}(\mathcal{S})$ ist definiert auf einem Transitionssystem für die Menge atomare Propositionen $Prop$.

3.1.3 Dienstsuche zur Anfragezeit

In diesem Abschnitt wird die Methodik zur Dienstsuche eingeführt. Dazu werden existierende Dienstbeschreibungen einzeln gegen eine Dienstanfrage abgeglichen. Ziel des automatischen Vergleiches ist es zu bestimmen, ob ein angebotener Dienst die Anforderungen aus der Dienstanfrage erfüllt, oder

Tabelle 3.2: Formale Semantik des μ-Kalküls

Name	**Syntax**	**Semantik**
True	$\top$	$\mathcal{S}$
False	$\bot$	$\emptyset$
Proposition	P	$\mathcal{V}_{Prop}(P)$
Konjunktion	$\Psi_1 \wedge \Psi_2$	$[\![\Psi_1]\!]_{\mathcal{V}} \cap [\![\Psi_2]\!]_{\mathcal{V}}$
Negation	$\neg\Psi$	$\mathcal{S} \setminus [\![\Psi]\!]_{\mathcal{V}}$
Minimaler Fixpunkt	$\mu X.\Psi(X)$	$\bigcap\{S \subseteq \mathcal{S} \mid S \subseteq [\![\Psi]\!]_{\mathcal{V}[X:=S]}\}$
Disjunktion	$\Psi_1 \vee \Psi_2$	$[\![\Psi_1]\!]_{\mathcal{V}} \cup [\![\Psi_2]\!]_{\mathcal{V}}$
Existenzquantor	$\langle a \rangle \Psi$	$\{s \mid \exists t : s \xrightarrow{a} t \wedge t \in [\![\Psi]\!]_{\mathcal{V}}\}$

nicht. Nachdem alle Dienste abgeglichen wurden, liefert die Dienstsuche eine Menge von Diensten zurück, die den Anforderungen der Anfrage genügen. Der Abgleich zwischen Anfrage und den angebotenen Dienstbeschreibungen wird in vollem Umfang zur Zeit der Anfragestellung berechnet.

Die Evaluierung der Anforderungen wird im Transitionssystem eines jeden angebotenen Dienstes berechnet. Dazu wird in einem ersten Schritt aus der Beschreibung des Dienstverhaltens das entsprechende Transitionssystem, wie im Folgenden beschrieben, aufgebaut. Anschließend werden die funktionalen Anforderungen anhand des Transitionssystem verifiziert.

Konstruktion des Transitionssystem Das Verhalten eines Dienstes sei definiert durch den Aufruf $@A\{\mathsf{x_1}{:}\mathsf{t_1}, \ldots, \mathsf{x_n}{:}\mathsf{t_n}\}$ des beschriebenen Prozesses P. Der Bezeichners A enthält eine Referenz zur Dienstbeschreibungsontologie O_D, welche die Terminologie der Domäne zur Beschreibung des Dienstes in der TBox (terminologischer Teil einer Ontologie), sowie die nichtfunktionalen Eigenschaften des Dienstes in der ABox (Teil der Ontologie mit Faktenwissen), modelliert. Ausgehend von der Verhaltensdefinition P des Agentenbezeichners A werden eine Menge von Transitionssystemen konstruiert. Für jeden Zustand des Systems wird eine OWL Ontologie zu dessen Beschreibung erstellt. Diese Zustandsontologien erben das Domänenwissen der Ontology O_D. Die Transitionen zwischen Zuständen sind entweder vom Typ Eingabe-, Ausgabe-Aktivität oder lokale Operation.

Der Startzustand s_0 wird wie folgt kreiert. Eine neue Zustandsontologie O_{s_0} wird erzeugt und das Vokabular der Domäne (alle TBox Axiome der Ontologie O_D) wird hinzugefügt. Jedes Argument x_i des Aufrufs von A wird

als Individuum $\mathsf{x_i}$ der Ontologie O_{s_0} hinzugefügt und als Mitglied der Klasse $\mathsf{t_i}$ spezifiziert. Letztere Information modelliert die Parametertypen. Weiterhin werden Beziehungen zwischen den Individuen $\mathsf{x_1}, \ldots, \mathsf{x_1}$ untereinander durch OWL Object Properties und sonstige Eigenschaften durch OWL Datatype Properties zu der ABox der Ontologie O_{s_0} hinzugefügt.

Die folgenden Zustände, als auch die Transitionen zwischen den Zuständen, werden gemäß der Ausführungssemantik des Prozessausdrucks P erzeugt. Im Fall einer Sequenz $P = \pi.Q$ wird eine Transition π zwischen dem aktuellen Zustand s_P zu einem neuen Folgezustand s_Q eingeführt. Die Beschreibung O_{s_Q} des neuen Zustand s_Q wird von der Beschreibung O_{s_P} in Abhängigkeit der Transition abgeleitet. Zum Beispiel werden bei einer Eingabe-Aktivität die Beschreibung O_{s_P} des vorherigen Zustandes unverändert übernommen und die Parameter der Eingabe als weitere Individuen in O_{s_Q} hinzugefügt. Bei einer lokalen Operation, die durch eine Menge von Änderungen beschrieben ist, erhält man die Beschreibung O_{s_P}, indem die durch Axiome beschriebene Änderungen (z.B. Löschen, Hinzufügen, Aktualisierung von Individuen) auf die Beschreibung O_{s_P} angewendet werden.

Die Konstruktion der Transitionssysteme wird im Fall einer parallelen Komposition $P = \prod_{1 \leq i \leq n} P_i$ rekursiv auf den Komponenten P_i für alle möglichen Ausführungspfade fortgesetzt. Die Konstruktion wird analog für eine Auswahl $P = \sum_{1 \leq i \leq n} P_i$ rekursiv ausgeführt.

Verifikation Funktionaler Anforderungen Gegeben sei mit L_{D_w} ein Transitionssystem, welches nach obiger Methode aus einer Beschreibung D_w eines Dienstes w abgeleitet wurde. Eine Anfrage $\mathcal{R}$ umfasst die Anforderung Ψ an das gewünschte Dienstverhalten. Atomare Ausdrücke von Ψ werden wie im Folgenden beschrieben ausgewertet. Zusammengesetzte Formeln werden zunächst in atomare Ausdrücke aufgebrochen und einzeln ausgewertet. Anschließend werden die Ergebnisse der atomaren Formeln rekursiv entsprechend der formalen Semantik aggregiert.

- Falls $\Psi = \top$, dann werden alle Zustände $\mathcal{S}$ des Transitionssystems $L_{D_w} = (\mathcal{S}, T, \rightarrow)$ zurückgegeben.
- Falls $\Psi = \bot$, dann wird eine leere Menge von Zuständen zurückgegeben.
- Falls $\Psi = P$, dann werden all diese Zustände aus $\mathcal{S}$ zurückgegeben, in denen die Proposition P gilt. Dazu iteriert man über die Zustände $\mathcal{S}$ des

Transitionssystems L_{D_w}. Ein Zustand $s \in \mathcal{S}$ wird zur Ergebnismenge hinzugefügt wenn die Daten-Abfrage[1] P als wahr in der Zustandsbeschreibung (OWL Ontologie) O_s ausgewertet werden kann. Wenn die Auswertung einer Daten-Abfrage an eine Ontologie O_s eine nicht-leere Ergebnismenge liefert, dann gilt die Proposition P im Zustand $s \in \mathcal{S}$, anderenfalls nicht.

- Falls $\Psi = \langle a \rangle \Psi'$, dann wird zunächst nach dem Typ der Aktivität a differenziert. Gibt es eine Transition vom Zustand s in den Nachfolgezustand t vom gleichen Typ wie die gewünschte Aktion a und sind die gewünschten Eigenschaften der Aktion (wie z.B. Eingabeparameter) erfüllt und gilt Ψ' im Zustand t, dann wird der Zustand s zur Ergebnismenge hinzugefügt.

Nach Abschluss der Verifikation einer Anforderung Ψ bleibt zu prüfen, ob sich der Startzustand des Transitionssystems in der Ergebnismenge befindet. Falls dies der Fall ist, genügt die gegebene Prozessdefinition allen Anforderungen Ψ bzgl. des gewünschten Dienstverhaltens, sonst nicht.

Verifikation Nicht-Funktionaler Anforderungen Wie bereits bei der Dienstmodellierung beschrieben, werden die Werte der nicht-funktionalen Eigenschaften in der Domänen-Ontologie O_D des Dienstes modelliert. Anforderungen an die nicht-funktionalen Diensteigenschaften sind durch Daten-Abfragen spezifiziert. Falls das Individuum, welches den Wert eines NFPs modelliert, sich in der Antwortmenge einer Abfrage befindet, ist diese nicht-funktionale Anforderung erfüllt.

3.1.4 Implementierung und Effizienz der Dienstsuche zur Anfragezeit

Der vorgestellte Ansatz zur Dienstsuche wurde mittels der Programmiersprache Java implementiert. Die Dienstsuche-Komponente baut auf zahlreiche andere Komponenten auf. Zum Beispiel wurde zur Dienstmodellierung und der Modellierung des Verhaltens durch Prozessausdrücke die ent-

[1] Derzeit unterstützt die auf dem *HermiT Ontologie Reasoner* basierende Implementierung nur konjunktive Abfragen. Es ist geplant die Implementierung so zu erweitern, dass auch ausdrucksstärkere Abfragen unterstützt werden können.

