Eine SPARQL-Schnittstelle für OData-Services

Dissertation

zur Erlangung des akademischen Grades Doktor der Naturwissenschaften (Dr. rer. nat.)

Fachbereich Elektrotechnik/Informatik
Universität Kassel

eingereicht von
M. Sc. Marc Kirchhoff

Dissertation an der Universität Kassel

Gutachter: 1. Prof. Dr. Kurt Geihs
2. Prof. Dr. Alexander Mädche

Datum der Einreichung: 26.06.2014
Datum der Disputation: 04.11.2014

Inhaltsverzeichnis

Abbildungsverzeichnis ..... **v**

Tabellenverzeichnis ..... **vi**

Verzeichnis der Listings ..... **vii**

Algorithmenverzeichnis ..... **x**

Danksagung ..... **xi**

Zusammenfassung ..... **xii**

Grundlagen **1**

1 Einleitung **2**

1.1 Motivation 2

1.2 Problem 4

1.3 Lösungsansatz 6

1.4 Beiträge 8

1.5 Aufbau der Arbeit 10

2 Grundlagen **11**

2.1 Resource Description Framework (RDF) 11

2.1.1 Terse RDF Triple Language (Turtle) 15

2.2 SPARQL Protocol and RDF Query Language (SPARQL) 17

2.2.1 Sprachelemente 18

2.2.2 SPARQL-Algebra 23

2.2.3 Evaluierungssemantik 26

2.3 Open Data Protocol (OData) 32

3 Verwandte Arbeiten **40**

3.1 Semantische Beschreibung von Web Services 40

3.1.1 SAWSDL 41

3.1.2 OWL-S 42

3.1.3 WSMO 43

3.1.4 hREST und MicroWSMO 44

3.1.5 RDFa 45

3.1.6 SA-REST 46

3.1.7 ReLL 46

3.1.8 EXPRESS 47

3.1.9 Zusammenfassung 48

3.2 Service-basierte SPARQL-Endpunkte 49

3.2.1 ANGIE 49

3.2.2 SEREDASj ... 50
3.2.3 Semantic Bridge for Web Services ... 51
3.2.4 Generic semantic problem-solving platform ... 52
3.2.5 SADI + SHARE ... 53
3.2.6 Uniform Semantic Data Integration System ... 54
3.2.7 Composing Data Providing Services ... 55
3.2.8 Weitere Arbeiten ... 56
3.2.9 Zusammenfassung ... 57

Lösungsansatz ... 59

4 Semantische Beschreibung von OData-Services ... 60

4.1 Abbildung konzeptueller Datenmodelle auf RDF-Templates ... 60
4.2 Semantische Erweiterung von CSDL ... 63
4.3 Aufbau der Ressourcen-URIs ... 64
4.4 Abbildung eines Entitätstyps auf ein Ressourcen-Template ... 71
4.4.1 Typ-Angaben und konkrete Aussagen ... 71
4.4.2 Datentypen und Sprachangaben ... 72
4.5 Abbildung eines Entitätstyps auf mehrere Ressourcen-Templates ... 76
4.6 Abbildung mehrerer Eigenschaften auf ein Aussagen-Template ... 78
4.7 Abbildung auf leere Knoten ... 81
4.8 Abbildung zweier Entitätstypen auf ein Ressourcen-Template ... 82
4.9 Abbildung einzelner Navigationseigenschaften auf abstrakte Aussagen 85
4.10 Abbildung mehrerer Entitätstypen auf ein Ressourcen-Template ... 88
4.11 Abbildung mehrerer Navigationseigenschaften auf abstrakte Aussagen 91
4.12 Mehrfachabbildungen von Eigenschaften ... 93
4.13 Beschreibung von komplexen Typen ... 94
4.14 Typhierarchien ... 97
4.15 Abbildungen in Abhängigkeit der Entitätsmenge ... 99
4.16 Beschreibung von eingeschränkten Entitätsmengen ... 103
4.17 Reifikation ... 105
4.18 Abbildung auf Listen ... 107
4.19 Definitionen ... 108
4.20 Zusammenfassung ... 110

5 Abbildung von SPARQL-Anfragen auf OData-Service-Aufrufe ... 112

5.1 Ermittlung der OData-URIs auf Basis der SPARQL-Algebra ... 114
5.2 Auflösung der semantischen Annotationen ... 120
5.2.1 Bestimmung der Subjekt-Templates ... 123
5.2.2 Bestimmung der Objekt-Templates ... 130

5.2.3 Auflösung der Tripel-Templates ... 132
5.3 Ermittlung der RDF-Graph-Templates ... 134
5.3.1 Der Pfad eines Tripel-Templates ... 140
5.3.2 Formalisierung ... 145
5.4 Bestimmung der relevanten Tripel-Templates ... 150
5.4.1 Vergleich eines Tripel-Patterns mit einem Tripel-Template ... 152
5.5 Eine Evaluierungssemantik für Graph-Templates ... 161
5.5.1 Pattern-Instanz-Template-Mappings ... 162
5.5.2 Auflösung von Tripel-Patterns ... 165
5.5.3 BGP ... 166
5.5.4 Union und Join ... 168
5.5.5 Filter ... 169
5.5.6 LeftJoin ... 170
5.5.7 Graph ... 170
5.6 Erzeugung und Verschmelzung der OData-URIs ... 171
5.6.1 Überführung von Pfaden nach OData-URIs ... 174
5.6.2 Die Erzeugung von Filter-Anweisungen ... 178
5.6.3 Verschmelzen von OData-URIs ... 181

Evaluierung ... 185

6 Implementierung und Evaluierung ... 186

6.1 Northwind-Service SPARQL-Endpunkt ... 187
6.2 ERP Services ... 190

7 Zusammenfassung und Ausblick ... 193

Anhang ... 198

A. Entitätsdatenmodell des Northwind-Services ... 199
B. CSDL-Erweiterungen in XSD ... 200
C. Semantisch erweitertes EDM des Northwind-Services ... 203
D. Northwind-Service Test-Anfragen ... 208
E. Semantisch erweiterterter Sales Order Service ... 214
F. Sales Order Service Test-Anfragen ... 218
G. CD ... 223

Publikationen ... 224

Literaturverzeichnis ... 225

Erklärung ... 240

Abbildungsverzeichnis

Abbildung 1: Ein Geschäftsobjektgraph. 2
Abbildung 2: SPARQL-OData-Layer. 7
Abbildung 3: Beziehungen zwischen Hersteller und Betreiber [108]. 9
Abbildung 4: RDF-Graph. 13
Abbildung 5: Leerer Knoten (angelehnt an das Beispiel aus [140]). 13
Abbildung 6: Eine ungeordnete Liste (*rdf:Bag*) in RDF. 14
Abbildung 7: Eine geschlossene Liste (*Collection*) in RDF. 14
Abbildung 8: Reifikation 15
Abbildung 9: Abbildung eines EDM auf ein RDF-Graph-Template durch die semantische Beschreibung. 61
Abbildung 10: Die semantische Beschreibung eines Entitätstyps. 61
Abbildung 11: Abbildung einer Entität auf einen RDF-Graphen. 62
Abbildung 12: Abbildung eines Entitätstyps auf mehrere Ressourcen-Templates. 76
Abbildung 13: Zwei Eigenschaften werden auf eine Aussage abgebildet. 79
Abbildung 14: Abbildung eines Entitätstyps auf das Template einer Ressource und eines leeren Knotens. 81
Abbildung 15: Abbildung zweier Entitätstypen auf ein Ressourcen-Template. 83
Abbildung 16: Abbildung einer Navigationseigenschaft auf zwei abstrakte Aussagen. 85
Abbildung 17: Abbildung einer Navigationseigenschaft mit mehreren Ressourcen-Templates. 87
Abbildung 18: Abbildung mehrerer Entitätstypen auf ein Ressourcen-Template. 88
Abbildung 19: Mehrere Verbindungen zwischen zwei Entitätstypen. 89
Abbildung 20: Abbildung mehrerer Navigationseigenschaften auf abstrakte Aussagen. 91
Abbildung 21: Unterschiedliche Abbildungen in Abhängigkeit der Navigationseigenschaften. 93
Abbildung 22: Abbildung eines komplexen Typs auf ein Ressourcen-Template. 95
Abbildung 23: Abbildung zweier Entitätstypen einer Typhierarchie auf zwei Ressourcen-Templates. 98
Abbildung 24: Mehrere Entitätsmengen des gleichen Entitätstyps. 100
Abbildung 25: Abbildung eines Entitätstyps in Abhängigkeit der Entitätsmengen. 100
Abbildung 26: Abbildung einer Typhierarchie in Abhängigkeit der Entitätsmengen. 101
Abbildung 27: Abbildung auf ein Ressourcen- und ein Reifikations-Template. 106
Abbildung 28: Abbildung mehrerer Eigenschaften auf eine offene Liste. 107
Abbildung 29: Abbildung auf eine geschlossene Liste. 108
Abbildung 30: Vorgehensweise bei der Überführung von SPARQL-Anfragen nach OData-URIs. 114
Abbildung 31: Ermittlung der Graph-Templates in Abhängigkeit der relevanten Ressourcen-Variablen. 139

Tabellenverzeichnis

Tabelle 1: SPARQL-Ergebnistabelle. 19
Tabelle 2: Verschiedene in SPARQL unterstützte Filter-Funktionen. 21
Tabelle 3: Die von SPARQL unterstützten Modifikatoren. 22
Tabelle 4: Die Operatoren der SPARQL-Algebra (entnommen aus [155]). 26
Tabelle 5: Abbildung der EDM-Datentypen auf die XSD-Datentypen. 74
Tabelle 6: Abbildung von XPath-/XQuery-Funktionen und Operatoren
auf OData-Funktionen und Operatoren. 179
Tabelle 7: Ergebnisse des Performance-Tests für den Northwind-Service in ms. 188
Tabelle 8: Ergebnisse des Performance-Tests für den Sales Order Service in ms. 191

Verzeichnis der Listings

Listing 1: RDF-Tripel. 12
Listing 2: Die Typangabe mit *rdf:type*. 12
Listing 3: Ein RDF-Graph in Turtle. 16
Listing 4: Abkürzende Schreibweise für Namensräume in Turtle. 16
Listing 5: Die abgekürzte Schreibweise für mehrere Aussagen über dasselbe Subjekt in Turtle. 16
Listing 6: Mehrere Aussagen mit gleichem Subjekt und Prädikat in Turtle. 17
Listing 7: Leerer Knoten in Turtle. 17
Listing 8: Abgekürzte Schreibweise für leere Knoten in Turtle. 17
Listing 9: Darstellung geschlossener Listen mit Turtle. 17
Listing 10: Eine SPARQL-Anfrage (angelehnt an die Anfrage aus [104]). 18
Listing 11: RDF-Graph. 18
Listing 12: Eine SPARQL-Anfrage mit dem *UNION*-Operator (angelehnt an die Anfrage aus [104]). 19
Listing 13: Die Angabe eines optionalen Graph-Musters in SPARQL (angelehnt an die Anfrage aus [104]). 20
Listing 14: Die *FILTER*-Anweisung in SPARQL. 20
Listing 15: Die Verwendung des *CONSTRUCT*-Operators in SPARQL. 21
Listing 16: Die Verwendung des *ASK*-Operators in SPARQL. 21
Listing 17: Der SPARQL-*DESCRIBE*-Operator. 22
Listing 18: Die Sortierung in SPARQL mit dem *ORDER BY*-Operator. 22
Listing 19: Benannte Graphen in SPARQL. 23
Listing 20: Eine SPARQL-Anfrage, welche in einen Ausdruck der SPARQL-Algebra überführt werden soll. 24
Listing 21: Der SPARQL-Algebra-Ausdruck für die SPARQL-Anfrage aus Listing 20. 24
Listing 22: Ein RDF-Datensatz. 27
Listing 23: Ein Atom Feed Dokument. 34
Listing 24: Ein Service-Dokument. 35
Listing 25: Ausschnitt aus einem Service-Metadaten-Dokument. 36
Listing 26: Eine URI zur Abfrage des Herstellernamens eines bestimmten Equipments. 37
Listing 27: Die OData *filter*-Anweisung. 38
Listing 28: Angabe des Hosts, des Typs und der URI-Struktur für den *Order*-Entitätstyp. 71
Listing 29: Ein Ressourcen-Template auf Basis des *Order*-Entitätstyps. 71
Listing 30: Beschreibung der Abbildung einzelner Eigenschaften auf konkrete Aussagen. 72
Listing 31: Aus einer Entität erzeugter RDF-Graph mit typisierten Literalen. 74
Listing 32: Spezifikation eines Typs außerhalb des XSD-Namensraums. 75
Listing 33: Literal mit einem Typ außerhalb des XSD-Namensraums. 75
Listing 34: Spezifikation der Sprache. 75
Listing 35: Literal mit Sprachangabe. 76
Listing 36: Definition von Ressourcen-Variablen. 77