Tabelle 3.3: Verwendete Ontologien zur Modellierung des Domänen-Wissens der synthetisierten Dienstbeschreibungen.

Name	Konzepte	Relationen
Semantic Web Conference Ontology[2]	107	35
Semantic Web Research Community Ontology[3] [SBH+05]	71	48
Pizza Ontology[4]	100	8

sprechenden Java-Programmierschnittellen, die bereits im Rahmen von WisNetGrid entwickelt wurden, wiederverwendet. Das Dienstverzeichnis, welches in [AHM10] dokumentiert ist, wurde zur Speicherung der semantischen Dienstbeschreibungen verwendet. Zusätzlich wird in der Suchkomponente der HermiT Ontologie Reasoner für die Sprache OWL 2 DL verwendet.

Für eine erste Leistungsüberprüfung des vorgestellten Suchansatzes werden verschiedene Anfragen auf mehrere tausend Beschreibungen komplexer Dienste ausgewertet. Ein Generator synthetisierte diese Dienstbeschreibungen aus verschiedenen Domänen. Ausgehend von gegebenen Domänen-Ontologien wurden Verhaltensbeschreibungen mit einer zufälligen Anzahl von Aktivitäten mit Parametern und Eigenschaften erzeugt, basierend auf dem Vokabular aus der Domänen-Ontologie (siehe Tabelle 3.3).

Jeder beschriebene Dienst setzt sich aus bis zu 10 Aktivitäten verschiedenen Typs zusammen. Eingabe- und Ausgabe-Aktivitäten besitzen bis zu 5 semantisch beschriebene Parameter. Die Effekte der lokale Aktivitäten sind durch bis zu 3 Änderungsaxiome beschrieben. Tabelle 3.4 zeigt die vollständigen Details des Experiments. Es wurden ebenfalls nicht-funktionale Eigenschaften modelliert. Ausgehend von den Klassifikation typischer NFPs in [OEH05] wurde jeder Dienst von bis zu 5 weiteren NFPs beschrieben.

Bei einer steigenden Anzahl von Dienstbeschreibungen wurden die benötigten Zeiten zur Beantwortung der Suchanfrage gemessen (vgl. Abb. 3.2). Dafür wurden verschiedene Anfragen getestet, zum Beispiel: (Q1) eine Proposition über die Prozessargumente des Aufrufs sowie deren Beziehungen, (Q2) eine Konjunktion von schließlich konsumierten Eingabeparametern and schließlich ausgegebenen Ausgabeparametern (siehe unten), (Q3) die stetige

[2]Semantic Web Conference Ontology Version 895 vom 11.05.2009, `http://data.semanticweb.org/ns/swc/ontology`

[3]Semantic Web for Research Communities Ontology Version 0.7.1 vom 22.06.2007

[4]Pizza Ontology Version 1.5 vom 12.02.2007, `http://www.co-ode.org/ontologies/pizza/`

Tabelle 3.4: Eigenschaften synthetisierter Dienstbeschreibungen.

Anzahl der Dienste	5000 ... 40000
Anzahl der NFP pro Dienst	< 5
Anzahl der Prozessargumente pro Dienst	≤ 3
Anzahl der Beziehungen zw. Prozessargumenten	≤ 3
Anzahl der Aktivitäten pro Dienst	≤ 10
Anzahl der Eingabe-/Ausgabe-Parameter pro Eingabe-/Ausgabe-Aktivität	≤ 3
Anzahl der Beziehungen zw. Eingabe-/Ausgabe-Parameter	≤ 3
Anzahl der Änderungsaxiome pro lokaler Aktivität	≤ 3

Möglichkeit zu haben, sich auszuloggen nachdem der Nutzer sich eingeloggt hatte (d.h., nachdem seine Login Berechtigung eingegeben wurden).

Beispiel: Anfrage Q2.

$$\begin{aligned} \Psi &= \textbf{eventually}\ \langle \mathsf{c[conference]} \rangle \top \wedge \textbf{eventually}\ \langle \mathsf{c[author]} \rangle \top \\ &\quad \wedge \textbf{eventually}\ \langle \mathsf{c} \langle \mathsf{location} \rangle \rangle P \\ P &= \mathsf{swc{:}hasLocation(conference, location), swc{:}hasAttendee(conference, author)} \end{aligned}$$

Bei der Bearbeitung einer Anfrage überprüft die Verifikationsmethode für jede verfügbare Dienstbeschreibung ob sie ein Modell für die Anfrage ist. Dies wird für jede Anforderung an eine (funktionale und nicht-funktionale) Diensteigenschaft aus der Anfrage getan. Die Verifikation wurde auf 10 Knoten verteilt um große Ladezeiten der Zustandsontologien eines jeden Zustandes der Prozessbeschreibungen zu kompensieren. Daher zeigen die Ergebnisse nur die Zeit zur Berechnung der Verifikation.

Die Messresultate in Abbildung 3.2 zeigen eine Notwendigkeit zur Reduktion der Komplexität bei der Verifikation zur Anfragezeit. Die rasch steigenden Antwortzeiten bei steigender Anzahl der verfügbaren Dienstbeschreibungen machen den vorgestellten Ansatz ineffizient, v.a. im Anbetracht der Szenarien zum Einsatz der Suche, wie zum Beispiel der automatischen Komposition aber auch für Endbenutzer bei der Bedienung einer Such-Oberfläche. Bereits bei 40.000 Diensten benötigt die Auswertung der einfachen Anfragen bis zu 900 Sekunden. Dies ist für die Benutzung des Systems nicht akzeptabel. Daher stellen wir im folgenden Kapitel einen Ansatz zur Effizienzsteigerung für diese Art der Dienstsuche vor und werden dann die Ergebnisse der Verbesserungen vorstellen.

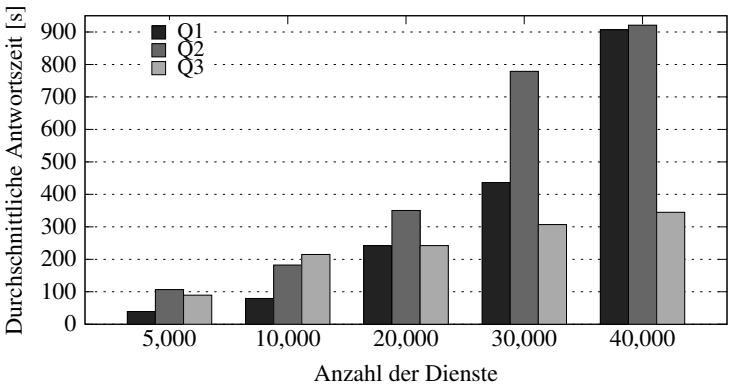

Abbildung 3.2: Durchschnittliche Antwortzeit in Sekunden für verschiedene Anfragen.

3.2 Klassifikationsbasierte Dienstsuche

In diesem Kapitel führen wir das Konzept der Dienstklassifikation ein. Eine Dienstklasse ist ein benanntes Anfragefragment und repräsentiert eine Menge von Diensten, die sich durch ähnliche Eigenschaften auszeichnen. Eine Dienstklasse ist formal definiert durch die Verwendung des Anfrageformalismus (siehe Abschnitt 3.1.2) und ist für Endbenutzer verständlich durch ein Schlüsselwort als deskriptiven Klassennamen beschrieben.

Dieser Ansatz besitzt mehrere Vorteile die in diesem Kapitel noch detailliert diskutiert werden. Das Potential der Automatisierung wird durch die formale Definition der Klassen erhöht und die Verwendung unseres Ansatzes wird durch die Klassennamen vereinfacht. Dazu gehört, dass die formale Definition der Klassen eine Klassenhierarchie impliziert. Diese Hierarchie kann zur effizienteren Dienstsuche ausgenutzt werden und kann im Vergleich zu einer manuell verwalteten Taxonomie keine Widersprüche enthalten. Außerdem können existierende Dienstbeschreibungen automatisch und korrekt klassifiziert werden. Wir werden ebenfalls aufzeigen, wie die Klassen zur vereinfachten Modellierung der Dienstbeschreibungen und Anfragen verwendet werden können. Letztlich wird durch die Veränderung der Klassen die Ausdrucksmächtigkeit der Dienstbeschreibungen um die Möglichkeit zur Modellierung der Unbestimmtheit vergrößert.

3.2.1 Dienstklassifikation

Der hybride Ansatz zur Modellierung der Dienstklassen zielt sowohl auf eine Effizienzsteigerung der Suche, als auch der Vereinfachung des Modellierungsaufwands für Endbenutzer. In den folgenden Absätzen stellen wir zunächst Dienstklassen und die Klassenhierarchie vor. Wir zeigen, dass die Verwendung dieses Ansatzes nicht auf eine zentrale global verwaltete Klassenhierarchie beschränkt ist. Danach wird der Einfluss auf die Dienstmodellierung und Anfragemodellierung, sowie auf den Ansatz des Dienstabgleich diskutiert. Schließlich wird die motivierende Effizienzsteigerung der Dienstsuche anhand weiterer Experimente evaluiert.

Klassendefinition

Eine Dienstklasse beschreibt eine Menge von Diensten mit ähnlichen Eigenschaften. Wird ein Dienst einer Klasse zugeordnet, bedeutet das, dass der Dienst ebenfalls durch die Eigenschaften der Klasse charakterisiert ist. Eine Dienstklasse $c = [def(c), name(c)]$ ist beschrieben durch eine formale Klassendefinition $def(c)$ und einen deskriptiven Klassennamen $name(c)$. Der Klassenname ist eine wörtliche Darstellung der durch die Klasse zugesicherten Diensteigenschaften und wird nur zur Steigerung der Verwendbarkeit benutzt. Durch Namen (Schlüsselworte) beschriebene Klassen wurden häufig bei Produkt- und Dienst-Taxonomien verwendet, allerdings erlauben die Schlüsselworte kein automatisches Schließen über diese Diensteigenschaften, welche durch die Dienstklassen repräsentiert werden. Daher wird eine zusätzliche formale Definition $def(c)$ notwendig und auch im vorgestellten Ansatz zur Dienstsuche angewendet.

Der Anfrageformalismus zur Beschreibung der NFRs $\mathcal{N}$, sowie der Anforderungen Ψ an das Dienstverhalten, wird zur Definition der Dienstklassen verwendet. NFRs werden in Ausdrücken der Beschreibungslogik beschrieben. Klassen definieren Intervalle bzw. Mengen von Literalen, Werten eines Datentyps oder Nominalen. Um zum Beispiel Dienstklassen zur Unterscheidung der Antwortzeiten der Dienste ($\mathsf{respTimeMS}$) zu erstellen, können diese wie folgt definiert werden:

$$\begin{aligned} \mathsf{FastService} &\equiv \mathsf{Service} \sqcap \exists \mathsf{respTimeMS}.\top \sqcap \forall \mathsf{respTimeMS}.\, \leq_{100} \\ \mathsf{SlowService} &\equiv \mathsf{Service} \sqcap \exists \mathsf{respTimeMS}.\top \sqcap \forall \mathsf{respTimeMS}.(>_{100} \sqcap \leq_{1000}) \end{aligned}$$

Die Anforderungen an das Dienstverhalten werden in der μ-Kalkül-basierten Anfragesyntax beschrieben. Die in den Klassendefinitionen referenzierten Prozessressourcen werden in einer Domänen-Ontologie formal beschrieben. Diese Ontologie wird der Klassendefinition angehängt.

Klassendefinitionen $\mathcal{R}_C$ können aus den verhaltensbasierten (Ψ) und nichtfunktionalen ($\mathcal{N}$) Anforderungen, sowie aus den existierenden Klassen (c), komponiert werden. Ψ verweist dabei auf den Beschreibungssyntax der Anforderungen an das Dienstverhalten aus Def. 3.1.2.

Syntax der Definition $\mathcal{R}_C$ von Dienstklassen

$$\mathcal{R}_C := \mathcal{R}_C \wedge \mathcal{R}_C \mid \mathcal{R}_C \vee \mathcal{R}_C \mid \neg\mathcal{R}_C \mid \mathcal{N} \mid \Psi \mid c$$

Beispiel Die Dienstklasse c_{ts} der Dienste mit der Ticketverkaufsfunktionalität ist beschrieben durch den Klassennamen

$$name(c_{ts}) = "TicketSellingServices"$$

und der Definition

$$def(c_{ts}) = \textbf{eventually}\ \mathsf{c}\langle\mathsf{ticket}\rangle \wedge \textbf{eventually}\ \mathsf{c}\langle\mathsf{receipt}\rangle \vee \textbf{eventually}\ \mathsf{c}\langle\mathsf{abort}\rangle.$$

Diese Definition bedeutet, dass ein Dienst dieser Klasse zu einem beliebigen Zeitpunkt der Prozessausführung ein Ticket ticket mit einer Bestätigung receipt ausgibt. Alternativ kann auch eine Abbruchbestätigung abort zurückgegeben werden. Die drei Ausgabeparameter sowie der zu verwendende Kommunikationskanal c seien in einer der Klassendefinition angehängten Ontologie weiter wie folgt beschrieben:

$$\mathsf{ex{:}HTTPConnection(c)}$$
$$\mathsf{ex{:}HTTPConnection} \sqsubseteq \mathsf{ex{:}MessageExchangeResource}$$
$$\mathsf{ex{:}Ticket(ticket)}$$
$$\mathsf{ex{:}Ticket} \sqsubseteq \mathsf{gr{:}ProductOrService}^{5}$$

Klassen-Mitgliedschaft Ein Dienst w ist Mitglied einer Dienstklasse c genau dann wenn die Dienstbeschreibung D_w der Klassendefinition $def(c)$ genügt. Weil $def(c)$ eine normale Dienstanfrage gemäß Definition 3.1.2 darstellt, kann die in Abschnitt 3.1.3 vorgestellte Technik zum Dienstabgleich ebenfalls zur automatischen Bestimmung der Mitgliedschaft eines Dienstes w in einer Klasse c verwendet werden.

Automatische Erkennung der Beziehung zwischen Klassen

Eine Klassenhierarchie $\mathcal{C} = (C, H)$ ist eine Graph-Struktur über eine endliche Menge C von Dienstklassen und eine Unterklassen-Beziehung $H \subseteq C \times C$ über den Dienstklassen C. Eine Klasse $c_i \in C$ ist eine Unterklasse $c_j \in C$, geschrieben als $(c_i, c_j) \in H$, genau dann, wenn alle Dienste der Klasse c_i (d.h. Dienstbeschreibungen die Modell der formalen Definition $def(c_i)$ der Klasse c_i sind) auch Mitglied der Klasse c_j (d.h. Dienstbeschreibungen sind ebenfalls Modell der formalen Definition $def(c_j)$) sind. H ist eine totale Ordnung die nicht auf eine Baumstruktur begrenzt ist, da Maschen möglich sind.

Es sei eine Menge C von Dienstklassen gegeben. Dann kann eine Klassenhierarchie $\mathcal{C} = (C, H)$ automatisch von den Definitionen der Klassen C abgeleitet werden. Zur Berechnung der Relation H werden Paare von Klassendefinitionen $def(c_i)$, $def(c_j)$ mit $\{c_i, c_j\} \subseteq C$ und $c_i \neq c_j$ miteinander verglichen. Dabei wird die Enthaltensein- bzw. die Simulationsbeziehung für jede gemeinsame Diensteigenschaft, die in c_i als auch in c_j beschrieben ist, bestimmt. Eine gemeinsame Eigenschaft ist gegeben falls beide Eigenschaften in $def(c_i)$ und $def(c_j)$ (i) sich auf die Verhaltensanforderung beziehen oder (ii) sich auf eine NFR beziehen und die entsprechenden Properties in den Ontologien sind entweder äquivalent oder es existiert eine Relation zwischen beiden Properties in einer Vermittlungsontologie[6] [SS04].

Im ersten Fall (i) wird die Technik aus [Koz83] zur Berechnung der Relation H angewandt. μ-Kalkül Ausdrücke werden in eine Normalform transformiert und die Relation zwischen beiden Ausdrücken kann ermittelt werden. Eine Klasse c_i ist eine Unterklasse einer Klasse c_j bzgl. der Anforderungen an das Dienstverhalten Ψ_i und Ψ_j genau dann, wenn jedes Modell von Ψ_i ein Modell von Ψ_j ist. Die Vollständigkeit des Ansatzes zur Axiomatisierung des μ-Kalkül aus [Koz83] wurde in [Wal95] gezeigt. Die Axiomatisierung bildet die Basis für die Überprüfung ob ein Ausdruck eine Unter-Formel eines anderen Ausdrucks ist. Wir wenden diese Regeln zur Berechnung der Unterklassen-Beziehung H an und halten somit auch die Klassenhierarchie konsistent.

3.2.2 Verwendung von Dienstklassen in Dienstbeschreibungen

Wir erweitern den Formalismus der Dienstmodellierung um die Möglichkeit zur Annotation der Dienste mit Klassen. Die Klassen C einer Klassenhierarchie $\mathcal{C}$ seien in einer Ontologie $O_{\mathbb{D}}$ im Dienstverzeichnis $\mathbb{D}$ modelliert. Jede Dienstklasse $c \in C$ wird durch ein Konzept c und jede Beziehung $(c_i, c_j) \in H$ zwischen zwei Dienstklassen $c_i \in C$ und $c_j \in C$ durch eine Relation $\mathsf{c_i} \sqsubseteq \mathsf{c_j}$ in der Ontologie $O_{\mathbb{D}}$ repräsentiert. Ein Dienst w wird einer Menge von Dienstklassen $C(w) \subseteq C$ zugewiesen indem das Individuum w vom Typ $\mathsf{sm{:}Service}$ als Element der Ontologie-Klassen $C(w) \subseteq C$ deklariert wird. Dadurch ist eine Annotation des Dienstes mit einer Menge von Dienstklassen

[6] Die Vermittlung zwischen verschiedenen Ontologien fusioniert und ordnet Entitäten beider Ontologien in eine gemeinsame Ontologie ein.

als eine Konjunktion der funktionalen und nicht-funktionalen Eigenschaften der jeweiligen Klassendefinitionen zu interpretieren.