Listing 37: Semantische Erweiterung der Eigenschaften bei der Abbildung auf mehrere Ressourcen-Templates. 78
Listing 38: Ein Referenzmechanismus für Eigenschaften. 79
Listing 39: Berechnung eines Objekts unter Verwendung von XQuery- und XPath-Funktionen. 80
Listing 40: Die Beschreibung leerer Knoten. 82
Listing 41: Abbildung mehrerer Entitätstypen durch explizite Angabe der Variable. 83
Listing 42: Abbildung mehrerer Entitätstypen durch explizite Kennzeichnung der Navigationseigenschaft. 84
Listing 43: Abbildung einer Navigationseigenschaft auf abstrakte Aussagen über die Angabe der Variablen. 86
Listing 44: Abbildung einer Navigationseigenschaft auf abstrakte Aussagen durch explizite Kennzeichnung. 86
Listing 45: Abbildung einer Navigationseigenschaft mit Angabe der Ziel-Ressource. 87
Listing 46: Abbildung einer Navigationseigenschaft mit Angabe der Subjekt-Ressourcen-Variable. 88
Listing 47: Abbildung über mehrerer Navigationseigenschaften durch explizite Angabe der Variable. 89
Listing 48: Abbildung über mehrerer Navigationseigenschaft durch explizite Kennzeichnung. 90
Listing 49: Abbildung mehrerer Navigationseigenschaften auf abstrakte Aussagen ohne Kennzeichnung der Navigationseigenschaft. 91
Listing 50: Abbildung mehrere Navigationseigenschaften auf eine abstrakte Aussage mittels expliziter Kennzeichnung. 92
Listing 51: Abbildung einer Eigenschaft auf mehrere Aussagen. 94
Listing 52: Beschreibung eines komplexen Typs. 95
Listing 53: Semantische Beschreibung einer Typ-Hierarchie. 99
Listing 54: Semantische Beschreibung zur Abbildung eines Entitätstyps in Abhängigkeit der Entitätsmengen. 101
Listing 55: Semantische Beschreibung zur Abbildung einer Typhierarchie in Abhängigkeit der Entitätsmengen. 102
Listing 56: Beschreibungen von Entitätsmengen mittels Filterausdrücken. 104
Listing 57: Beschreibung der Einschränkung einer Entitätsmenge mit einem SPARQL-Operator. 104
Listing 58: Beschreibung der Reifikation. 106
Listing 59: Beschreibung der Abbildung mehrerer Eigenschaften auf eine offene Liste. 107
Listing 60: Eine Beispiel-SPARQL-Abfrage. 112
Listing 61: Eine SPARQL-Anfrage. 116
Listing 62: Ausschnitt aus dem RDF-Graph-Template des Northwind-Services. 117
Listing 63: Ein Ausdruck der SPARQL-Algebra. 117
Listing 64: Semantische Annotationen in vereinfachter Schreibweise. 120
Listing 65: Semantische Erweiterung einer Typ-Hierarchie. 124
Listing 66: Einbeziehung eines komplexen Typs in die Abbildung. 129
Listing 67: Auflösung einer geschlossenen Liste. 133

Listing 68: Ausschnitt aus einem semantisch erweiterten CSDL-Dokument. 135
Listing 69: Abbildung eines Entitätstyps und eines komplexen Typs auf ein Ressourcen-Template. 142
Listing 70: Ein einfaches Graph-Muster mit leeren Knoten. 151
Listing 71: Ein RDF-Graph. 152

Algorithmenverzeichnis

Algorithmus 1: $STr = sT\_P(p)$ 126
Algorithmus 2: $STr = sT\_ET(e)$ 127
Algorithmus 3: $STr = upRV(t)$ 128
Algorithmus 4: $STr = sT\_CT(c)$ 129
Algorithmus 5: $OTr = oT\_P(p)$ 131
Algorithmus 6: $OTr = oT\_N(n)$ 132
Algorithmus 7: $TT = generateTTs(ST, pt, OT)$ 133
Algorithmus 8: $R = gtRV(t)$ 137
Algorithmus 9: $GT = gT(es)$ 146
Algorithmus 10: $GT = gt(t, R)$ 147
Algorithmus 11: $GT = rTTs(M, R)$ 147
Algorithmus 12: $PA = cP(tt)$ 148
Algorithmus 13: $tt = cTT(tt, pa)$ 149
Algorithmus 14: $m = matchPT(p, t)$ 157
Algorithmus 15: $m = matchTT(t1, t2)$ 160
Algorithmus 16: $m = matchTP\_TT(tp, tt)$ 160
Algorithmus 17: $ou = createODataURI(tt, tp)$ 174
Algorithmus 18: $ou = calcOU(pa)$ 177
Algorithmus 19: $F = calcFilter(tt, tp)$ 181
Algorithmus 20: $ou = mergeURIs(ou1, ou2)$ 183
Algorithmus 21: $OU = merge(OU1, OU2)$ 184

Danksagung

Diese Arbeit ist im Rahmen des durch das BMBF geförderten RES-COM-Projekts bei der SAP AG in Dresden in Kooperation mit dem Fachgebiet Verteilte Systeme an der Universität Kassel entstanden. Ich danke der SAP AG, die mir über die Finanzierung einer Doktorandenstelle diese Doktorarbeit ermöglicht hat. Insbesondere möchte ich Herrn Dr. Jochen Rode und Frau Dr. Barbara Schennerlein für ihre freundschaftliche Zusammenarbeit danken. Trotz knapper Ressourcen haben sie mir stets genügend Zeit für die Durchführung dieser Arbeit eingeräumt.

Herrn Prof. Dr. Geihs möchte ich für die Betreuung dieser Arbeit danken. Ohne seine zahlreichen Anregungen und die konstruktive Kritik, die mich bereits seit Beginn meines Studiums begleitet haben, wäre die Erstellung dieser Arbeit nicht möglich gewesen. Herrn Prof. Dr. Mädche danke ich für seine hilfreichen Hinweise, und dass er das Zweitgutachten mit großer Begeisterung für das Thema übernommen hat.

Des Weiteren möchte ich meinen Kollegen bei SAP danken, die durch ihre fortwährende kollegiale Zusammenarbeit und Unterstützung einen motivierenden Beitrag zur Erstellung dieser Dissertation geleistet haben. Mein Dank gilt insbesondere: Stefan Hesse, Alexander Claus, Dr. Sven Horn, Dr. Daniela Wünsch, Dr. Baris Sertkaya, Dr. Bart-Jan van Putten, Thomas Janke, Dr. Volodymyr Vasyutynskyy, Christian Hengstler, Dr. Matthias Heinrich und Dr. Steffen Göbel. Mein besonderer Dank gilt außerdem Dr. Markus Küstner, der mir im Laufe der letzten Jahre im Rahmen zahlreicher Projekte sowohl fachlich als auch methodisch stets eine große Hilfe war. Darüber hinaus möchte ich den Studenten danken, die mich die letzten Jahre unterstützt haben.

Nicht zuletzt möchte ich meinen Eltern für die Korrektur dieser Arbeit und ihre Unterstützung danken.

Zusammenfassung

Enterprise-Resource-Planning-Systeme (ERP-Systeme) bilden für die meisten mittleren und großen Unternehmen einen essentiellen Bestandteil ihrer IT-Landschaft zur Verwaltung von Geschäftsdaten und Geschäftsprozessen. Geschäftsdaten werden in ERP-Systemen in Form von Geschäftsobjekten abgebildet. Ein Geschäftsobjekt kann mehrere Attribute enthalten und über Assoziationen zu anderen Geschäftsobjekten einen Geschäftsobjektgraphen aufspannen. Existierende Schnittstellen ermöglichen die Abfrage von Geschäftsobjekten, insbesondere mit Hinblick auf deren Attribute. Die Abfrage mit Bezug auf ihre Position innerhalb des Geschäftsobjektgraphen ist jedoch über diese Schnittstellen häufig nur sehr schwierig zu realisieren. Zur Vereinfachung solcher Anfragen können semantische Technologien, wie RDF und die graphbasierte Abfragesprache SPARQL, verwendet werden. SPARQL ermöglicht eine wesentlich kompaktere und intuitivere Formulierung von Anfragen gegen Geschäftsobjektgraphen, als es mittels der existierenden Schnittstellen möglich ist. Die Motivation für diese Arbeit ist die Vereinfachung bestimmter Anfragen gegen das im Rahmen dieser Arbeit betrachtete SAP ERP-System unter Verwendung von SPARQL.

Zur Speicherung von Geschäftsobjekten kommen in ERP-Systemen typischerweise relationale Datenbanken zum Einsatz. Die Bereitstellung von SPARQL-Endpunkten auf Basis von relationalen Datenbanken ist ein seit längerem untersuchtes Gebiet. Es existieren verschiedene Ansätze und Tools, welche die Anfrage mittels SPARQL erlauben. Aufgrund der Komplexität, der Größe und der Änderungshäufigkeit des ERP-Datenbankschemas können solche Ansätze, die direkt auf dem Datenbankschema aufsetzen, nicht verwendet werden. Ein praktikablerer Ansatz besteht darin, den SPARQL-Endpunkt auf Basis existierender Schnittstellen zu realisieren. Diese sind weniger komplex als das Datenbankschema, da sie die direkte Abfrage von Geschäftsobjekten ermöglichen. Dadurch wird die Definition des Mappings erheblich vereinfacht. Das ERP-System bietet mehrere Schnittstellen an, die sich hinsichtlich des Aufbaus, der Zielsetzung und der verwendeten Technologie unterscheiden. Unter anderem wird eine auf OData basierende Schnittstelle zur Verfügung gestellt. OData ist ein REST-basiertes Protokoll zur Abfrage und Manipulation von Daten. Von den bereitgestellten Schnittstellen weist das OData-Interface gegenüber den anderen Schnittstellen verschiedene Vorteile bei Realisierung eines SPARQL-Endpunktes auf. Es definiert eine Abfragesprache und einen Link-Adressierungsmechanismus, mit dem die zur Beantwortung einer Anfrage benötigten Service-Aufrufe und die zu übertragende Datenmenge erheblich reduziert werden können. Das Ziel dieser Arbeit besteht in der Entwicklung eines Verfahrens zur Realisierung eines SPARQL-Endpunktes auf Basis von OData-Services.

Dazu wird zunächst eine Architektur vorgestellt, die als Grundlage für die Implementierung eines entsprechenden Systems dienen kann. Ausgehend von dieser Architektur, werden die durch den aktuellen Forschungsstand noch nicht abgedeckten Bereiche ermittelt. Nach bestem Wissen ist diese Arbeit die erste, welche die Abfrage von OData-Schnittstellen mittels SPARQL untersucht. Dabei wird als Teil dieser Arbeit ein neuartiges Konzept zur semantischen Beschreibung von OData-Services vorgestellt. Dieses ermöglicht die Definition von Abbildungen der von den Services bereitgestellten Daten auf RDF-Graphen. Aufbauend auf den Konzepten zur semantischen Beschreibung wird eine

Evaluierungssemantik erarbeitet, welche die Auflösung von Ausdrücken der SPARQL-Algebra gegen semantisch annotierte OData-Services definiert. Dabei werden die Daten aller OData-Services ermittelt, die zur vollständigen Abarbeitung einer Anfrage benötigt werden. Zur Abfrage der relevanten Daten wurden Konzepte zur Erzeugung der entsprechenden OData-URIs entwickelt. Das vorgestellte Verfahren wurde prototypisch implementiert und anhand zweier Anwendungsfälle für die im betrachteten Szenario maßgeblichen Servicemengen evaluiert.

Mit den vorgestellten Konzepten besteht nicht nur die Möglichkeit, einen SPARQL-Endpunkt für ein ERP-System zu realisieren, vielmehr kann jede Datenquelle, die eine OData-Schnittstelle anbietet, mittels SPARQL angefragt werden. Dadurch werden große Datenmengen, die bisher für die Verarbeitung mittels semantischer Technologien nicht zugänglich waren, für die Integration mit dem Semantic Web verfügbar gemacht. Insbesondere können auch Datenquellen, deren Integration miteinander bisher nicht oder nur schwierig möglich war, über Systeme zur föderierten Abfrage miteinander integriert werden.

Teil I

Grundlagen

1 Einleitung

1.1 Motivation

Die meisten mittleren und großen Unternehmen verwenden zur Verwaltung ihrer Geschäftsdaten und Geschäftsprozesse ERP-Systeme (*Enterprise-Resource-Planning*). Geschäftsdaten, wie z.B. Kunden, Bestellungen, Aufträge usw., werden in ERP-Systemen in Form von Geschäftsobjekten (*business objects*) repräsentiert. Geschäftsobjekte enthalten eine Menge von Attributen (*Name*, *Beschreibung* usw.) und Assoziationen zu anderen Geschäftsobjekten. Mehrere Geschäftsobjekte können durch gegenseitige Assoziation einen Graphen aufspannen (*Geschäftsobjektgraph*).

Abbildung 1 zeigt einen vereinfachten Geschäftsobjekt-Graph, der in einem ERP-System im Rahmen zweier Kundenaufträge (*Sales Orders*) erstellt wurde [105]. Sobald ein Kunde eine Bestellung aufgibt, wird im ERP-System ein *Sales Order* Geschäftsobjekt angelegt (z.B. *Sales Order 1A*). Das *Sales Order* Geschäftsobjekt enthält eine Relation auf ein Geschäftsobjekt vom Typ Fertigungsauftrag (*Production Order 1A*), welches alle Informationen enthält, die notwendig sind, um die bestellten Güter herzustellen. Die zur Produktion benötigten Materialien werden durch mehrere, mit dem Fertigungsauftrag verbundene, *Material* Geschäftsobjekte (*Material 1A, 1B, 1C*) repräsentiert. Da die Materialien 1A und 1C nicht mehr vorrätig sind, müssen sie zunächst bestellt werden. Die entsprechenden Kaufaufträge werden durch Geschäftsobjekte vom Typ *Bestellung* (*Purchase Order 1A, 2A*) abgebildet.

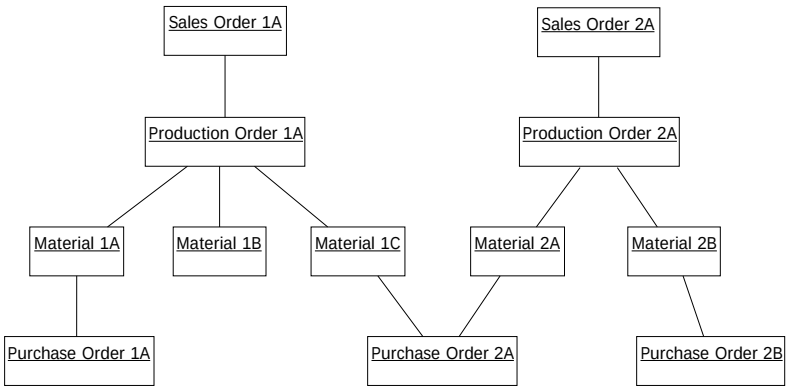


Abbildung 1: Ein Geschäftsobjektgraph.

Wenn sich die Erfüllung einer Bestellung (z.B. *Purchase Order 2A*) und damit die Lieferung der entsprechenden Materialen (*Material 1C, 2A*) durch einen externen Lieferanten verzögert, dann verzögert sich auch der Start der Produktionsprozesse, welche die entsprechenden Materialien benötigen. Die Herausforderung besteht darin, für eine gegebene Bestellung (*Purchase Order*) alle Aufträge (*Sales Orders*) zu ermitteln, die im Falle einer Verzögerung betroffen wären.

SAP ERP ist das am weitesten am Markt verbreitete ERP-System [2]. Diese Arbeit fokussiert sich daher auf das ERP-System von SAP. Soweit nicht anders erwähnt, ist im Folgenden mit ERP-System immer das ERP-System von SAP gemeint. Das ERP-System stellt verschiedene Schnittstellen zur Abfrage von Geschäftsobjekten zur Verfügung. Dazu gehören:

- **Business Application Programming Interface (BAPI):** Proprietäre, entfernt-aufrufbare Funktionsmodule.
- **Enterprise Services (ES):** Eine SOAP-/WSDL-basierte Web Service Schnittstelle.
- **NetWeaver Gateway:** Eine Infrastruktur zur Erstellung und Bereitstellung von OData-basierten REST-Services.

SAP NetWeaver Gateway bietet standardmäßig keine Services[1] an. Stattdessen ermöglicht Gateway die Erstellung anwendungsspezifischer Services. Jede Abfrage von Geschäftsobjekten setzt somit die Erstellung der entsprechenden Services voraus. Die BAPI und die Enterprise Services bieten eine nach Geschäftsobjekt-Typen gruppierte Menge von Funktionen bzw. Services zur Abfrage von Geschäftsobjekten an. Der Aufbau der Schnittstellen ermöglicht die einfache Abfrage von einzelnen Geschäftsobjekten nach ihren Attributen. Abfragen von Geschäftsobjekten bzgl. deren Relationen zu anderen Geschäftsobjekten (wie in Abbildung 1 dargestellt) sind hingegen nur sehr schwierig zu realisieren. Ein Grund dafür besteht darin, dass viele der bereitgestellten Funktionen und Services nicht die Geschäftsobjekt-übergreifende Abfrage erlauben. Um die Abfrage von Geschäftsobjekten, insbesondere hinsichtlich ihrer Anordnung im Geschäftsobjektgraph, zu vereinfachen, können semantische Technologien wie SPARQL verwendet werden. SPARQL ist eine graphbasierte Abfragesprache für RDF [155]. Die Verwendung von SPARQL zur Abfrage von ERP-Systemen hat gegenüber den beschriebenen Schnittstellen folgende Vorteile:

- Anfragen können wesentlich kompakter und intuitiver formuliert werden.
- Die Relationen zwischen den Geschäftsobjekten müssen nicht *erraten* werden, da sie durch eine Ontologie formalisiert werden können.
- Da SPARQL auf RDF basiert, können auf RDF aufbauende Techniken, wie z.B. Inferenzmechanismen, mit in die Abfrageverarbeitung einbezogen werden.

[1] Die Begriffe *Service* und *Service-Interface* bzw. *Service-Schnittstelle* werden im Folgenden entsprechend den Definitionen in [124] verwendet.

1.2 Problem

Im Allgemeinen existieren zwei Ansätze, um einen SPARQL-Endpunkt für eine bestehende Datenquelle bereitzustellen [164]:

- **ETL (*Extract Transform Load*):** Die Daten werden aus der ursprünglichen Datenquelle geladen, nach RDF überführt und in einem Triple Store abgespeichert. Der SPARQL-Prozessor arbeitet auf Basis der im Triple Store abgespeicherten RDF-Daten. Für zahlreiche Datenformate existieren Werkzeuge zur Überführung nach RDF.
- **Dynamisches Mapping:** Der SPARQL-Prozessor arbeitet direkt auf der Original-Datenquelle. Es werden nur die Daten abgefragt, die für die Beantwortung einer vorliegenden SPARQL-Anfrage notwendig sind.

Die Replikation der Daten, wie sie bei dem ETL-Ansatz verfolgt wird, ist aufgrund der Menge und Änderungshäufigkeit der typischerweise in ERP-Systemen vorhandenen Daten kein praktikabler Ansatz. Stattdessen ist ein dynamischer Ansatz erforderlich, bei dem die Daten in Abhängigkeit der vorliegenden SPARQL-Anfrage dynamisch aus dem ERP-System abgefragt werden. In SAP ERP-Systemen werden relationale Datenbanken zur Speicherung der Geschäftsobjekte verwendet. Es existieren verschiedene Ansätze und Tools um SPARQL-Endpunkte mittels dynamischem Mappings für relationale Datenbanken bereitzustellen (z.B. *Virtuoso* [76] oder *D2RQ* [18]). Grundsätzlich können zwei Ansätze zur Abbildung des Datenbankschemas auf die des SPARQL-Endpunktes zugrundeliegende Ontologie unterschieden werden [168]:

- **Direktes Mapping:** Die Ontologie wird automatisch auf Basis des Datenbankschemas generiert [167, 168, 169]. Dabei werden aus den Tabellen und Spalten automatisch die Konzepte und Eigenschaften der Ontologie erzeugt.
- **Mapping-Sprache:** Mittels einer Mapping-Sprache wird manuell ein Mapping von dem Datenbankschema auf die Ontologie erstellt [70, 98].

Weder der erste noch der zweite Ansatz können für relationale Datenbanken von ERP-Systemen verwendet werden. Die zu erstellende Ontologie, welche die Grundlage für den SPARQL-Endpunkt bildet, muss die Geschäftsobjekte, deren Attribute und die Relationen zu anderen Geschäftsobjekten modellieren. Geschäftsobjekt-Typen haben in der Regel keine direkte Abbildung im Datenbankschema des ERP-Systems. Stattdessen sind sie häufig über mehrere Tabellen verteilt. Diese Tabellen und ihre Spalten, welche den Attributen der Geschäftsobjekte entsprechen, haben oftmals nicht-intuitive Bezeichnungen. Der direkte Mapping-Ansatz würde folglich in einer sehr komplexen Ontologie mit mehreren Konzepten für die meisten Geschäftsobjekt-Typen resultieren. Eine solche Ontologie wäre für die Anfrageformulierung ungeeignet. Obwohl es theoretisch möglich ist, manuell ein Mapping

des Datenbankschemas auf eine existierende Ontologie zu erstellen, hätte diese Vorgehensweise ebenfalls verschiedene Nachteile:

- Das ERP-Datenbankschema besteht aus mehreren tausend Tabellen, die häufig schwierig zu verstehen sind. Der Aufwand für die manuelle Erstellung eines Mappings wäre daher erheblich.
- Viele Geschäftsobjekt-Typen haben keine direkte Abbildung im Datenbankschema. Dadurch wird der Aufwand zur Definition des Mappings weiter erhöht.
- Es ist nicht ungewöhnlich, dass sich das Datenbankschema des ERPs in Abhängigkeit der Version ändert, d.h. das Datenbankschema eines ERP-Systems einer bestimmten Version kann anders aufgebaut sein als das Datenbankschema der vorherigen Version. Ein einmal definiertes Mapping wäre daher nur für eine bestimmte Version gültig.

Dementsprechend können existierende Ansätze, welche direkt auf relationalen Datenbanken aufsetzen, für ERP-Systeme nicht verwendet werden. Ein besserer Ansatz besteht darin, bestehende ERP-Schnittstellen, welche die direkte Abfrage von Geschäftsobjekten ermöglichen, als Datenquelle für den SPARQL-Endpunkt zu verwenden. Diese Vorgehensweise hat gegenüber der Verwendung der relationalen Datenbank als Datenquelle folgende Vorteile:

- Die Schnittstellen sind weniger komplex als das Datenbankschema. Des Weiteren sind sie gut dokumentiert. Dadurch wird die Definition des Mappings vereinfacht.
- Die Funktionen und Services der Schnittstellen sind den Geschäftsobjekt-Typen zugeordnet. Sie ermöglichen die direkte Abfrage der Geschäftsobjekte. Damit wird das Mapping auf die entsprechenden Konzepte der Ontologie wesentlich vereinfacht.
- Für existierende Funktionen bzw. Services ist Abwärtskompatibilität für einen längeren Zeitraum garantiert. Ein Mapping, das in Bezug auf eine bestimmte ERP Version definiert wurde, ist somit auch für zukünftige Versionen gültig. Funktionen oder Services, die für bestimmte Anwendungsfälle erstellt wurden und in das Mapping mit einbezogen wurden, unterliegen üblicherweise nicht der Versionierung durch SAP. Die entsprechenden Mapping-Definitionen sind daher ebenfalls für zukünftige Versionen gültig.
- Neue Funktionen und Services, welche im Rahmen einer neuen Version veröffentlicht wurden, können dem SPARQL-Endpunkt zur Verfügung gestellt werden, ohne dass die bereits bestehende Mapping-Definition geändert werden muss.

Von den bereits in Abschnitt 1.1 erwähnten ERP-Schnittstellen (BAPI, ES, Gateway) hat SAP NetWeaver Gateway verschiedene Vorteile bei der Verwendung als Datenquelle für einen SPARQL-Endpunkt. Wie bereits erwähnt, ist NetWeaver Gateway eine Komponente, welche die Bereitstellung und Konsumierung von Geschäftsdaten aus SAP Backend Systemen, wie

dem SAP ERP, ermöglicht. Gateway basiert auf dem OData Protokoll [137], welches die Erstellung von REST-basierten Services [81, 82] ermöglicht. Gateway hat gegenüber der BAPI und den Enterprise Services bei der Verwendung als Abfragemechanismus für einen SPARQL-Endpunkt folgende Vorteile:

- OData definiert eine Abfragesprache, welche die serverseitige Filterung der Daten ermöglicht. Dadurch kann die zu übertragende Datenmenge wesentlich reduziert werden.
- Neben der OData-Schnittstelle für das SAP ERP-System sind entsprechende Schnittstellen auf für andere Systeme geplant bzw. existieren bereits, wie z.B. CRM, SRM, Business Objects, HANA usw. Die, im Rahmen der Arbeit, zu entwickelnden Konzepte können daher wiederverwendet werden, um SPARQL-Endpunkte auch für andere Systeme bereitzustellen.
- Der Link-Adressierungsmechanismus von OData ermöglicht eine wesentliche Reduzierung der notwendigen Server-Anfragen bei der Abfrage von mehreren in Relation stehenden Geschäftsobjekten.

Diese Arbeit adressiert die Herausforderung der Erstellung eines SPARQL-Endpunktes auf Basis gegebener OData-Services.

1.3 Lösungsansatz

In diesem Abschnitt wird der in dieser Arbeit realisierte Lösungsansatz vorgestellt [107]. Abbildung 2 zeigt die Architektur des Systems zur Realisierung eines SPARQL-Endpunktes auf Basis von OData-Services. Zur Darstellung der Architektur wurde TAM (*Technical Architecture Modeling*) [1] verwendet. Die Architektur ist in Form eines Block-Diagramms dargestellt. Wobei die Rechtecke aktive Systeme repräsentieren und die passiven Systeme bzw. die Datenspeicher durch Rechtecke mit zwei abgerundeten Kanten abgebildet werden. Die durch einen Kreis mit angehängtem *R* unterbrochenen Verbindungen markieren dabei Anfrage/Antwort-Kanäle. Pfeile von bzw. zu einem Datenspeicher kennzeichnen Lese- bzw. Schreibzugriffe.

Die semantischen Service-Beschreibungen aller am System registrierten OData-Services werden in der *Service Registry* gespeichert. Anfragen in Form von SPARQL werden vom Client an die *Query Engine* gestellt. Die Query Engine analysiert die SPARQL-Anfrage und evaluiert sie gegen die in der Service Registry gespeicherten semantischen Service-Beschreibungen. Wenn die von einem Service bereitgestellten Daten für die Abarbeitung der Anfrage relevant sind, erzeugt die Query Engine die entsprechenden URIs zur Abfrage dieser Daten. Die erzeugten URIs werden von der Query Engine an die *Execution Engine* weitergeleitet, welche die adressierten Daten von den OData-Services abruft. Die von den OData-Services zurückgelieferten Daten in XML werden von dem *RDF Adapter* nach RDF überführt und in einem *Triple Store* gespeichert. Ein *Reasoner* kann auf Basis gegebener Inferenzregeln zusätzliche Tripel erzeugen. Die eigentliche, vom Client gesendete, Anfrage

wird von einem *SPARQL Prozessor* auf dem im Triple Store gespeicherten RDF-Datensatz ausgeführt. Das Ergebnis wird an die Query Engine übermittelt, welche sie an den Client zurückgibt.

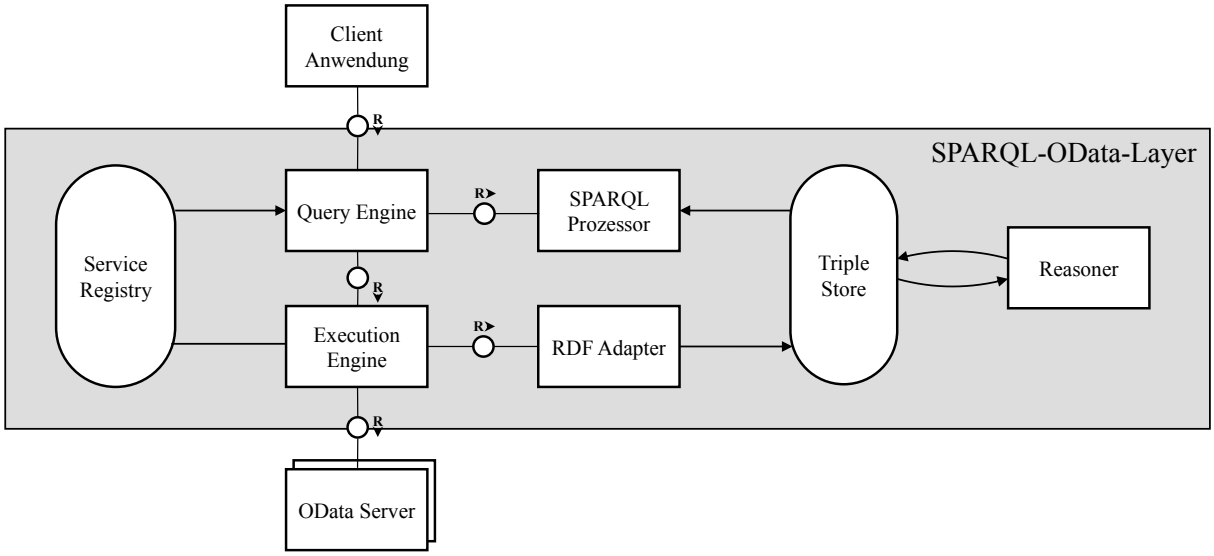


Abbildung 2: SPARQL-OData-Layer.