Beispiel: Klassifikation eines Dienstes w in der Ontologie $O_{\mathbb{D}}$

$$\mathsf{sm{:}FastService(w)}$$
$$\mathsf{sm{:}TicketSellingService(w)}$$
$$\mathsf{sm{:}FastService} \sqsubseteq \mathsf{sm{:}Service}$$
$$\mathsf{sm{:}SellingService} \sqsubseteq \mathsf{sm{:}Service}$$
$$\mathsf{sm{:}TicketSellingService} \sqsubseteq \mathsf{sm{:}SellingService}$$

Expressivität Die Annotation der Dienstbeschreibungen mit Dienstklassen ermöglicht es vage Aussagen über die beschriebenen Eigenschaften zu treffen, da der Anfrageformalismus zur Klassendefinition verwendet wird. Bisher betrachteten wir Dienstbeschreibungen als eine Menge von Schlüssel-Wert-Paaren, jeweils mit konkreten Werten (vgl. Abschnitt 3.1.1). Dagegen spezifiziert man in Anfragen einen Wertebereich oder -menge je gewünschter Eigenschaft.
Die Verwendung vager Aussagen in Dienstbeschreibungen wird benötigt, wenn während der Modellierung der Dienstbeschreibung konkrete Werte einer Eigenschaft nicht bekannt sind, nicht garantiert werden können oder nicht bekannt gegeben werden sollen. Zum Beispiel kann die Angabe der Antwortzeit eines Dienstes durch ein Intervall von Werten geeigneter beschrieben werden.
Gleichermaßen beschränken Dienstklassen auch das Verhalten der Dienste und können ebenfalls ausdrücken welche Funktionalität ein Dienst nicht anbietet. Zum Beispiel kann eine Klassendefinition ausdrücken, dass keine Kreditkarteninformation als Eingabe benötigt wird. Dies konnte bisher nicht explizit angegeben werden, da der Formalismus zur Beschreibung des Dienstverhaltens auf der Open World Assumption basiert.

Vereinfachungen Die Komplexität der Beschreibungen nicht-funktionaler Eigenschaften, sowie des komplexen Verhaltens, wird durch die Wiederverwendung existierender Klassen reduziert. Es können formale Definitionen typischer Diensteigenschaften wiederverwendet und weiter verfeinert werden, anstatt diese in vollem Umfang neu zu erstellen. Bei einer gemeinschaftlichen Dienstmodellierung haben oft nur Domänen- oder Prozessmodellie-

rungsexperten das Vorwissen zur Modellierung der Dienstspezifika die über den Bereich der Dienstklassen hinaus gehen.

Da Klassennamen (deskriptive Bezeichnung der Klassen) es den Benutzern ermöglichen die Klassenhierarchie auch ohne Verständnis der formalen Definitionen zu verwenden, können Dienstbeschreibungen auch ohne jegliches Verständnis der verwendeten Formalismen (π-Kalkül Prozessalgebra, Beschreibungslogiken) erstellt werden. Die Komplexität, als auch die Ausmaße einer Dienstbeschreibung, sowie der Aufwand zu deren Erstellung, werden durch die Nutzung der Dienstklassen reduziert. Das dazu notwendige Auffinden geeigneter Dienstklassen verbleibt eine manuelle Aufgabe, die das Durchstöbern der Klassenhierarchie oder eine Schlüsselwort-basierte Suche als zusätzlichen aber dennoch weniger schwierigen Aufwand für den Benutzer enthält.

Beispiel Es sei D_w die Beschreibung eines Dienstes w zur Buchung von Flugtickets. Das Verhalten ϕ ist so definiert, dass der Prozess mit der Eingabe von Start- und Zielort (from, to) sowie des gewünschten Reisetages (date) beginnt. Die Eingabe benutzt den Kommunikationskanal c zwischen Dienstaufrufer und Dienstanbieter und ist in der Domänen-Ontologie O_D beschrieben als eine Kommunikation, z.B., basierend auf einem verschlüsselten Web-Protokoll. Die Eingabeparameter werden vom Dienstanbieter verarbeitet. Diese interne Verarbeitung wird durch die lokale Operation Δ_{getF} abstrakt modelliert. Diese Operation ermittelt geeignete Flüge flights, die dann zurück an den Aufrufenden, z.B. zur Anzeige im Web Browser, gesendet werden. Anschließend folgt die Bezahlungsprozedur, die als Teilprozess P_{pay} angeben ist, und das Ticket eticket zum gewählten Flug, sowie eine Bestätigung receipt des Kaufs, wird über c ausgegeben.

$$\phi \;=\; \mathsf{c}[\mathsf{from},\mathsf{to},\mathsf{date}].\Delta_{getF}.\mathsf{c}\langle\mathsf{flights}\rangle.\mathsf{c}[\mathsf{flight}].P_{pay}.\mathsf{c}\langle\mathsf{eticket},\mathsf{receipt}\rangle.\mathbf{0}$$

Die Operation Δ_{getF} zur Ermittlung verfügbarer Flüge flights zu den gegebenen Eingaben ist durch eine Menge von Ontologie-Axiomen beschrieben. Diese Axiome beschreiben die Effekte der Ausführung wie folgt. Es werden jeweils nur Fakten hinzugefügt, welche durch den Typ ADD des Axioms gekennzeichnet sind.

$$\Delta_{getF} = \Big\{ \quad [ADD, \mathsf{ex{:}FlightList(flights)}], [ADD, \mathsf{ex{:}hasEntry(flights, f)}]$$
$$[ADD, \mathsf{ex{:}Flight(f)}], [ADD, \mathsf{ex{:}hasDate(flight, date)}]$$
$$[ADD, \mathsf{ex{:}hasStart(f, from)}], [ADD, \mathsf{ex{:}hasEnd(f, to)}] \Big\}$$

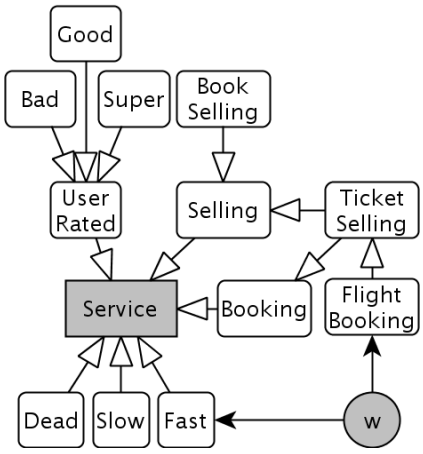

Abbildung 3.3: Ausschnitt aus einer Klassenhierarchie mit Dienst w als Mitglied der jeweiligen Klassen.

Während sich nicht immer jegliche Spezifika eines Dienstes w durch eine Menge von Annotationen mit generischen Dienstklassen darstellen lässt, kann in diesem Beispiel eine Dienstklasse c mit $name(c) = "FlightTicketBooking"$ und $def(c) = \Psi_c$ die gebräuchliche Funktionalität von Diensten zur Flugbuchung abdecken. Durch die Verwendung der einmalig zu definierenden Klassen reduziert sich der beispielhaft dargestellte Modellierungsaufwand des obigen Dienstes w.

Beispiel: Klassendefinition von Diensten zur Flugbuchung.

$$\Psi_c = (\langle \mathsf{c}[\mathsf{from}] \rangle \top \wedge \langle \mathsf{c}[\mathsf{to}] \rangle \top \wedge \langle \mathsf{c}[\mathsf{date}] \rangle \top) \wedge (\textbf{eventually}\ \langle \mathsf{c} \langle \mathsf{ticket} \rangle \rangle \wedge \textbf{eventually}\ \langle \mathsf{c} \langle \mathsf{receipt} \rangle \rangle \vee \textbf{eventually}\ \langle \mathsf{c} \langle \mathsf{abort} \rangle \rangle)$$

Die Klasse c definiert das gemeinsame Verhalten der Flugbuchungsdienste. Es werden anfangs Eingaben konsumiert (Startort from, Zielort to, Flugdatum date) und schließlich werden ein Flugticket ticket und eine Rechnung receipt (oder alternativ eine Abbruchbestätigung abort) ausgegeben. Gleichermaßen können Klassen bezüglich nicht-funktionaler Eigenschaften, wie zum Beispiel FastService aus Abschnitt 3.2.1 angewendet werden. Abbildung 3.3 zeigt einen Ausschnitt einer möglichen Hierarchie über Klassen dieser Domäne des gegebenen Beispiels.

3.2.3 Verwendung von Dienstklassen in Dienstanfragen

Analog zur Dienstbeschreibung tritt eine Reduzierung des Modellierungsaufwands, sowie eine Vereinfachung der Anfragebeschreibungen, durch die Verwendung der Dienstklassen ein. Die oben genannten Vorteile gelten daher auch für die Anfragemodellierung. Bei Wiederverwendung der Dienstklassen können Benutzer schneller und einfacher Anfragen erstellen. Weiterhin erlaubt eine Klassenhierarchie, sowie die Klassifikation der Dienste in entsprechende Klassen, das Durchstöbern eines Katalogs mit existierenden Diensten, wenn die Klassenhierarchie z.B. baumartig angezeigt wird.