In Bezug auf das semantische Reasoning existieren zahlreiche Forschungsarbeiten und verschiedene Implementierungen, die unterschiedliche Komplexitätsstufen abdecken (z.B. *Pellet* [21], *KAON2* [19] und *HermiT* [10]). Das gleiche gilt für SPARQL-Prozessoren (z.B. *ARQ* [17], *Sesame* [23] und *Ontobroker* [20]), Triple Stores (z.B. *AllegroGraph* [13], *Stardog* [11] und *Jena TDB* [12]) und RDF-Adapter (z.B. *XML2RDF* [15], *Sponger* [16] und *Krextor* [14]). Die Anfrage von OData-Services kann mittels eines einfachen HTTP-Clients erfolgen. Da OData-Services entweder XML oder JSON zurückgeben, kann das Parsen der Antworten ebenfalls mit bereits existierenden Bibliotheken realisiert werden.

Soweit bekannt, existieren keine Ansätze zur semantischen Beschreibung, die speziell mit Hinblick auf OData-Services entwickelt wurden. Des Weiteren existieren keine Verfahren zur Auflösung von SPARQL-Anfragen gegen OData-Services. Daher erfolgt im Rahmen dieser Arbeit eine Fokussierung auf die semantische Beschreibung von OData-Services und die Realisierung der Query Engine.

1.4 Beiträge

Der in dieser Arbeit vorgestellte Ansatz ist das erste Verfahren zur Realisierung eines SPARQL-Endpunktes auf Basis von OData-Services. Als Teil dieses Ansatzes wurden folgende, neuartige Konzepte erarbeitet, die den aktuellen Forschungsstand erweitern:

1. **Semantische Beschreibung von OData-Services:** Es wurde eine Erweiterung der Conceptual Schema Definition Language (CSDL) [138] spezifiziert, welche die Definition von Abbildungen des einem OData-Service zugrundeliegenden Entitätsdatenmodells auf RDF-Graph-Templates ermöglicht. Die RDF-Graph-Templates beschreiben die Struktur und den Aufbau der RDF-Graphen, die aus den vom OData-Service zurückgegebenen Daten erzeugt werden können.
2. **Evaluierungssemantik:** Auf Basis der durch die SPARQL-Spezifikation definierten Evaluierungssemantik [155] wurde eine Evaluierungssemantik mit Hinblick auf RDF-Graph-Templates definiert. Diese ermöglicht die Auflösung von SPARQL-Anfragen gegen die durch die semantischen Beschreibungen der OData-Services gegebenen RDF-Graph-Templates.
3. **Erzeugung von OData-URIs:** Es wurden Konzepte entwickelt, um die sich nach der Evaluierung einer SPARQL-Anfrage gegen die RDF-Graph-Templates ergebenden relevanten Tripel-Templates in URIs zu überführen. Diese URIs adressieren alle Daten, die benötigt werden, um die durch die Tripel-Templates beschriebenen Tripel erzeugen zu können.

Das in dieser Arbeit beschriebene Verfahren kann verwendet werden, um für ein ERP-System auf Basis seiner OData-Schnittstelle einen SPARQL-Endpunkt zu realisieren. Wie in Abschnitt 1.1 beschrieben, kann dieser SPARQL-Endpunkt zur vereinfachten Anfrage des ERP-Systems verwendet werden. Neben der Vereinfachung bestimmter Anfragen an ERP-Systeme kann das vorgestellte Verfahren auch eingesetzt werden, um mehrere ERP-Systeme miteinander zu integrieren.

Um unternehmensübergreifende Kooperationen zu ermöglichen, ist oftmals eine Verknüpfung der in verschiedenen ERP-Systemen vorhandenen Daten notwendig [108]. Abbildung 3 zeigt ein typisches Szenario.

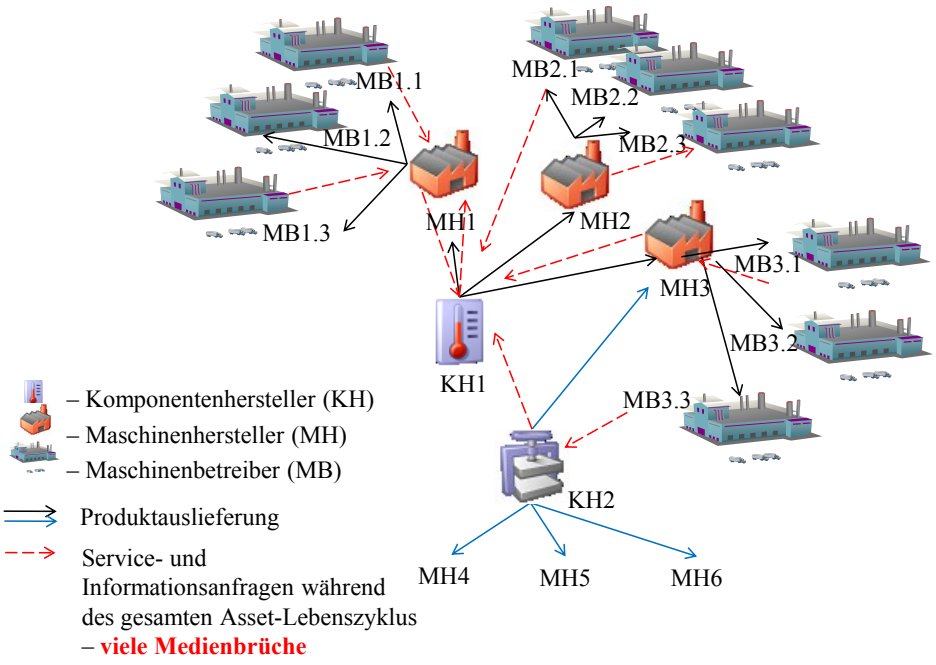


Abbildung 3: Beziehungen zwischen Hersteller und Betreiber [108].

Komponentenhersteller (*KH1*, *KH2*) produzieren Geräte, welche sie an verschiedene Maschinenhersteller (*MH1*, *MH2*, *MH3*) verkaufen. Die Maschinenhersteller verbauen die Geräte in Maschinen und verkaufen diese Maschine an ihre Kunden, die Maschinenbetreiber (*MB1.1*, *MB2.1* usw.). Ein einzelner Maschinenbetreiber (z.B. *MB3.3*) hat somit mehrere Gerätelieferanten (z.B. *KH1*, *MH3*). Diese verfügen über unterschiedliche Informationen zu einem bestimmten Gerät. Um in bestimmten Situationen (wie z.B. dem Ausfall eines Gerätes) effektiv reagieren zu können, ist eine Verknüpfung dieser Informationen notwendig. Mit dem in dieser Arbeit vorgestellten Verfahren können SPARQL-Endpunkte für die jeweiligen ERP-Systeme der involvierten Parteien bereitgestellt werden. Mittels Systemen zur föderierten Abfrage (z.B. [116, 156]) besteht dann die Möglichkeit, die verschiedenen Systeme miteinander zu integrieren. Neben ERP-Systemen kann diese Vorgehensweise für alle Datenquellen verwendet werden, die eine OData-Schnittstelle zur Verfügung stellen. Dadurch werden große Datenmengen, die bisher hinter OData-Services verborgen waren, für die Integration mit dem Semantic Web [44] verfügbar gemacht. Zusammenfassend können mit den in dieser Arbeit eingeführten Konzepten folgende Anwendungsfälle realisiert werden:

- **Vereinfachung der Anfrage bestehender Systeme:** Bestimmte graphbasierte Anfragen können mittels SPARQL sehr kompakt und intuitiv formuliert werden. Mit den in dieser Arbeit vorgestellten Konzepten besteht daher die Möglichkeit, die Anfrage bestehender Systeme, die ein OData-Interface zur Verfügung stellen, zu vereinfachen.
- **Integration bestehender Systeme:** Systeme, die bisher nicht oder nur schwierig miteinander integriert werden konnten, können nun über ihre OData-Schnittstelle mittels SPARQL mit anderen Systemen, die eine SPARQL-Schnittstelle zur Verfügung stellen, z.B. mittels Systemen zur föderierten Abfrage, miteinander integriert werden.

- **Integration von OData-Services in das Semantic Web:** Die in dieser Arbeit vorgestellten Konzepte ermöglichen die Bereitstellung von SPARQL-Endpunkten für jeden bereits existierenden OData-Service. Damit werden große Mengen an Daten, die bisher nicht für die Verarbeitung mittels semantischer Technologien zugänglich waren, für die Integration mit dem Semantic Web verfügbar gemacht.

1.5 Aufbau der Arbeit

Diese Arbeit ist in drei Teile unterteilt. Der erste Teil *Grundlagen* befasst sich in Kapitel 2 mit den für diese Arbeit relevanten Technologien. Insbesondere werden RDF (Abschnitt 2.1), SPARQL (Abschnitt 2.2) und OData (Abschnitt 2.3) vorgestellt. Des Weiteren werden in Kapitel 3 die verwandten Arbeiten aus den Bereichen der semantischen Beschreibung von Web Services (Abschnitt 3.1) und der Service-basierten SPARQL-Endpunkte (Abschnitt 3.2) betrachtet. Der zweite Teil *Lösungsansatz* enthält die wesentlichen Beiträge dieser Arbeit. In Kapitel 4 werden die Konzepte zur semantischen Beschreibung von OData-Services erläutert. Diese umfassen unter anderem eine Erweiterung von CSDL zur Abbildung von Entitätsdatenmodellen auf RDF-Graph-Templates. Des Weiteren wird eine Formalisierung der Konzepte eingeführt, die als Grundlage für die Definition der Evaluierungssemantik für RDF-Graph-Templates in Kapitel 5 dient. Neben der Auflösung von SPARQL-Anfragen gegen die durch die semantischen Beschreibungen der OData-Services gegebenen RDF-Graph-Templates, wird in Kapitel 5 auch die Erzeugung der OData-URIs auf Basis der Templates beschrieben. Der dritte Teil umfasst die *Implementierung und Evaluierung* der vorgestellten Lösung. Die Evaluierung der prototypischen Implementierung ist zweigeteilt. Zunächst erfolgt in Abschnitt 6.1 eine Evaluierung gegen den öffentlich verfügbaren Northwind-Service. Wobei die Gesamtperformance betrachtet wird und die von den einzelnen Komponenten (siehe Abbildung 2) benötigten Zeiten zueinander ins Verhältnis gesetzt werden. In Abschnitt 6.2 werden für einen ERP-Service die Zeiten ermittelt, die von der Query Engine zur Kalkulation der OData-URIs benötigt werden. Die Tests wurden dabei unter Berücksichtigung der bei dem ERP-Szenario relevanten Servicemengen durchgeführt.

2 Grundlagen

2.1 Resource Description Framework (RDF)

Bei den in diesem Abschnitt gemachten Erläuterungen handelt es sich teilweise um eine ergänzte Fassung meiner in [104] gemachten Ausführungen.

RDF (*Resource Description Framework*) [56, 96, 140] ist ein Datenmodell, das unter anderem zur allgemeinen Wissensrepräsentation eingesetzt wird [31]. Der RDF Primer weist darauf hin, dass der beabsichtigte Zweck von RDF insbesondere im Bereich der Repräsentation von Metadaten über Web Ressourcen liegt [140]. RDF bildet die Grundlage für diese Arbeit und ist daher für diese von integraler Bedeutung. RDF wurde vom W3C (*World Wide Web Consortium*) im Rahmen der von Tim Berners-Lee ausgerufenen *Semantic Web* Initiative [44] entwickelt. RDF liegt seit 2004 als W3C Recommendation vor. Als diese Arbeit begonnen wurde, wurde unter der Bezeichnung RDF 1.1 an einer Weiterentwicklung von RDF gearbeitet [69]. Da es sich zu diesem Zeitpunkt um einen Working Draft handelte, bezieht sich diese Arbeit, soweit nicht anders erwähnt, auf die RDF Version von 2004 [56, 96, 140]. Das Namensraum-Präfix *rdf* steht im Folgenden für den RDF-Namensraum (*http://www.w3.org/1999/02/22-rdf-syntax-ns#*).

Ein grundlegender Begriff in RDF ist der einer Aussage (*Statement*). Ein Beispiel einer Aussage ist: „John Doe wohnt in New York“. Eine Aussage setzt sich aus einem Subjekt (*John Doe*), einem Prädikat (*wohnt*) und einem Objekt (*New York*) zusammen. Ein RDF-Dokument besteht aus einer Menge von diesen, auch als *RDF-Tripeln* bezeichneten, Aussagen. Die eindeutige Identifikation des Elements, über das eine Aussage getroffen wird (Subjekt), die sogenannte Ressource, erfolgt in Form einer URI bzw. einer URI-Referenz[2] [45]. Die Person *John Doe* könnte z.B. durch folgende URI eindeutig identifiziert werden: *http://beispiel.org/#John_Doe*. Prädikate werden ebenfalls durch URIs identifiziert, z.B. *http://beispiel.org/#Wohnort*. Objekte hingegen können entweder Literale (New York) oder ein durch eine URI referenziertes Element sein (*http://beispiel.org/#New_York*). Im Rahmen dieser Arbeit werden Aussagen, die als Objekte Literale aufweisen, als *konkrete Aussagen* bezeichnet. Aussagen, die zwei Ressourcen miteinander in Beziehung setzen, indem die URI der entsprechenden Ressource das Objekt der Aussage bildet, werden als *abstrakte Aussagen* bezeichnet. Diese Benennung erfolgt mit Anlehnung an die in OWL verwendeten Bezeichnungen von *abstrakten* und *konkreten Rollen* [99].

[2] Eine URI-Referenz (*URIref*) ist eine URI zusammen mit einem durch das Doppelkreuz (#) eingeleitetem optionalen Fragment. Im Folgenden werden die Begriffe URI und URI-Referenz synonym verwendet.

Die beiden Aussagen „John Doe wohnt in New York“ und „John Doe ist 27 Jahre alt“ können in RDF als Tripel, bestehend aus URIs und Literalen, formuliert werden (siehe Listing 1).

```
1. (http://beispiel.org/#John_Doe,
2.  http://beispiel.org/#Wohnort,
3.  http://beispiel.org/#New_York)
4.
5. (http://beispiel.org/#John_Doe
6.  http://beispiel.org/#Alter,27)
```

Listing 1: RDF-Tripel.

Alles, was durch eine URI identifiziert werden kann, wird in RDF als Ressource bezeichnet. Subjekte und Prädikate sind demnach immer Ressourcen. Objekte können Ressourcen sein. Aussagen selbst sind ebenfalls Ressourcen. Über jede Ressource können Aussagen getroffen werden. Insbesondere ist es möglich, Aussagen über Aussagen zu treffen (*Reifikation*). Zu beachten ist, dass die URIs nicht notwendigerweise auf existierende Web-Ressourcen verweisen müssen. Sie dienen ausschließlich zur Identifizierung. Durch eine Menge von Aussagen über Elemente einer bestimmten Domäne entsteht ein Wissensgraph. Dieser Wissensgraph kann als Datenbasis für ein Inferenzsystem dienen, das mit entsprechenden Schlussfolgerungstechniken aus diesem neues (implizites) Wissen gewinnen kann.

Literale (wie der Wert *27* in Zeile 6, Listing 1) können in RDF typisiert werden. RDF definiert, mit Ausnahme des *rdf:XMLLiteral*-Datentyps[3], keine eigenen Datentypen. Vielmehr können Datentypen aus beliebigen Namensräumen (wie z.B. XSD [130]) verwendet werden, die einem bestimmten konzeptionellen Rahmenwerk entsprechen [56].

RDF definiert ein Prädikat *rdf:type*, um eine Ressource als Instanz einer Klasse zu spezifizieren. Damit kann *John Doe* der Typ *Student* zugewiesen werden (siehe Listing 2). Die RDF-Spezifikation spezifiziert mehrere Klassen mit spezieller Bedeutung, z.B. *rdf:Property* (Klasse aller Prädikate).

```
1. (http://beispiel.org/#John_Doe, rdf:type,
2.  http://beispiel.org/#Student)
```

Listing 2: Die Typangabe mit *rdf:type*.

Eine für den Menschen leichter verständliche Darstellungsform, als die in Listing 1 und Listing 2 dargestellte Tripel-Darstellung, ist der Graph (siehe Abbildung 4). Subjekte und Objekte werden durch Knoten repräsentiert. Dabei werden URIs in Ellipsen und Literale in Rechtecke eingefasst. Die Prädikate bilden die Kanten zwischen den Knoten.

[3] Mit *rdf:XMLLiteral* können XML-Inhalte als Literale repräsentiert werden.

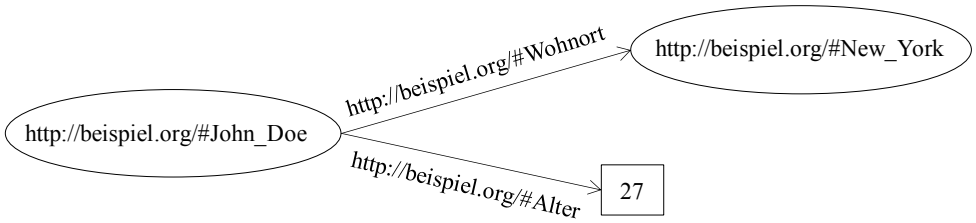


Abbildung 4: RDF-Graph.

Es existieren verschiedene Darstellungsformen (z.B. N3 [43], Turtle [40] und RDF/XML [38]), die je nach Anwendungszweck unterschiedlich gut geeignet sind. Im Rahmen dieser Arbeit werden die Graph-Darstellung und Turtle verwendet. Die Graph-Darstellung wurde bereits gezeigt. Turtle wird in Abschnitt 2.1.1 erläutert. Obwohl RDF/XML ein gängiges Serialisierungsformat ist, das von vielen Tools unterstützt wird, ist es für das Verständnis dieser Arbeit nicht von Bedeutung. Es wird daher nicht weiter erläutert.

Ein RDF-Graph kann sogenannte *leere Knoten* (*blank nodes*) enthalten. Leere Knoten sind Ressourcen, denen keine URI zur Identifizierung zugewiesen wurde. Sie können als Subjekt oder Objekt, aber nicht als Prädikat einer Aussage eingesetzt werden. Abbildung 5 zeigt einen leeren Knoten, welcher eine Adresse repräsentiert. Leere Knoten dienen dazu, zusätzliche Struktur in den RDF-Graphen einzuführen, ohne dass die Erzeugung zusätzlicher Platzhalter URIs notwendig wird. Zur Unterscheidung mehrerer leerer Knoten in einem RDF-Graphen werden diesen bei den meisten Serialisierungformaten Knoten-IDs zugewiesen (siehe z.B. Listing 7 in Abschnitt 2.1.1). Diese Knoten-IDs dienen nicht zur eindeutigen Identifizierung der leeren Knoten, sondern nur zu deren Unterscheidung. Im Gegensatz zu URIs sind sie nur innerhalb eines Dokumentes eindeutig. Des Weiteren können sie geändert werden, ohne dass sich die Bedeutung der Aussagen bzw. des RDF-Graphen ändert [99].

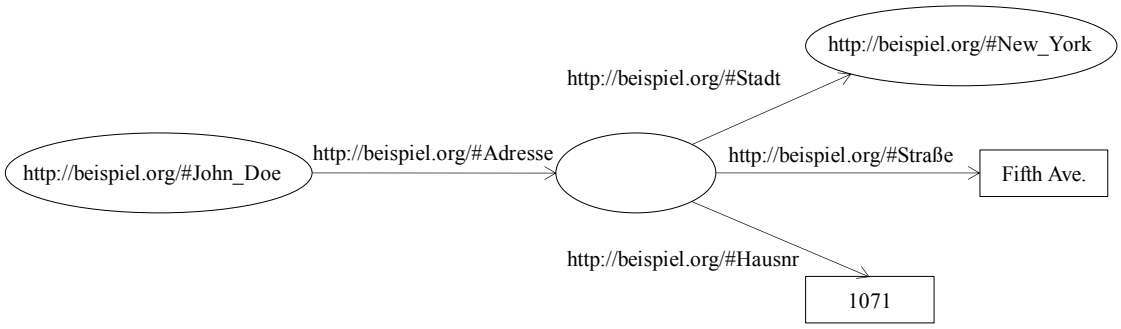


Abbildung 5: Leerer Knoten (angelehnt an das Beispiel aus [140]).

Zur Definition listenartiger Strukturen stellt RDF spezielle Konstrukte bereit. RDF unterscheidet zwei Arten von Listen: *Container* (offene Listen) und *Collections* (geschlossene Listen) [99]. RDF definiert drei Arten offener Listen: *rdf:Seq* (geordnete Listen), *rdf:Bag* (ungeordnete Liste), *rdf:Alt* (Liste alternativer Werte). Abbildung 6 zeigt eine ungeordnete Liste, welche die Wohnorte von *John Doe* auflistet.

Abbildung 6: Eine ungeordnete Liste (*rdf:Bag*) in RDF.

Listen werden oftmals durch leere Knoten realisiert[4], die über das *rdf:type*-Prädikat den Listentyp (z.B. *rdf:Bag*) zugewiesen bekommen. Die Elemente einer Liste werden mittels nummerierter Prädikate (*rdf:_1*, *rdf:_2*, *rdf:_3*, …) referenziert. Für geordnete Listen wird die Reihenfolge durch die Nummern bestimmt. Bei ungeordneten Listen ist die Nummerierung nicht von Bedeutung. Mit *rdf:_1* wird bei Listen alternativer Werte der Standard- oder bevorzugte Wert spezifiziert. Die Nummerierung bei allen weiteren Elementen ist nicht maßgeblich. Dem Namen entsprechend, sind offene Listen nie abgeschlossen. Die in Abbildung 6 dargestellte Liste enthält mindestens die drei angegebenen Elemente. Sie kann aber noch weitere enthalten.

Um Listen spezifizieren zu können, die ausschließlich die angegebenen Elemente enthalten, definiert RDF geschlossene Listen (*Collections*). Abbildung 7 zeigt das bereits bekannte Beispiel in Form einer geschlossenen Liste.

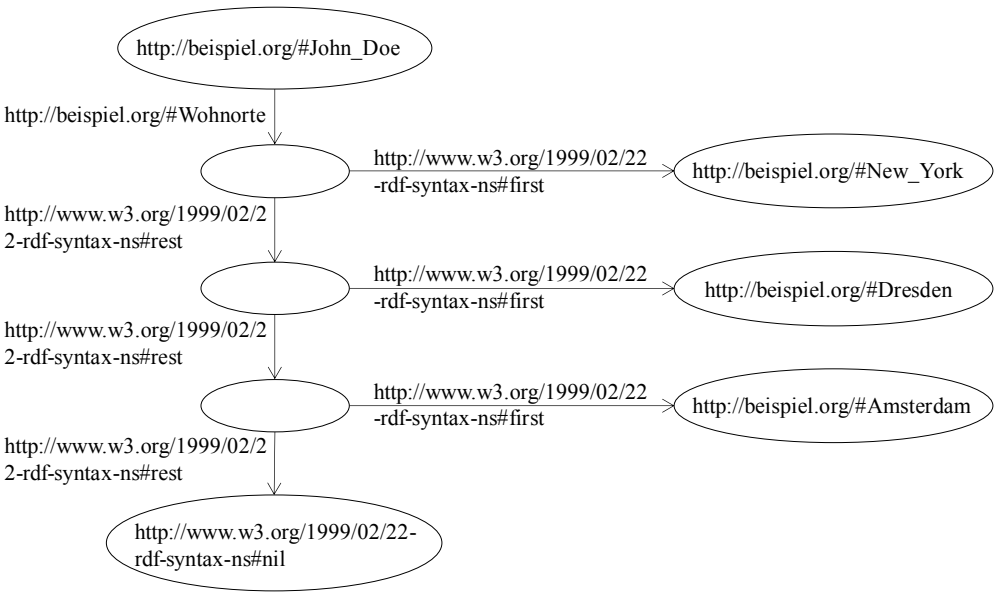


Abbildung 7: Eine geschlossene Liste (*Collection*) in RDF.

[4] Obwohl Listen zur Referenzierung auch URIs zugewiesen werden können.

Eine geschlossene Liste ist eine Verkettung (leerer) Knoten des Typs *rdf:List* (wobei dieser Typ implizit für jeden Knoten angenommen wird), für welche jeweils zwei Aussagen mit den beiden Prädikaten *rdf:first* und *rdf:rest* existieren. Das Prädikat *rdf:first* weist dabei jeweils auf das entsprechende Element, das sich an dieser Position der Liste befindet. Der nächste Knoten der Liste wird über das *rdf:rest*-Prädikat referenziert. Abgeschlossen wird die Liste mit *rdf:nil* (leere Liste).

Wie bereits erwähnt, besteht in RDF die Möglichkeit, Aussagen über Aussagen zu treffen (*Reifikation*). RDF definiert dazu die Klasse *rdf:Statement*. Aussagen werden als Instanzen dieser Klasse repräsentiert. In dem Beispiel aus Abbildung 8 wird der Aussage „John Doe wohnt in New York“ ein Urheber *be:PersonA*[5] zugewiesen. Dazu wurde eine Instanz der Klasse *rdf:Statement* als leerer Knoten definiert. Dieser leere Knoten referenziert über die drei Prädikate *rdf:subject*, *rdf:predicate* und *rdf:object* das Subjekt, das Prädikat und das Objekt der Aussage. Die Zuweisung des Urhebers zu dieser Aussage geschieht über eine Referenzierung von *be:PersonA* durch den leeren Knoten über das *be:Urheber*-Prädikat.

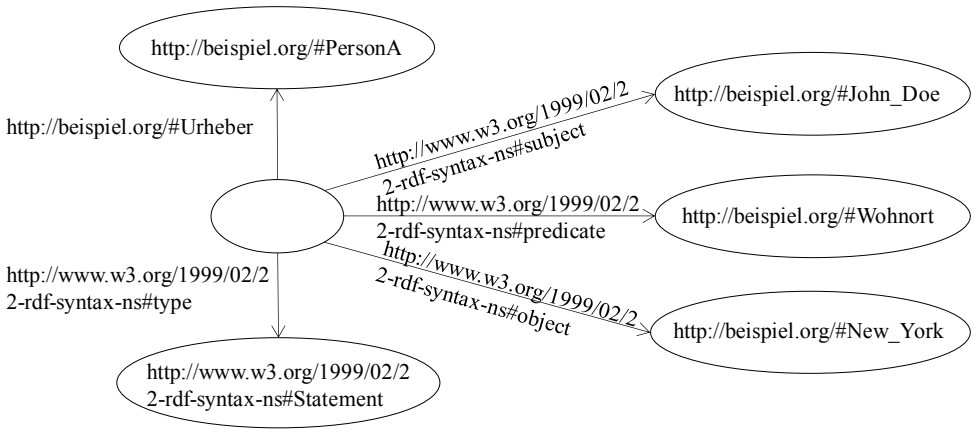


Abbildung 8: Reifikation

2.1.1 Terse RDF Triple Language (Turtle)

Die *Terse RDF Triple Language* (*Turtle*) [40] ist ein kompaktes, für den Menschen leicht lesbares Serialisierungsformat für RDF-Graphen. Turtle ist eine Untermenge von N3 [43] und wird unter anderem, in abgewandelter Form, von der in Abschnitt 2.2 beschriebenen RDF-Abfragesprache SPARQL verwendet. Im Gegensatz zu RDF/XML, mit der bestimmte RDF-Graphen nicht serialisiert werden können, ermöglicht Turtle die Darstellung aller RDF-Graphen [40]. Listing 3 zeigt die Turtle-Darstellung für den aus Abbildung 4 bekannten RDF-Graphen.