Der Formalismus zur Anfragebeschreibung wurde bereits so erweitert, dass er zusätzlich zu NFRs und Anforderungen an das Verhalten auch die Verwendung von Dienstklassen einschliesst (siehe Def. 3.2.1). Das folgende Beispiel zeigt eine Anfrage $\mathcal{R}$ für einen gewünschten Flugbuchungsdienst x mit einer Super Nutzerbewertung sowie schneller Antwortzeit. Diese Anforderungen an einen gewünschten Dienst können durch die Verwendung der Klassen aus der abgebildeten Hierarchie (Abb. 3.3) ausgedrückt werden. Das gewünschte Verhalten wird von der obigen Klassendefinition Ψc der Flugbuchungsdienste geerbt, könnte aber ebenso weiter verfeinert werden, um zum Beispiel die Modalitäten der Bezahlung weiter einzuschränken. Daraus lässt sich nun leicht erkennen, dass eine solche Anfrage mit gleichem Anforderungsumfang viel komplexer ohne die Verwendung der Dienstklassen ausfallen würde.

Beispiel: Anfrage $\mathcal{R}$ unter Verwendung von Dienstklassen.

$$\mathcal{R} \;=\; \mathsf{ex{:}Super(x)} \wedge \mathsf{ex{:}Fast(x)} \wedge \mathsf{ex{:}FlightTicketBooking(x)}$$

3.2.4 Verwendung verschiedener Klassenhierarchien

Wir gehen davon aus, dass die Verwendung einer globalen und zentral verwalteten Hierarchie von Dienstklassen, aus der eine Indexstruktur zur effizienten Dienstsuche abgeleitet werden kann, den Anforderungen bezüglich der Offenheit des Web nicht nachkommen kann. Da die Klassen in unserem Ansatz eine formale Definition besitzen, können neue Klassen automatisch in die Hierarchie der existierenden Klassen eingeordnet werden. Dabei können neue Klassen mit dem grundlegenden Formalismus (vgl. Syntax in Def. 3.1.2)

oder auch mit der erweiterten Syntax unter Einbeziehung bereits existierender Klassen konstruiert werden (vgl. Def. 3.2.1). Zum Beispiel ist die Klasse c_{fts} der Dienste zum Verkauf von Flugtickets als Erweiterung (d.h. Unterklasse) von der Klasse c_{ts} der Ticketverkaufsdienste definiert, indem c_{fts} wie folgt definiert wird:

$$def(c_{fts}) = c_{ts} \wedge \textbf{eventually } \mathsf{c[date]} \wedge \textbf{eventually } \mathsf{c[startLocation]}...$$

Unser Ansatz erlaubt es, dass ein Benutzer zur Formulierung einer Anfrage $\mathcal{R}$ eine Klassenhierarchie $\mathcal{C}_\mathcal{R} = (C_\mathcal{R}, H_\mathcal{R})$ benutzt, die verschieden von der Systemhierarchie $\mathcal{C}_{sys} = (C_{sys}, H_{sys})$ sei. Die Systemhierarchie bezeichnet dabei die im Dienstverzeichnis zur Indizierung verwendete Klassenhierarchie. Der anfragende Benutzer verwendet Klassen aus $C_\mathcal{R}$ in der Anfrage $\mathcal{R}$. Im Gegensatz zur traditionellen Dienstsuche, wie in Abschnitt 3.1.3 eingeführt, werden während der Beantwortung der Anfrage $\mathcal{R}$ die Klassen $C_\mathcal{R}$ zunächst gegen die Klassen C_{sys} der Systemhierarchie abgeglichen. Unter der Annahme dass es weniger Klassen C_{sys} als Dienstbeschreibungen im Dienstverzeichnis $\mathbb{D}$ gibt, bleibt ein Effizienzvorteil im Vergleich zur traditionellen Dienstsuche bestehen.

3.2.5 Erweiterung des Suchansatzes

Die Systemhierarchie $\mathcal{C} = (C, H)$ über den Dienstklassen C sei der Suchkomponente verfügbar. Während der Initialisierung der Suchkomponente werden die Dienstbeschreibungen im Dienstverzeichnis $\mathbb{D}$ klassifiziert und diese Klassifikationsinformation wird zur späteren Verwendung als Indexstruktur $i \subseteq C \times \mathbb{D}$ zusätzlich zu der Klassenhierarchie materialisiert. Dafür wird die Klassenzugehörigkeit jeder Dienstbeschreibung in $\mathbb{D}$ bestimmt. Falls eine Dienstbeschreibung $D_w \in \mathbb{D}$ mit einer Dienstklasse $c \in C$ annotiert ist, dann wird (c, D_w) zum Index i hinzugefügt. Sonst wird die Dienstbeschreibung D_w gegen die Klassendefinition $def(c)$ einer jeden Klasse $c \in C$ abgeglichen und für jede Klasse in $C' \subseteq C$, für die der Abgleich (vgl. Abschnitt 3.1.3) ein positives Ergebnis liefert, wird $\forall c' \in C' : (c', D_w)$ dem Index i hinzugefügt.

Der Index i über die Klassifikationsinformation erlaubt das unmittelbare Abrufen der Dienstbeschreibungen, die einer Dienstklasse zugewiesen wurden und erspart somit den Aufruf der teuren Methode zum Dienstabgleich zur Anfragezeit. Wir nehmen an, eine Dienstanfrage $\mathcal{R}$ spezifiziert die Anforderung, dass gewünschte Dienste Mitglied der Dienstklassen $C_\mathcal{R} \subseteq C$ sein

sollen und es sei zudem spezifiziert, dass $\mathcal{R}$ zusätzlich explizit modellierte Anforderungen stellt, welche nicht bereits durch die Klassen $C_{\mathcal{R}}$ abgedeckt wurden. Bei der Dienstsuche werden im ersten Schritt zur Beantwortung von $\mathcal{R}$ die Dienste $i(C_{\mathcal{R}})$, die Mitglied der Klassen $C_{\mathcal{R}}$ sind, vom Index i abgerufen. In einem zweiten Schritt werden die weiteren Anforderungen von $\mathcal{R}$ (die nicht durch die Klassendefinitionen abgedeckt wurden) nur mit den Diensten $i(C_{\mathcal{R}})$ aus dem ersten Schritt abgeglichen.

Durch die Einführung des Index i wird der Umfang der zur Anfragezeit abzugleichenden Anforderungen und der Suchraum reduziert. Mit Suchraum bezeichnen wir die Menge der Dienste für welche die Anforderungen einer Anfrage abgeglichen werden müssen. Wir nehmen an, dass ein Abrufen von Diensten einer oder mehrerer Klassen schneller als der Abgleich aller gewünschten Anforderungen an Diensteigenschaften zur Anfragezeit ist. Offensichtlich beeinflussen mehrere Faktoren, wie zum Beispiel die Komplexität der Klassendefinition, die Größe der Klassenhierarchie, sowie die Anzahl der Dienstbeschreibungen, die Gültigkeit dieser Annahme. Wir werden in der folgenden Evaluierung in Abschnitt 3.2.6 zeigen, dass der vorgestellte Dienstklassifikationsansatz sich signifikant positiv auf die mittlere Zeit zum Dienstabgleich während der Anfragezeit auswirkt.

3.2.6 Evaluierung der Dienstsuche

Die Experimente zur Messung der Effizienz der Dienstsuche aus Abschnitt 3.1.4 wurden für den klassifikationsbasierten Ansatz wiederholt. Basierend auf dem Open Directory Project[7] (ein Webverzeichnis zur Klassifikation von Webseiten) wurde eine beispielhafte Dienstklassenhierarchie für dieses Experiment erstellt. Das Projekt verwendet ein hierarchisches Schema zur Organisation von Webseiten-Einträgen im Verzeichnis. Einträge zu einem ähnlichen Thema sind in Kategorien gruppiert, welche dann auch noch in kleinere Gruppen unterteilt sind. Insgesamt umfasst das Verzeichnis zurzeit 787.501 Kategorien und Gruppen, davon haben wir die Kategorien mit einer maximalen Tiefe von 2 aus der Hierarchie extrahiert und in der OWL Ontologie $O_{\mathbb{D}}$ des Dienstverzeichnisses als OWL Klassen mit den Klassenbeziehungen modelliert. Danach wurden 524 Ontologie-Klassen modelliert, welche im folgenden Experiment die Dienstklassen darstellen.