[5] Das Präfix *be* steht für den Namensraum *http://beispiel.org/#*.

```
1. # Das ist ein Turtle-Dokument
2. <http://beispiel.org/#John_Doe>
3. <http://beispiel.org/#Wohnort>
4. <http://beispiel.org/#New_York> .
5. <http://beispiel.org/#John_Doe>
6. <http://beispiel.org/#Alter> "27"^^xsd:integer .
```

Listing 3: Ein RDF-Graph in Turtle.

Einzelne Tripel werden in Turtle durch einen Punkt abgegrenzt. Die URIs der Ressourcen werden in spitze Klammern eingeschlossen. Die Angabe von Literalen erfolgt innerhalb von Apostrophen oder Anführungszeichen. Wenn das Literal Zeilenumbrüche enthält, muss es durch drei Apostrophe (''') oder drei Anführungszeichen (""") eingefasst werden.

Optional kann einem Literal ein Datentyp oder eine Sprachangabe zugewiesen werden. Die Angabe des Datentyps erfolgt durch ^^, gefolgt von der URI des Datentyps (siehe Listing 3: Zeile 6). Wenn die Typ-Angabe fehlt, wird der Typ *xsd:string* angenommen. Das Präfix *xsd* steht im Folgenden für den XSD-Namensraum (*http://www.w3.org/2001/XMLSchema#*) [130]. Zahlen müssen nicht durch Apostrophe oder Anführungszeichen eingeschlossen werden. Der Datentyp wird in Abhängigkeit der angegebenen Zahl als *xsd:integer*, *xsd:decimal* oder *xsd:double* festgelegt. Boolesche Werte können ebenfalls ohne Apostrophe bzw. Anführungszeichen mittels *true* oder *false* angegeben werden.

Dem Literal wird eine Sprachangabe zugewiesen, indem ein @, gefolgt von dem Sprach-Identifier (z.B. *en*), an das Literal angehangen wird. Kommentare werden durch die Raute eingeleitet. Wie in XML besteht die Möglichkeit, Namensräume durch Präfixe abzukürzen (siehe Listing 4). In den folgenden Beispielen wird die Definition des Präfixes aus Gründen der Übersichtlichkeit weggelassen.

```
1. @prefix be:<http://beispiel.org/#>
2. be:John_Doe be:Wohnort be:New_York .
3. be:John_Doe be:Alter 27
```

Listing 4: Abkürzende Schreibweise für Namensräume in Turtle.

Um das redundante Aufzählen von URIs bei Aussagen über dasselbe Subjekt zu vermeiden, kann in Turtle das Semikolon verwendet werden (siehe Listing 5).

```
1. be:John_Doe be:Wohnort be:New_York ;
2.             be:Alter 27 .
```

Listing 5: Die abgekürzte Schreibweise für mehrere Aussagen über dasselbe Subjekt in Turtle.

Wenn sowohl Subjekt als auch Prädikat übereinstimmen, ermöglicht Turtle die verkürzte Angabe mit dem Komma. Listing 6 demonstriert die Darstellung für den Fall, dass John Doe sowohl in New York als auch in Amsterdam einen Wohnsitz hat.

```
1. be:John_Doe be:Wohnort be:New_York, be:Amsterdam .
```

Listing 6: Mehrere Aussagen mit gleichem Subjekt und Prädikat in Turtle.

Bei leeren Knoten wird das Namensraumpräfix durch einen Unterstrich ersetzt. Dem Doppelpunkt folgt die ID des leeren Knotens.

```
1. be:John_Doe be:Adresse _:ad .
2. _:ad be:Strasse "Fifth Ave."
3. _:ad be:Hausnr "1071"
```

Listing 7: Leerer Knoten in Turtle.

Leere Knoten können in abgekürzter Form als unbeschriftete, leere Knoten angegeben werden. Die Definition erfolgt mittels eckiger Klammer, wobei die ID des leeren Knotens entfällt. Listing 8 zeigt den RDF-Graphen aus Listing 7 unter Verwendung unbeschrifteter, leerer Knoten. Diese können beliebig tief geschachtelt werden.

```
1. be:John_Doe be:Addresse [
2.             be:Strasse "Fifth Ave." ;
3.             be:Hausnr 1071 ] .
```

Listing 8: Abgekürzte Schreibweise für leere Knoten in Turtle.

Geschlossene Listen (*Collections*) können in Turtle mittels runder Klammern dargestellt werden (siehe Listing 9). Im Gegensatz zu den in Listing 6 gemachten Angaben impliziert die geschlossene Liste in Listing 9 eine Struktur. Die Ressource *be:New_York* befindet sich in der Liste vor *be:Amsterdam*, was z.B. so interpretiert werden könnte, dass New York der bevorzugte Wohnsitz von John Doe ist. Für offene Listen (*Container*) bietet Turtle keine spezielle Syntax.

```
1. be:John_Doe be:Wohnort (be:New_York be:Amsterdam)
```

Listing 9: Darstellung geschlossener Listen mit Turtle.

2.2 SPARQL Protocol and RDF Query Language (SPARQL)

SPARQL (*SPARQL Protocol and RDF Query Language*) ist eine Anfragesprache für RDF-Graphen. SPARQL wurde wie RDF vom W3C im Rahmen der Semantic Web Initiative entwickelt. Es liegt seit 2008 als W3C Recommendation vor [155]. Des Weiteren existiert eine Empfehlung des W3C hinsichtlich der Übertragung von Anfragen (*SPARQL Protocol*) [62] und der Kodierung von Anfrageergebnissen in XML (*SPARQL Query Results XML Format*) [39]. Mittlerweile existiert unter der Versionsnummer 1.1 eine Weiterentwicklung von SPARQL [91, 179]. Da sich SPARQL 1.1 zu Beginn dieser Arbeit noch in der Entwicklung befand [92], bezieht sich diese Arbeit auf Version 1.0 [155].

2.2.1 Sprachelemente

Listing 10 zeigt eine SPARQL-Anfrage, die den Wohnort aller Studenten zurückgibt, die 27 Jahre alt sind. Für den in Listing 11 dargestellten RDF-Graphen liefert diese SPARQL-Anfrage das in Tabelle 1 dargestellte Ergebnis.

```
1. PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2. PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
3. PREFIX be:<http://beispiel.org/#>
4. SELECT ?student ?wohnort
5. WHERE {
6.  ?student rdf:type be:Student .
7.  ?student be:Alter '27'^^xsd:integer .
8.  ?student be:Wohnort ?wohnort
9. }
```

Listing 10: Eine SPARQL-Anfrage (angelehnt an die Anfrage aus [104]).

Die dargestellte SPARQL-Anfrage setzt sich aus drei Teilen zusammen. Einer Menge von Namensraumdeklarationen (*PREFIX*), der Angabe der Ergebnisvariablen (*SELECT*) und einem durch das *WHERE*-Schlüsselwort eingefassten einfachen Graph-Muster (*BGP*[6]). Das einfache Graph-Muster beschreibt den gesuchten Graphen. Es besteht aus einer Menge von sogenannten *Tripel-Pattern*. Ein Tripel-Pattern ist ein RDF-Tripel, dessen Subjekt, Prädikat oder Objekt durch eine Variable ersetzt sein kann. Die Notation ist an das in Abschnitt 2.1.1 erläuterte Turtle angelehnt[7]. Variablen werden durch vorgestellte Fragezeichen (oder durch das Dollarzeichen) gekennzeichnet.

```
1. @prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2. @prefix be:<http://beispiel.org/#>
3. be:John_Doe rdf:type be:Student ;
4.             be:Alter 27 ;
5.             be:Wohnort be:New_York, be:Amsterdam .
6. be:David_Mills rdf:type be:Student ;
7.                be:Alter 27 ;
8.                be:Wohnort be:New_York .
```

Listing 11: RDF-Graph

[6] Basic Graph Pattern

[7] Es werden auch die aus Turtle bekannten Abkürzungen mit Komma und Semikolon unterstützt.

Eine Variable ist ein Platzhalter, der während der Anfrageverarbeitung durch konkrete Werte aus dem RDF-Graphen ersetzt wird. Der SPARQL-Prozessor versucht, für das durch die Tripel-Pattern beschriebene Graph-Muster, während der Anfrageverarbeitung, einen passenden RDF-Teilgraphen zu ermitteln. Für jeden passenden Teilgraphen werden die Belegungen der in *SELECT*-Anweisung aufgelisteten Variablen in die Ergebnismenge der SPARQL-Anfrage aufgenommen.

Die Angabe von Literalen in Tripel-Pattern mit bestimmten Datentypen oder Sprachangaben kann mit der aus Turtle bekannten Notation (^^ bzw. @) erfolgen (vergleiche Listing 3).

student	**wohnort**
be:John_Doe	be:New_York
be:John_Doe	be:Amsterdam
be:David_Mills	be:New_York

Tabelle 1: SPARQL-Ergebnistabelle.

Mit dem Schlüsselwort *UNION* können alternative Graph-Muster spezifiziert werden. Listing 12 zeigt eine Beispiel-Anfrage[8], welche die Wohnorte aller Studenten zurückgibt, die 27 oder 28 Jahre alt sind.

```
1. SELECT ?student ?wohnort
2. WHERE {
3.  ?student rdf:type be:Student ;
4.           be:Wohnort ?wohnort .
5.  { ?student be:Alter 27 } UNION
6.  { ?student be:Alter 28 }
7. }
```

Listing 12: Eine SPARQL-Anfrage mit dem *UNION*-Operator (angelehnt an die Anfrage aus [104]).

Das Ergebnis dieser SPARQL-Anfrage entspricht dabei der Vereinigung der Ergebnisse zweier, wie in Listing 10 dargestellter, SPARQL-Anfragen, wobei für die zweite Anfrage der Wert für das *be:Alter*-Prädikat auf 28 Jahre gesetzt werden muss. Die alternativen Graph-Muster werden durch geschweifte Klammern eingefasst. Solche durch Klammern eingefasste Graph-Muster werden auch als *gruppierende Graph-Muster* bezeichnet. Jede SPARQL-Anfrage enthält mindestens ein gruppierendes Graph-Muster. Dieses folgt auf die *WHERE*-Anweisung.

SPARQL bietet die Möglichkeit, Graph-Muster als optional zu kennzeichnen. Listing 13 zeigt eine SPARQL-Anfrage, die neben dem Wohnort auch die Straße, in dem der Student wohnt, zurückgibt. Da das Graph-Muster zur Abfrage der Straße als optional spezifiziert wurde, führt das Nichtvorhandensein dieser Angabe im RDF-Graphen nicht zu einem

[8] Die Definitionen der Namensraumpräfixe werden aus Gründen der Übersichtlichkeit in den folgenden SPARQL-Anfragen nicht mehr explizit angegeben.

Fehlschlag der Anfrage. Stattdessen bleibt die Variable in der Ergebnismenge ungebunden. *UNION*- und *OPTIONAL*-Anweisungen können beliebig geschachtelt werden.

```
1. SELECT ?student ?wohnort ?strasse
2. WHERE {
3.  ?student rdf:type be:Student ;
4.           be:Alter 27 .
5.           be:Wohnort ?wohnort .
6.  OPTIONAL {
7.   ?student be:Adresse ?adresse .
8.   ?adresse be:Strasse ?strasse }
9. }
```

Listing 13: Die Angabe eines optionalen Graph-Musters in SPARQL (angelehnt an die Anfrage aus [104]).

Zur Filterung der Daten stellt SPARQL das Schlüsselwort *FILTER* zur Verfügung. In dem in Listing 14 dargestellten Beispiel erfolgt eine Filterung der Studenten nach deren Alter. Es werden nur die Studenten in der Ergebnismenge berücksichtigt, die älter als 25 Jahre sind. *FILTER*-Anweisungen beziehen sich immer auf das gesamte gruppierende Graph-Muster, in dem sich die Anweisung befindet. Die Position der *FILTER*-Anweisung innerhalb des gruppierenden Graph-Musters ist dabei nicht von Bedeutung.

```
1. SELECT ?student ?wohnort
2. WHERE {
3.  ?student rdf:type be:Student .
4.  ?student be:Wohnort ?wohnort
5.  ?student be:Alter ?alter.
6.  FILTER (?alter > 25)
7. }
```

Listing 14: Die *FILTER*-Anweisung in SPARQL.

SPARQL stellt auf Basis der XML-Anfragesprachen XQuery und XPath [131] eine Menge von Funktionen und Operatoren bereit, die in den Filterbedingungen verwendet werden können. Dazu gehören unter anderem Vergleichsoperatoren (=, !=, >, >=, <, <=) für die gängigsten Datentypen. Des Weiteren werden verschiedene Funktionen definiert, um z.B. Variablenwerte auf bestimmte Bedingungen zu prüfen. In Tabelle 2 sind einige Funktionen aufgelistet.

Neben diesen Funktionen bietet SPARQL zur Verknüpfung von Filterbedingungen folgende Operatoren: && (Konjunktion), || (Disjunktion) und ! (Negation). Des Weiteren werden grundlegende arithmetische Operatoren bereitgestellt: + (Addition), - (Subtraktion), / (Division), * (Multiplikation).

Funktion	Semantik
BOUND(A)	Gibt *true* zurück, wenn die Variable A gebunden ist, ansonsten *false*.
isIRI(A)	Gibt *true* zurück, wenn A eine IRI ist, ansonsten *false*.
isURI(A)	Gibt *true* zurück, wenn A eine URI ist, ansonsten *false*.
isBLANK(A)	Gibt *true* zurück, wenn A ein leerer Knoten ist, ansonsten *false*.
isLITERAL(A)	Gibt *true* zurück, wenn A ein Literal ist, ansonsten *false*.
STR(A)	Gibt die String-Repräsentation (xsd:string) von A zurück.
LANG(A)	Gibt die Sprachangabe des Literals A zurück.
DATATYPE(A)	Gibt den Datentyp von A zurück (in Form einer URI).
sameTerm(A, B)	Gibt *true* zurück, wenn A und B die gleichen Terme sind, ansonsten *false*.
LANGMATCHES(A, B)	Gibt *true* zurück, wenn A eine Sprachangabe des Musters B aufweist, ansonsten *false*.
REGEX(A, B)	Gibt *true* zurück, wenn A auf den regulären Ausdruck B passt, ansonsten *false*.

Tabelle 2: Verschiedene in SPARQL unterstützte Filter-Funktionen.

Außer der bereits bekannten *SELECT*-Anweisung bietet SPARQL zur Spezifikation des Ausgabeformats die drei Anweisungen *CONSTRUCT*, *DESCRIBE* und *ASK*. Mit der *CONSTRUCT*-Anweisung ist es möglich, RDF-Graphen als Ausgabe einer SPARQL-Anfrage zu erzeugen. Listing 15 zeigt ein Beispiel. Auf die *CONSTRUCT*-Anweisung folgt ein Graph-Muster, welches das Format des Ausgabegraphen spezifiziert. Für jede mögliche Variablenbelegung wird der angegebene Graph erzeugt. Das Graph-Muster kann einen beliebig umfangreichen Graphen beschreiben. Wobei dieser auch konstante Literale und leere Knoten enthalten kann.

```
1. CONSTRUCT { ?student be:WohntIn ?wohnort . }
2. WHERE {
3.  ?student rdf:type be:Student .
4.  ?student be:Alter 27 .
5.  ?student be:Wohnort ?wohnort
6. }
```

Listing 15: Die Verwendung des *CONSTRUCT*-Operators in SPARQL.

Mit dem Schlüsselwort *ASK* ist es möglich, zu prüfen, ob im RDF-Graphen ein Teilgraph existiert, der auf das angegebene Graph-Muster passt. Die Anfrage aus Listing 16 gibt *true* zurück, wenn es einen Studenten gibt, der 27 Jahre alt ist. Ansonsten wird *false* zurückgegeben.

```
1. ASK {
2.  ?student rdf:type be:Student .
3.  ?student be:Alter 27 }
```

Listing 16: Die Verwendung des *ASK*-Operators in SPARQL.

Wenn z.B. die Struktur der Informationen nicht bekannt ist, kann mittels des *DESCRIBE*-Schlüsselwort die Entscheidung darüber, welche Informationen über eine bestimmte Ressource zurückgegeben werden sollen, an den SPARQL-Endpunkt abgegeben werden. In Listing 17 entscheidet der Server, welche Informationen über die gefundenen Studenten zurückgegeben werden.

```
1. DESCRIBE ?student
2. WHERE {
3.  ?student rdf:type be:Student .
4.  ?student be:Alter 27 }
```

Listing 17: Der SPARQL-*DESCRIBE*-Operator.

Um die Lösungsmenge einzugrenzen und zu sortieren, bietet SPARQL mehrere Modifikatoren. Tabelle 3 gibt einen Überblick. Es sei angemerkt, dass die bereits bekannte *SELECT*-Anweisung (*Projection*) auch zu den Modifikatoren gezählt wird. Da diese bereits erläutert wurde, wird sie hier nicht noch einmal explizit aufgeführt.

Modifikator	**Semantik**
ORDER BY	Ermöglicht die Sortierung der Ergebnismenge.
DISTINCT	Entfernt Duplikate aus der Ergebnismenge.
REDUCED	Die Entscheidung, ob Duplikate entfernt werden, liegt beim SPARQL-Prozessor. Es können alle, keine oder manche Duplikate entfernt werden.
OFFSET	Gibt die Ergebnisse ab der angegebenen Position zurück.
LIMIT	Gibt nur die angegebene Anzahl von Elementen zurück.

Tabelle 3: Die von SPARQL unterstützten Modifikatoren.

Als Beispiel für die Verwendung eines Modifikators zeigt Listing 18 die Sortierung der Studenten nach dem Alter mit *ORDER BY*. Die Sortierung erfolgt aufsteigend (*ASC*). Eine absteigende Sortierung kann mit der *DESC*-Funktion erzwungen werden.

```
1. SELECT ?student ?alter
2. WHERE {
3.  ?student rdf:type be:Student .
4.  ?student be:Alter ?alter }
5. ORDER BY ASC(?alter)
```

Listing 18: Die Sortierung in SPARQL mit dem *ORDER BY*-Operator.

Alle bisher gezeigten SPARQL-Anfragen werden vom SPARQL-Prozessor gegen den sogenannten Standard-Graphen (*Default Graph*) ausgeführt. Der Standard-Graph kann üblicherweise für den SPARQL-Prozessor konfiguriert werden oder er wird bei Aufruf über den SPARQL Service [62] angegeben. Des Weiteren stellt der SPARQL-Standard mit dem Schlüsselwort *FROM* eine Möglichkeit bereit, den Standard-Graphen gegen den die SPARQL-Anfrage ausgeführt werden soll, direkt in der SPARQL-Anfrage zu spezifizieren.

Außerdem besteht die Möglichkeit, mit *FROM NAMED* sogenannte *benannte Graphen* zu definieren, auf die über das *GRAPH*-Schlüsselwort innerhalb der SPARQL-Anfrage direkt Bezug genommen werden kann. Listing 19 zeigt die Verwendung benannter Graphen.

```
1. SELECT ?src ?student
2. FROM NAMED <http://graph.org/g1>
3. FROM NAMED <http://graph.org/g2>
4. WHERE {
5.  GRAPH ?src {
6.  ?student rdf:type be:Student .
7.  ?student be:Alter 27 }
8. }
```

Listing 19: Benannte Graphen in SPARQL.

In dem dargestellten Beispiel wurden zwei benannte Graphen definiert. Auf das *GRAPH*-Schlüsselwort folgt eine Variable, die während der Ausführung jeweils als Wert die URIs der benannten Graphen annimmt. Das durch die eckigen Klammern des *GRAPH*-Schlüsselwortes eingeschlossene Graph-Muster wird gegen beide benannten Graphen evaluiert. Die gefundenen Studenten werden zusammen mit der jeweiligen URI des Graphen, in dem sie gefunden wurden, ausgegeben. Anstelle einer Variablen kann auf das *GRAPH*-Schlüsselwort auch die URI eines Graphen folgen. In diesem Fall wird das Graph-Muster nur gegen diesen Graphen ausgeführt.

2.2.2 SPARQL-Algebra

Die Semantik der Sprachelemente wurde in dem vorhergehenden Abschnitt umgangssprachlich erläutert. Solche umgangssprachlichen Erläuterungen können oft unterschiedlich interpretiert werden. Um die Ergebnismenge einer SPARQL-Anfrage für einen gegebenen RDF-Graphen eindeutig bestimmen zu können, ist eine formale Definition der Semantik der SPARQL-Operatoren notwendig. Zu diesem Zweck wurde die SPARQL-Algebra geschaffen. Die SPARQL-Algebra ermöglicht es, die Ergebnismenge für eine gegebene SPARQL-Anfrage auszurechnen [99]. SPARQL und die SPARQL-Algebra können mit denen aus dem Bereich der relationalen Datenbanken bekannten SQL und der relationalen Algebra verglichen werden. Im Gegensatz zur relationalen Algebra, welche nicht mächtig genug ist, um SQL vollständig abbilden zu können, kann jede mögliche SPARQL-Anfrage in einen Ausdruck der SPARQL-Algebra überführt werden. Es sei aber darauf hingewiesen, dass die SPARQL-Algebra die gleiche Ausdrucksstärke wie die relationale Algebra besitzt [30]. Im Folgenden wird die SPARQL Algebra vorgestellt und es wird gezeigt, wie SPARQL-Anfragen in einen Ausdruck der SPARQL-Algebra überführt werden können. Des Weiteren wird erläutert, wie diese aufgelöst werden. Die Evaluierungssemantik der SPARQL-Algebra, wie sie durch die SPARQL-Spezifikation definiert ist, bildet den Ausgangspunkt für die in Kapitel 5 vorgestellte, mit Hinblick auf RDF-Graph-Templates entwickelte Evaluierungssemantik. Sie ist daher für diese Arbeit von fundamentaler Bedeutung.

Listing 20 zeigt eine SPARQL-Anfrage, die im Folgenden als Beispiel für die Überführung in die SPARQL-Algebra dient. Die dargestellte Abfrage gibt alle Studenten zurück, die älter als 25 Jahre sind und in New York oder Amsterdam wohnen. Falls die Angabe vorhanden ist, in welchem Semester sich der jeweilige Student befindet, so wird diese Information ebenfalls zurückgegeben.

```
1. SELECT ?student ?semester
2. WHERE {
3.  ?student be:Alter ?alter .
4.  FILTER (?alter > 25)
5.  OPTIONAL { ?student be:Semester ?semester }
6.  { ?student be:Wohnort be:New_York }
7.  UNION
8.  { ?student be:Wohnort be:Amsterdam }
9. }
```

Listing 20: Eine SPARQL-Anfrage, welche in einen Ausdruck der SPARQL-Algebra überführt werden soll.

Listing 21 zeigt den zu dieser Abfrage gehörenden Ausdruck in SPARQL-Algebra.

```
1. Project(
2.  ToList(
3.   Filter((?alter>25),
4.    Join(
5.     LeftJoin(
6.      Join(Z,
7.       BGP(?student <http://beispiel.org/#Alter> ?alter)),
8.      Join(Z,
9.       BGP(?student <http://beispiel.org/#Semester>
10.          ?semester))),
11.    Union(
12.     Join(Z,BGP(?student <http://beispiel.org/#Wohnort>
13.                 <http://beispiel.org/#New_York>)),
14.     Join(Z,BGP(?student <http://beispiel.org/#Wohnort>
15.                 <http://beispiel.org/#Amsterdam>))))))
16.),{?student,?semester})
```

Listing 21: Der SPARQL-Algebra-Ausdruck für die SPARQL-Anfrage aus Listing 20.

Die folgende Erläuterung zur Überführung von SPARQL-Anfragen in Ausdrücke der SPARQL-Algebra entspricht im Wesentlichen den in der SPARQL-Spezifikation [155] gemachten Ausführungen. Der erste Schritt bei der Überführung einer SPARQL-Anfrage in einen Ausdruck der SPARQL-Algebra besteht darin, dass alle Abkürzungen aufgelöst werden. Insbesondere werden die durch Präfixe abgekürzten URIs durch die vollständige URI-Angabe ersetzt. Anschließend wird das gruppierende Graph-Muster der *WHERE*-Klausel der

SPARQL-Anfrage an die im Folgenden erläuterte *trans*-Funktion übergeben, welche diese rekursiv in einen Ausdruck der SPARQL-Algebra G überführt. Dabei wird bei jedem Funktionsaufruf zunächst $G \coloneqq Z$ gesetzt, wobei Z das leere Graph-Muster ist. Es werden folgende Operatoren der SPARQL-Algebra eingeführt: *BGP*, *Join*, *LeftJoin*, *Filter*, *Union* und *Graph*. Entsprechend des Typs des übergebenen Graph-Musters P werden bei der Abarbeitung der *trans*-Funktion folgende Fallunterscheidungen getroffen:

Einfaches Graph-Muster: Wenn P ein einfaches Graph-Muster ist, dann gilt $G \coloneqq BGP(P)$.

Gruppierende Graph-Muster: Aus jedem Element E des gruppierenden Graph-Musters P werden zunächst alle Filteranweisungen der Form *FILTER(e)* entfernt. Jedes Element E wird nun folgendermaßen in einen Ausdruck der SPARQL-Algebra überführt, wobei zwei Fälle unterschieden werden müssen:

1.) Wenn E die Form *OPTIONAL p* hat, dann wird zunächst für das Graph-Muster p der zugehörige SPARQL-Algebra-Ausdruck a berechnet: $a \coloneqq trans(p)$. Anschließend werden zwei Fälle unterschieden:

- Wenn a die Form $Filter(e, a_2)$ hat, dann setze $G \coloneqq LeftJoin(G, a_2, e)$
- Anderenfalls setze $G \coloneqq LeftJoin(G, a, true)$

2.) Wenn E eine andere Form hat, dann setze $G \coloneqq Join(G, trans(E))$.

Wenn das gruppierende Graph-Muster Filteranweisungen enthalten hat, dann werden die Ausdrücke $e_1, e_2, \dots, e_n$ der entfernten Filteranweisungen konjunktiv verknüpft und zusammen mit dem ermittelten Ausdruck der SPARQL-Algebra G als Argument an den SPARQL-Algebra-Operator *Filter* übergeben: $G \coloneqq Filter(e_1 \&\& e_2 \&\& \dots \&\& e_n, G)$

Vereinigung von gruppierenden Graph-Mustern: Wenn P zwei Graph-Muster p_1 und p_2 enthält, die über das SPARQL-Schlüsselwort *UNION* verbunden sind (p_1 *UNION* p_2), dann gilt: $G \coloneqq Union(trans(p_1), trans(p_2))$.

Benannte Graphen: Sei e eine IRI[9] oder eine Variable und sei p ein Graph-Pattern. Für das Graph-Pattern *GRAPH e a* ist der zugehörige Ausdruck der SPARQL-Algebra gegeben durch: $G \coloneqq Graph(e, trans(a))$.

Die bereits erwähnten Operatoren der SPARQL-Algebra werden entweder auf Graph-Mustern (*BGP*) oder, wie in Abschnitt 2.2.3 noch erläutert wird, auf Multimengen von Lösungen (*Union*, *Filter*, *Join*, *LeftJoin*) ausgeführt. Die im Folgenden zur Überführung der Modifikatoren (Tabelle 3) vorgestellten SPARQL-Algebra-Operatoren erwarten als Parameter Sequenzen von Lösungen (*solution mappings*). Zur Überführung der von den Operatoren

[9] Internationalized Resource Identifier: Eine Obermenge der URIs, welche neben dem lateinischen Alphabet auch andere Zeichen (z.B. chinesische Schriftzeichen) erlaubt [74].