[7] `http://www.dmoz.org/` (auch bekannt als Directory Mozilla)

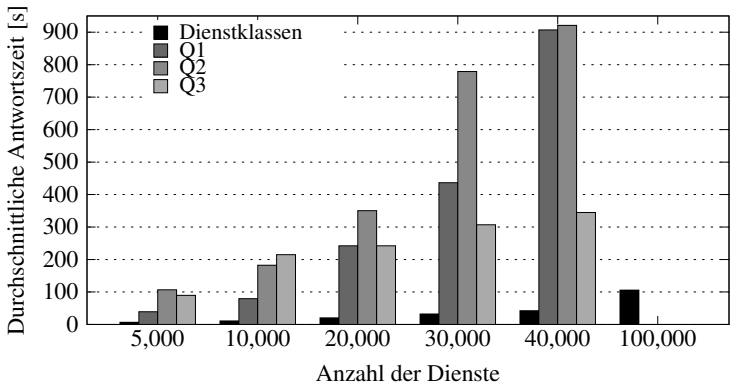

Abbildung 3.4: Durchschnittliche Zeit zum Abfragen der Dienste der Schnittmenge von 2 Dienstklassen (Balken in schwarz) im Vergleich zu den gemessenen Zeiten (graue Balken) des ersten Experimentes mit Q1, Q2, Q3 von Abbildung 3.2, Abschnitt 3.1.4.

Zu 40 von diesen Dienstklassen wurden die formalen Klassendefinitionen mit funktionalen sowie nicht-funktionalen Eigenschaftsbeschreibungen manuell spezifiziert. Diese Hierarchie wird dem Dienstverzeichnis als zusätzliche Information verfügbar gemacht und wird zur Formulierung der Anfragen in diesem Experiment verwendet. Die semantischen Dienstbeschreibungen werden wie im vorgehenden Experiment synthetisiert und werden in einem weiteren Schritt automatisch klassifiziert. Dazu wird die in Abschnitt 3.1.3 vorgestellte Methode zum Dienstabgleich zur Berechnung der Klassenzugehörigkeit jedes Dienstes angewendet. Diese Berechnung wird in der Initialisierungsphase ausgeführt und wird daher bei der Messung der Anfragebeantwortungszeit nicht berücksichtigt. Im Unterschied zu den vorherigen Experimenten können die Dienstanfragen nun auch Dienstklassen einer gegebenen Klassenhierarchie enthalten. Die verwendeten Beispielanfragen werden so modifiziert, dass vorhandene Klassen anstatt der jeweiligen expliziten Anfrageformulierung, die einer Klassendefinition entspricht, ersetzt. Diese Anfragen mit Dienstklassen werden nun wiederholt gegen die selbigen Dienstbeschreibungen des vorherigen Experiments im Verzeichnis ausgewertet. Im Idealfall können alle Anforderungen der Dienstanfrage durch die Dienstklassen abgebildet werden. Gemäß dieser Erwartungen ist die gemessene durchschnittliche Zeit zur Verifikation der Anfragen ausschließlich mit Dienstklassen signifikant kleiner. Abbildung 3.4 zeigt die durchschnittlichen Ergebnisse für Anfragen aus zwei konjunktiv verknüpften zufälligen Dienstklassen aus der Klassenhierar-

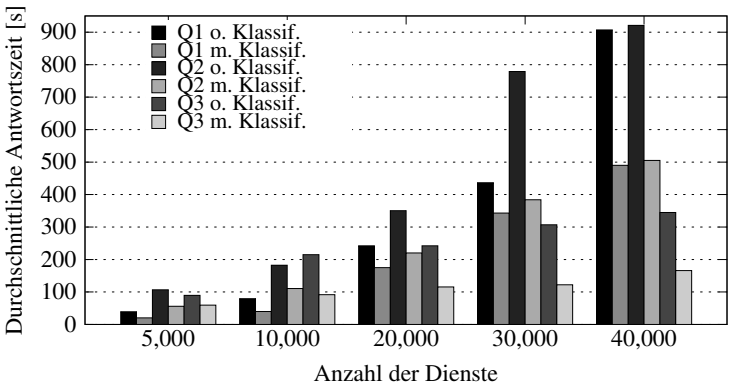

Abbildung 3.5: Durchschnittliche Antwortzeit in Sekunden für verschiedene Anfragen mit Dienstklassen im Vergleich zu den Ergebnissen der Dienstsuche ohne Verwendung der Dienstklassen aus Abbildung 3.2, Abschnitt 3.1.4.

chie. Diese Zeit, dargestellt durch den schwarzen Balken, unter Verwendung der Dienstklassen ist signifikant kleiner als bei der Dienstsuche ohne Dienstklassen, dargestellt durch die grauen Balken Q1, Q2 und Q3. Die gleichen funktionalen bzw. nicht-funktionalen Anforderungen können dabei durch die Dienstklassen abgeglichen werden. Wie ersichtlich wird, kann die Dienstsuche unter Verwendung der vorgestellten Dienstklassen auch für wesentlich mehr Dienstbeschreibungen im Dienstverzeichnis angewendet werden.

Während Abbildung 3.4 den Idealfall abdeckt, können die Dienstanfragen auch explizit modellierte Anforderungen, sowie die Nennung von Dienstklassen, enthalten. Dabei liegt die Zeit zur Verifikation der Anforderungen zwischen den Zeiten im Idealfall (bei ausschließlicher Verwendung von Dienstklassen) und den Zeiten gemessen im ersten Experiment bei ausschließlicher Verwendung von explizit formulierter Anforderungen. In Abbildung 3.5 werden die Beispiele aus Abschnitt 3.1.3 wieder aufgegriffen. Die Dienstklassen mit formaler Klassendefinition ersetzen die Teile der Anfragen Q1, Q2 und Q3, welche mit der Klassendefinition übereinstimmen. Die Messergebnisse in Abbildung 3.5 belegen die obige Vermutung: Auch bei der Vermischung von Dienstklassen und expliziter Anforderung in einer Anfrage wird die Effizienz der Dienstsuche durch die Verwendung einer Klassifikation verbessert.

Kapitel 4

Verwandte Arbeiten

Metadata von Ressourcen im Web, Diensten und Produkten sind ein weit verbreitetes, einfaches und nützliches Mittel, um großen Informationsmengen einer einfachen Strukturierung zu unterwerfen. Typischerweise werden Metadaten zur verbesserten Auffindbarkeit der Ressourcen [GGWW08] eingesetzt. Es gibt mehrere Metadaten-Vokabulare in unterschiedlichen Domänen, wie zum Beispiel Dublin Core [WKLW98], Good Relations [goo], Friend-of-a-friend [BM10], mit denen man die Probleme bezüglich der Heterogenität der Daten zu überwinden versucht. In dieser Arbeit wurden existierende Vokabulare wiederverwendet. Die vorgestellte Dienstsuche basierend auf Metadaten ist natürlich spezifisch für die im WisNetGrid benötigten Metadaten der Dienste. Dennoch können generische Werkzeuge zur Indexierung und effizienten Suche von und in Texten auch zur Dienstsuche basierend auf Metadaten als Erweiterung unseres Ansatzes verwendet werden.

In der vergangenen Zeit wurden einige Ansätze zur semantischen Beschreibung komplexer und zusammengesetzter Dienste vorgeschlagen. Der bekannteste Vertreter ist OWL-S [w3c, MBH+04], zu dem auch Ansätze zur Suche der mit dem OWL-S Service Profile beschriebenen atomarer Dienste entwickelt wurden [Syc03]. Das OWL-S Process Model benutzt ausschließlich OWL Ontologien zur Beschreibung der zusammengesetzten Dienste. Dies ist problematisch, da eine Ontologie die Semantik eines Prozesses nicht abbilden kann, d.h., sie erlaubt nicht, die durch den Dienst implementierte Dynamik abzubilden. Im Vergleich dazu wird bei der im WisNetGrid verwendeten semantischen Modellierung der Dienste eine Ontologie pro Zustand in der Ausführung zur Beschreibung der Ressourcen verwendet. Veränderungen in

den Aussagen über die Ressourcen zwischen verschiedenen Zuständen würden somit bei dem Ansatz mit OWL-S zu Widersprüchen führen.

Die Web Services Business Process Execution Language (WS-BPEL) ist ein verbreiteter Vertreter einer Prozessbeschreibungssprache [wsb07]. Allerdings liegt der Fokus dieser Sprache eher auf die Modellierung und Ausführung von Workflows innerhalb eines Unternehmens. Dagegen wird die formale Beschreibung der inter-organisatorischen und dezentralen Dienste mit komplexen Interaktionsmustern nicht betrachtet, welche allerdings für die im WisNetGrid anvisierte Komponierbarkeit der Dienste notwendig ist [AHM+09]. Außerdem wurden bisher für WS-BPEL, wie auch für OWL-S Process Model basierte Dienstbeschreibungen keine formalen Matchmaking- und Retrieval-Techniken in der Literatur vorgestellt.

Die Unified Service Description Language (USDL) ist ein abstraktes Beschreibungmodell für Dienste mit dem Fokus auf Eigenschaften, welche relevant im Umfeld von Geschäftprozessen sind. Die Sprache umfasst Module zur Modellierung nicht-funktionaler Eigenschaften (basierend auf der Arbeit von O'Sullivan [OEH05]) um die Qualität eines Dienstes zu beschreiben und auch Module zur funktionalen Beschreibung. Jedoch stellt USDL nur ein strukturelles und abstraktes Modell der Dienstbeschreibungen vor. Die Modellierung einzelner Werte von Diensteigenschaften sind unterspezifiziert.

Der in [WHM10] vorgestellte Ansatz zur Verifikation allgemeiner Eigenschaften ist fokussiert auf die Bestimmung von Konflikten, Erreichbarkeit und Ausführbarkeit von Prozessmodellen. Obwohl die Überprüfung dieser Eigenschaften sehr häufig angewendet werden müssen, konzentriert sich unsere Arbeit auf die Verifikation von Eigenschaften, die vom Benutzer in einer Anfrage formuliert werden, wie z.B. ein gewünschtes Interaktionsmuster.

Nicht-funktionale Eigenschaften von Diensten und deren formale Repräsentation wurden oft im Kontext des Rankings von Diensten untersucht [OEH05, KP06, TRF07]. Die Modellierung nicht-funktionaler Eigenschaften in Form von Policies wurde in [PPCM08, PCP09] vorgeschlagen. In einem ausdrucksstarken Modell können Bedingungen, Maßeinheiten und Wertebereiche spezifiziert werden. Das in unserem Ansatz verwendete Modell zur Beschreibung nicht-funktionaler Eigenschaften kann leicht um diese Informationen (Bedingungen und Maßeinheiten) erweitert werden. Wertebereiche nicht-funktionaler Eigenschaften werden bereits durch die Verwendung der Dienstklassen unterstützt.
Policies beschreiben qualitativ und quantitativ messbarer Eigenschaften und

können bei der Dienstbeschreibung (wieder-) verwendet werden. Eine Effizienzsteigerung einer NFP-basierten Dienstsuche basierend auf den Policies wurde bislang noch nicht untersucht.

In unserem Dienstsuchansatz haben wir die oft verwendete Idee der Materialisierung vorausberechneter Ergebnisse zur Steigerung der Effizienz angewendet. Eine Indexstruktur, z.B. basierend auf der simuliert-durch Beziehung [San96] zwischen Prozessmodellen, wird erstellt und kann zur effizienteren Modellprüfung (model checking) ausgenutzt werden. Im Vergleich zu dem vorgestellten Ansatz erbringt die Nutzung einer solchen Indexstruktur keine Vereinfachung für den Benutzer während der Dienst- und Anfragemodellierung.

Die Klassifikation funktionaler Eigenschaften semantisch beschriebener Web Services wurde ebenfalls zur effizienten Suche ausgenutzt [Lar06, SHH07]. Da jedoch die meisten dieser Ansätze keine formale Definitionen der Klassen besitzen, ist es weder möglich, Dienste automatisch zu klassifizieren, noch ist es möglich, automatisch zu überprüfen, ob eine Klassifizierung konsistent mit den verbleibenden Dienstbeschreibung, wie z.B. den Preconditions und Effects ist. Obwohl der Ansatz in [SHH07] formal definierte Klassen (sogenannte Goals) verwendet, können Benutzer die in der Anfrage verwendeten Klassen nicht weiter mit zusätzlichen expliziten Anforderungen verfeinern. Weiterhin können Dienstbeschreibungen nicht mit Klassen annotiert werden.

Kapitel 5

Zusammenfassung und Ausblick

Das Auffinden von Diensten im Grid ist eine zentrale Funktionalität in der Diensteschicht der WisNetGrid Architektur. Dabei wurden in diesem Bericht zwei verschiedene Ansätze der Dienstsuche vorgestellt, welche komplementär verwendet werden können. Die Dienstsuche basierend auf den Metadaten der Dienstbeschreibungen ist vor allem für eine grobe und intuitive Suche nach Dienstbeschreibungen im WisNetGrid Dienstverzeichnis geeignet. Die Metainformationen der im Verzeichnis abgelegten Beschreibungen werden durch den WisNetGrid Metadatendienst verwaltet. Anforderungen einer Suchanfrage können leicht in die SPARQL Anfragesprache übersetzt werden. Der Metadatendienst identifiziert passende Dienste basierend auf den RDF Beschreibungen der Metainformationen über die Dienstbeschreibungen.

Allerdings beinhaltet der erste Suchansatz nicht die Möglichkeit die Eigenschaften des beschriebenen Dienstes einzuschränken. Die im Grid anvisierten Dienste und Workflows besitzen häufig ein komplexes Verhalten. Bisher wurden keine Ansätze zur verhaltensbasierten semantischen Dienstsuche im Grid vorgestellt. Unser dafür vorgestellter Ansatz zur Dienstsuche basierend auf semantischen Beschreibungen funktionaler und nicht-funktionaler Diensteigenschaften füllt diese Lücke und ist die Grundlage für die Komposition der Dienste. Die Dienstkomposition ist eine weitere Komponente in der Diensteschicht und wird im Bericht [Wis11] in einer ersten Version vorgestellt.

Wir haben in den Experimenten mit dem zweiten Ansatz gezeigt, dass der Einsatz komplexer Formalismen zur Beschreibung der Dienste und der Anfrage in Kombination mit der Nutzung semantischer Technologien schnell

auf Skalierbarkeitsprobleme stößt. Ausgehend von der naïven Implementierung der Dienstsuche wurde die Klassifikation der Dienste als leistungssteigernde Erweiterung vorgestellt. Dienstklassen zeichnen sich durch deren formale Definition, sowie eine vom Benutzer verständliche Bezeichnung aus. Dieser Ansatz bringt vielfältige Vorteile hervor, u.a.: (1) Effizienzsteigerung des Suchansatzes durch die Klassifizierung der Dienste vor Beantwortung der Anfrage, (2) Vereinfachung der Modellierung von Dienstbeschreibungen und Dienstanfragen durch die Wiederverwendung existierender Dienstklassen sowie (3) die Möglichkeit zur Modellierung von Unbestimmtheit in Dienstbeschreibungen. Die wiederholten Experimente mit der Nutzung der Dienstklassen offenbarten die Effizienzsteigerung. Der genaue Grad der Steigerung ist jedoch abhängig vom jeweiligen Einsatz (d.h. der Verwendung der Klassen in der Anfrage, Vernetzung und Größe der Klassenhierarchie) und ließ sich daher nur grob umreißen.

Für die Zukunft verbleibt es den Fokus verstärkt auf die Leistungsfähigkeit, Effizienz und Skalierbarkeit des Ansatzes zu setzen. Insbesondere soll untersucht werden wie Dienstanfragen so umformuliert werden können, dass die neue Anfrage von so vielen Dienstklassen wie möglich Gebrauch macht. Dadurch lässt sich die vorberechnete Dienstklassifikation zu einem maximalen Grad zur Zeit der Anfrageauswertung ausnutzen. Ein weiterer wichtiger Aspekt ist die Anpassung und Optimierung der verwendeten Klassenhierarchie. Dabei gilt es die Klassenhierarchie so vorzuhalten, dass sich eine bestmögliche Sucheffizienz für eine unbekannte Abfolge von Dienstanfragen einstellt, indem die Größe der Klassenhierarchie den Anforderungen entsprechend gewählt werden kann.

Literaturverzeichnis

[ADH+09] Agarwal, Sudhir ; Daivandy, Jason ; Harms, Patrick ; Hose, Katja ; Hünich, Denis ; Michels, Carolin ; Müller-Pfefferkorn, Ralph ; Schenkel, Ralf ; Schuller, Bernd: *D3.1.1 Architektur des Gesamtsystems.* Dezember 2009. – verfügbar unter `http://www.wisnetgrid.org/`

[Aga07] Agarwal, Sudhir: *Formal Description of Web Services for Expressive Matchmaking.* Institut AIFB, D-76128 Karlsruhe, Universität Karlsruhe (TH), Diss., 2007

[AHJM10] Agarwal, Sudhir ; Harms, Patrick ; Jäkel, René ; Michels, Carolin: *D3.2.2 Semantische Beschreibungssprache für Web-Dienste.* Oktober 2010. – verfügbar unter `http://www.wisnetgrid.org/`

[AHM+09] Agarwal, Sudhir ; Harms, Patrick ; Michels, Carolin ; Molch, Silke ; Radermacher, Eva: *D3.2.1 Anforderungen an die semantische Beschreibungssprache für Web-Dienste.* Dezember 2009. – verfügbar unter `http://www.wisnetgrid.org/`

[AHM10] Agarwal, Sudhir ; Harms, Patrick ; Michels, Carolin: *D3.2.3 Diensteverzeichnis.* Juni 2010. – verfügbar unter `http://www.wisnetgrid.org/`

[ALS09] Agarwal, S. ; Lamparter, S. ; Studer, R.: Making Web services tradable - A policy-based approach for specifying preferences on Web service properties. In: *Web Semantics: Science, Services and Agents on the World Wide Web, Special Issue on Policies* (2009)

[BG04] Brickley, Dan ; Guha, R.V.: *RDF Vocabulary Description Language 1.0: RDF Schema.* W3C Recommendation, 2004

[BM10] Brickley, Dan ; Miller, Libby: FOAF Vocabulary Specification 0.97. Version: January 2010. `http://xmlns.com/foaf/spec/20100101.html`. 2010. – Namespace document

[BS01] Bradfield, Julian ; Stirling, Colin: Modal Logics and mu-Calculi: An Introduction. In: Bergstra, J. A. (Hrsg.) ; Ponse, A. (Hrsg.) ; Smolka, Scott A. (Hrsg.): *Handbook of Process Algebra*. New York, NY, USA : Elsevier Science Inc., 2001. – ISBN 0444828303, S. 293–330

[DHH+10] Daivandy, Jason M. ; Harms, Patrick ; Hose, Katja ; Hünich, Denis ; Jäkel, René ; Metzger, Steffen ; Müller-Pfefferkorn, Ralph ; Schenkel, Ralf ; Schuller, Bernd: *D2.2.1 Spezifikation der API fur Metadatendienste*. März 2010. – verfügbar unter `http://www.wisnetgrid.org/`

[GGWW08] Gill, Tony ; Gilliland, Anne J. ; Whalen, Maureen ; Woodley, Mary ; Baca, Murtha (Hrsg.): *Introduction to Metadata*. Online Edition, Version 3.0. Los Angeles : Getty Research Insitute, 2008 `http://www.getty.edu/research/conducting_research/standards/intrometadata/`. – ISBN 978–0–89236–967–6

[goo] *Good Relations. The Web Vocabulary for E-Commerce*. `http://www.heppnetz.de/projects/goodrelations/`

[HKP+09] Hitzler, Pascal (Hrsg.) ; Krötzsch, Markus (Hrsg.) ; Parsia, Bijan (Hrsg.) ; Patel-Schneider, Peter F. (Hrsg.) ; Rudolph, Sebastian (Hrsg.): *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 27 October 2009. – Available at `http://www.w3.org/TR/owl2-primer/`

[JA10] Junghans, Martin ; Agarwal, Sudhir: Web Service Discovery Based on Unified View on Functional and Non-Functional Properties. In: *ICSC*, IEEE Computer Society, 2010

[JAS10] Junghans, Martin ; Agarwal, Sudhir ; Studer, Rudi: Towards Practical Semantic Web Service Discovery. In: Aroyo, Lora (Hrsg.) ; Antoniou, Grigoris (Hrsg.) ; Hyvönen, Eero (Hrsg.) ; Teije, Annette ten (Hrsg.) ; Stuckenschmidt, Heiner (Hrsg.) ; Cabral, Liliana (Hrsg.) ; Tudorache, Tania (Hrsg.): *ESWC (2)* Bd. 6089, Springer, 2010 (Lecture Notes in Computer Science). – ISBN 978–3–642–13488–3, S. 15–29

[Koz83] Kozen, Dexter: Results on the Propositional mu-Calculus. In: *Theor. Comput. Sci.* 27 (1983), S. 333–354

[KP06] Kritikos, Kyriakos ; Plexousakis, Dimitris: Semantic QoS Metric Matching. In: *Proceedings of the European Conference on Web Services*. Washington, DC, USA : IEEE Computer Society, 2006. – ISBN 0–7695–2737–X, 265–274

[Lar06] Lara, Rubén: Two-phased Web Service Discovery. In: *Proceedings of AI-Driven Technologies for Services-Oriented Computing Workshop at AAAI-06, Boston, USA* (2006)

[MBH+04] Martin, David ; Burstein, Mark ; Hobbs, Jerry ; Lassila, Ora ; McDermott, Drew ; McIlraith, Sheila ; Narayanan, Srini ; Paolucci, Massimo ; Parsia, Bijan ; Payne, Terry ; Sirin, Evren ; Srinivasan, Naveen ; Sycara, Katia: *OWL-S: Semantic Markup for Web Services*. `http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/`. Version: 2004

[MM04] Manola, Frank ; Miller, Eric ; W3C (Hrsg.): *RDF Primer*. `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`. Version: 2004 (W3C Recommendation)

[OEH05] O'Sullivan, J. ; Edmond, D. ; Hofstede, A.H.M. ter: Formal Description of Non-Functional Service Properties. In: *Informe Técnico FIT-TR-2005-01, Business Process Management Group, Centre for Information Technology Innovation, Queensland University of Technology* (2005)

[PCP09] Palmonari, Matteo ; Comerio, Marco ; Paoli, Flavio: Effective and Flexible NFP-Based Ranking of Web Services. In: *Proceedings of the 7th International Joint Conference on Service-Oriented Computing*. Berlin, Heidelberg : Springer-Verlag, 2009 (ICSOC-ServiceWave '09). – ISBN 978–3–642–10382–7, 546–560

[PKPS02] Paolucci, Massimo ; Kawamura, Takahiro ; Payne, Terry R. ; Sycara, Katia P.: Semantic Matching of Web Services Capabilities. In: Horrocks, Ian (Hrsg.) ; Hendler, James A. (Hrsg.) ; Horrocks, Ian (Hrsg.) ; Hendler, James A. (Hrsg.): *International Semantic Web Conference* Bd. 2342, Springer, 2002 (Lecture Notes in Computer Science), 333–347

[PPCM08] Paoli, Flavio D. ; Palmonari, Matteo ; Comerio, Marco ; Maurino, Andrea: A Meta-model for Non-functional Property Descriptions of Web Services. In: *Proceedings of the 2008 IEEE International Conference on Web Services*. Washington, DC, USA : IEEE Computer Society, 2008. – ISBN 978–0–7695–3310–0, 393–400

[PS08] Prud'hommeaux, Eric ; Seaborne, Andy: *SPARQL Query Language for RDF*. W3C Recommendation. `http://www.w3.org/TR/rdf-sparql-query/`. Version: January 2008

[San96] Sangiorgi, Davide: Bisimulation for Higher-Order Process Calculi. In: *Information and Computation* 131 (1996), S. 141–178

[SBH+05] Sure, York ; Bloehdorn, Stephan ; Haase, Peter ; Hartmann, Jens ; Oberle, Daniel: The SWRC Ontology - Semantic Web for Research Communities. In: *Proc. of the 12th Portuguese Conference on Artificial Intelligence - Progress in Artificial Intelligence (EPIA 2005)* Bd. 3803, Springer, December 2005 (LNCS), S. 218–231

[SHH07] Stollberg, Michael ; Hepp, Martin ; Hoffmann, Jörg: A Caching Mechanism for Semantic Web Service Discovery. In: *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*. Berlin, Heidelberg : Springer-Verlag, 2007 (ISWC'07/ASWC'07). – ISBN 3–540–76297–3, 978–3–540–76297–3, S. 480–493

[SS04] Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies*. Springer, 2004 (International Handbooks on Information Systems). – ISBN 3–540–40834–7

[Sti01] Stirling, Colin: *Modal and Temporal Properties of Processes*. New York, NY, USA : Springer-Verlag New York, Inc., 2001. – ISBN 0–387–98717–7

[Syc03] Sycara, Katia P.: Automated discovery, interaction and composition of Semantic Web services. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 1 (2003), Nr. 1, S. 27–46. `http://dx.doi.org/10.1016/j.websem.2003.07.002`. – DOI 10.1016/j.websem.2003.07.002. – ISSN 15708268

[TRF07] Toma, Ioan ; Roman, Dumitru ; Fensel, Dieter: On Describing and Ranking Services based on Non-Functional Properties. In: *Proceedings of the Third International Conference on Next Generation Web Services Practices*. Washington, DC, USA : IEEE Computer Society, 2007 (NWESP '07). – ISBN 0–7695–3022–2, 61–66

[w3c] *OWL-S: Semantic Markup for Web Services*. `http://www.w3.org/Submission/OWL-S/`

[Wal95] Walukiewicz, Igor: On the Completeness of Kozen's Axiomatization of the μ-Calculus. In: *Proceedings of the 10th IEEE LICS*, 1995, S. 14–24

[WHM10] Weber, Ingo ; Hoffmann, Jörg ; Mendling, Jan: Beyond Soundness: On the Verification of Semantic Business Process Models. In: *Distributed and Parallel Databases (DAPD)* 27 (2010), Juni, Nr. 3, S. 271–343. `http://dx.doi.org/10.1007/s10619-010-7060-9`. – DOI 10.1007/s10619–010–7060–9

[Wis11] WisNetGrid: *D3.3.1 Untersuchung und Vergleich von bestehenden Workflow Werkzeugen bzgl. der Anforderungen*. Juni 2011. – verfügbar unter `http://www.wisnetgrid.org/`

[WKLW98] Weibel, S. ; Kunze, J. ; Lagoze, C. ; Wolf, M.: *Dublin Core Metadata for Resource Discovery*. United States, 1998

[wsb07] Alves, Alexandre (Hrsg.) ; Arkin, Assaf (Hrsg.) ; Askary, Sid (Hrsg.) ; Barreto, Charlton (Hrsg.) ; Ben (Hrsg.) ; Curbera, Francisco (Hrsg.) ; Ford, Mark (Hrsg.) ; Goland, Yaron (Hrsg.) ; Guízar, Alejandro (Hrsg.) ; Kartha, Neelakantan (Hrsg.) ; Liu, Canyang K. (Hrsg.) ; Khalaf, Rania (Hrsg.) ; König, Dieter (Hrsg.) ; Marin, Mike (Hrsg.) ; Mehta, Vinkesh (Hrsg.) ; Thatte, Satish (Hrsg.) ; Rijn, Danny van d. (Hrsg.) ; Yendluri, Prasad (Hrsg.) ; Yiu, Alex (Hrsg.). OASIS Web Services Business Process Execution Language (WSBPEL) TC: Web Services Business Process Execution Language Version 2.0 / OASIS Web Services Business Process Execution Language (WSBPEL) TC. 2007. – Forschungsbericht