BGP, *Union*, *Filter*, *Join* und *LeftJoin* zurückgegebenen Multimengen in eine Sequenz von Lösungen wird der Operator *ToList* eingeführt. Nach Überführung des gruppierenden Graph-Musters der *WHERE*-Klausel in einen Ausdruck der SPARQL-Algebra *G* wird dieser daher um den *ToList*-Operator erweitert: $G \coloneqq ToList(G)$. Anschließend können die aus Tabelle 3 bekannten Modifikatoren (inklusive der *SELECT*-Klausel) überführt werden. Im Folgenden ist jeweils angegeben, wie der Ausdruck *G* der SPARQL-Algebra ersetzt werden muss, wenn die SPARQL-Anfrage die auf der linken Seite dargestellte SPARQL-Anweisung erhält. Die Regeln sind in der angegebenen Reihenfolge auszuführen.

1. $ORDER\ BY\ s \implies G \coloneqq OrderBy(G, s)$, wobei s eine Liste von Sortieranweisungen ist.
2. $SELECT\ v_1\ v_2\ \dots v_n \implies G \coloneqq Project(G, \{v_1, v_2, \dots, v_n\})$ *oder*
   $SELECT * \implies G \coloneqq Project(G, \{v_1, v_2, \dots, v_n\})$, wobei $v_1, v_2, \dots, v_n$ alle in der Anfrage vorkommenden Variablen sind.
3. $DISTINCT \implies G \coloneqq Distinct(G)$
4. $REDUCED \implies G \coloneqq Reduced(G)$
5. $OFFSET\ o$ und $LIMIT\ l \implies G \coloneqq Slice(G, o, l)$ *oder*
   $OFFSET\ o \implies G \coloneqq Slice(G, o, s - o)$, wobei *s* die Anzahl der Ergebnisse der SPARQL-Anfrage für einen gegebenen RDF-Graphen ist. *Oder*
   $LIMIT\ l \implies G \coloneqq Slice(G, 0, l)$

Zusammenfassend ergeben sich die in Tabelle 4 dargestellten Operatoren der SPARQL-Algebra.

Graph-Pattern-Operatoren	**Ergebnismengen-Modifikatoren**
BGP	ToList
Join	OrderBy
LeftJoin	Project
Filter	Distinct
Union	Reduced
Graph	Slice

Tabelle 4: Die Operatoren der SPARQL-Algebra (entnommen aus [155]).

2.2.3 Evaluierungssemantik

Nachdem eine SPARQL-Anfrage in einen Ausdruck der SPARQL-Algebra überführt wurde, wird sie entsprechend der in [155] definierten Evaluierungssemantik gegen einen RDF-Graphen aufgelöst. Das Ergebnis ist eine Multimenge[10] von Lösungen in Form partieller Funktionen, den sogenannten *Lösungs-Mappings* (*solution mappings*). Es sei als Beispiel die

[10] Eine *Multimenge* kann im Gegensatz zu einer *Menge* dasselbe Element mehrfach enthalten.

bereits aus Listing 20 bekannte Anfrage bzw. der zugehörige Ausdruck in SPARQL-Algebra aus Listing 21 gegeben. Wenn dieser SPARQL-Algebra-Ausdruck, entsprechend der weiter unten erläuterten Evaluierungssemantik, gegen den in Listing 22 dargestellten RDF-Datensatz aufgelöst wird, ergeben sich zwei Lösungen, die durch die beiden Lösungs-Mappings μ_1 und μ_2 repräsentiert werden.

```
1. be:John_Doe be:Semester 7 ; be:Alter 27 ;
2.             be:Wohnort be:Amsterdam .
3. be:David_Mills be:Semester 9 ; be:Alter 31 ;
4.                be:Wohnort be:New_York .
```

Listing 22: Ein RDF-Datensatz.

Das Lösungs-Mapping μ_1 ist dabei durch $\mu_1(?\,student) = be{:}\,John\_Doe$ und $\mu_1(?\,semester) = 7$ definiert. Lösungs-Mapping μ_2 bildet beide Variablen folgendermaßen ab: $\mu_2(?\,student) = be{:}\,David\_Mills$ und $\mu_2(?\,semester) = 9$.

Zur formalen Definition von Lösungs-Mappings und Lösungen ist zunächst eine Formalisierung von RDF und SPARQL erforderlich. Die folgenden Definitionen entsprechen im Wesentlichen den entsprechenden Definitionen der SPARQL-Spezifikation [155]. Die Definitionen sind so oder in ähnlicher Form auch in zahlreicher, existierender Literatur zu finden (siehe z.B. [32, 84, 101, 148]). Sei I die Menge aller *IRIs*, L die Menge aller *Literale* und B die Menger aller *leerer Knoten*. Die Menge aller *RDF-Terme* T ist die Vereinigung dieser Mengen: $T = I \cup L \cup B$. Ein *RDF-Tripel* t (im Folgenden auch als *Tripel* bezeichnet) ist definiert als $t = (s, p, o) \in (I \cup B) \times I \times T$. Dabei bezeichnen s, p und o das *Subjekt*, *Prädikat* und *Objekt* des Tripels. Für ein spezifisches Tripel t werden im Folgenden mit $t.s$, $t.p$ und $t.o$ das Subjekt, Prädikat und Objekt dieses Tripels gekennzeichnet. Eine Menge von Tripeln bildet einen *RDF-Graphen* G (im Folgenden auch als *Graphen* bezeichnet): $G \subset (I \cup B) \times I \times T$. Seien $G, G_1, G_2, \dots, G_n$ Graphen und $u_1, u_2, \dots, u_n$ IRIs. Die Menge $D \coloneqq \{G, (< u_1 >, G_1), (< u_2 >, G_2), \dots, (< u_n >, G_n)\}$ bildet einen RDF-Datensatz (im Folgenden auch als *Datensatz* bezeichnet). Dabei ist G der *Standard-Graph* des Datensatzes und die Tupel $(< u_i >, G_i)$ für $i \in \{1, 2, \dots, n\}$ sind die *benannten Graphen*.

Sei V die Menge aller *Variablen*. Ein *Tripel-Pattern* tp ist definiert als $tp = (s, p, o) \in (T \cup V) \times (I \cup V) \times (T \cup V)$. Wie bei einem Tripel bezeichnen s, p und o das Subjekt, Prädikat und Objekt eines Tripel-Patterns. Für ein spezifisches Tripel-Pattern tp werden das Subjekt, Prädikat und Objekt durch $tp.s$, $tp.p$ und $tp.o$ gekennzeichnet. Ein *einfaches Graph-Muster* bgp setzt sich aus einem oder mehreren Tripel-Pattern zusammen: $bgp \subset (T \cup V) \times (I \cup V) \times (T \cup V)$.

Ein *Lösungs-Mapping* ist eine partielle Funktion $\mu: V \longrightarrow T$, die Variablen einer SPARQL-Anfrage auf RDF-Terme abbildet. Analog dazu bildet ein *RDF-Instanz-Mapping* die leeren Knoten eines Graph-Musters auf RDF-Terme ab: $\sigma: B \longrightarrow T$. Die Komposition eines Lösungs-Mappings μ und eines RDF-Instanz-Mappings σ wird als *Pattern-Instanz-Mapping* bezeichnet: $P(x) = \mu(\sigma(x))$. In Widerspruch zu den Funktionsdefinitionen, aber wie in der Literatur und in der SPARQL-Spezifikation üblich, wird für eine Menge von Tripel-Patterns tps mit $\mu(tps)$ und $\sigma(tps)$ die Menge an Tripel-Patterns beschrieben, die sich ergeben, wenn

alle in den Tripel-Pattern vorkommenden Variablen bzw. leere Knoten entsprechend der Funktion ersetzt werden. Das gleiche gilt mit Bezug auf Pattern-Instanz-Mappings.

Aufbauend auf dieser Definition wird im Folgenden die Auflösung (Evaluierung) der SPARQL-Algebra-Operatoren definiert. Der dafür verwendete Formalismus ist an die von Glimm [84] und Perez [148] verwendeten Formalismen angelehnt. Die Definitionen stimmen mit den entsprechenden Definitionen der SPARQL-Spezifikation [155] überein.

Einfaches Graph-Muster

Ein Lösungs-Mapping μ ist eine *Lösung* für ein einfaches Graph-Muster bgp mit Bezug auf einen RDF-Graphen G, wenn ein RDF-Instanz-Mapping σ existiert, so dass das Pattern-Instanz-Mapping $P = \mu \circ \sigma$ das einfache Graph-Muster auf einen Teilgraphen von G abbildet: $P(bgp) \subset G$.

Darauf aufbauend wird die Evaluierung eines einfachen Graph-Musters bgp gegen den Datensatz D mit dem aktiven Graph G folgendermaßen definiert:

$$[\![bgp]\!]_{D,G} = \dot{\{}\mu \mid \mu \; ist \; eine \; Lösung \; für \; bgp \; mit \; Bezug \; auf \; G\dot{\}}$$

Angelehnt an die von Glimm [84] verwendete Notation werden $\dot{\{}$ und $\dot{\}}$ verwendet, um zu kennzeichnen, dass es sich um eine Multimenge handelt. Die Vielfachheit von μ in der Multimenge aller Lösungen ergibt sich durch die Anzahl der verschiedenen RDF-Instanz-Mappings, welche als Komposition mit dem Lösungs-Mapping das einfache Graph-Muster auf einen Subgraphen des RDF-Graphen abbilden. Wenn n verschiedene RDF-Instanz-Mappings existieren $\sigma_1, \sigma_2, \ldots, \sigma_n$ so dass gilt $\mu(\sigma_i(bgp)) \subset G$ für alle $i \in \{1, 2, \ldots, n\}$, dann entspricht n der Vielfachheit von μ.

Union

Die Auswertung des *Union*-Operators für die beiden Graph-Pattern-Operatoren P_1 und P_2 ist definiert als die Vereinigung der Multimengen von Lösungen, die sich aus der Auswertung der beiden Graph-Pattern-Operatoren ergeben:

$$[\![Union(P_1, P_2)]\!]_{D,G} = [\![P_1]\!]_{D,G} \cup [\![P_2]\!]_{D,G}$$

Die Vielfachheit eines Lösungs-Mappings μ in der Ergebnis-Multimenge entspricht dabei der Summe der Vielfachheiten von μ in den Ergebnis-Multimengen der jeweiligen Graph-Pattern Operatoren.

Join

Die Auswertung des *Join*-Operators ist definiert als der Join der Multimengen von Lösungen, die sich aus der Auswertung der beiden Graph-Pattern-Operatoren ergeben:

$$[\![Join(P_1, \mathrm{P}_2)]\!]_{D,G} = [\![P_1]\!]_{D,G} \bowtie [\![P_2]\!]_{D,G}$$

Der Join zweier Multimengen von Lösungs-Mappings ist dabei als die Vereinigung aller kompatiblen Lösungs-Mappings definiert:

$$\Omega_1 \bowtie \Omega_2 = \dot{\{}\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1 \wedge \mu_2 \in \Omega_2 \wedge comp(\mu_1, \mu_2)\dot{\}}$$

Die Vielfachheit von μ in der Ergebnismenge entspricht dabei der Summe der Multiplikationen der Vielfachheiten aller $\mu_1 \in \Omega_1$ und $\mu_2 \in \Omega_2$ für die $\mu = \mu_1 \cup \mu_2$ gilt.
Zwei Lösungs-Mappings μ_1 und μ_2 sind *kompatibel*, wenn sie für jede Variable, welche in beiden Definitionsmengen enthalten ist, die gleichen Werte liefern:

$$comp(\mu_1, \mu_2) = \begin{cases} true, & \forall v \in dom(\mu_1) \cap dom(\mu_2) \longrightarrow \mu_1(v) = \mu_2(v) \\ false, & in\ allen\ anderen\ Fällen \end{cases}$$

Die Vereinigung zweier Lösungs-Mapping ist dabei folgendermaßen definiert:

$$\mu_1 \cup \mu_2(v) \coloneqq \begin{cases} \mu_1(v), & falls\ v \in dom(\mu_1) \\ \mu_2(v), & falls\ v \in dom(\mu_2) \end{cases}$$

Filter
Die Auflösung des *Filter*-Operators ist für den Filter-Ausdruck F definiert durch:

$$[\![Filter(F, P)]\!]_{D,G} = Filter(F, [\![P]\!]_{D,G})$$

Dabei wird für jedes Lösungs-Mapping μ geprüft, ob der Filter-Ausdruck F nach Abbildung der Variablen entsprechend des Lösungs-Mappings μ zu *true* ausgewertet werden kann. Wenn dies nicht der Fall ist, wird das Lösungs-Mapping nicht Bestandteil der Ergebnismenge.

$$Filter(F, \Omega) = \dot{\{}\mu \mid \mu \in \Omega \wedge [\![\mu(F)]\!] = true\dot{\}}$$

Die Vielfachheit eines Lösungs-Mappings μ in der Ergebnismenge entspricht dabei der Vielfachheit von μ in Ω.

LeftJoin
Der *LeftJoin*-Operator wird als linker Join über die Multimengen von Lösungen unter Berücksichtigung der Filter-Anweisung definiert:

$$[\![LeftJoin(P_1, P_2, F)]\!]_{D,G} = [\![P_1]\!]_{D,G} \underset{F}{\bowtie} [\![P_2]\!]_{D,G}$$

Das Ergebnis des linken Joins zweier Multimengen von Lösungen enthält die Vereinigung aller Lösungs-Mappings, die zueinander kompatibel sind und für die der Filter-Ausdruck zu *true* ausgewertet wird. Wenn für ein Lösungs-Mapping der ersten Multimenge kein kompatibles Lösungs-Mapping in der zweiten Multimenge enthalten ist oder wenn für alle

Vereinigungen die Filter-Bedingung zu *false* ausgewertet wird, dann ist dieses Lösungs-Mapping ebenfalls Bestandteil der Ergebnismenge.

$$\Omega_1 \underset{F}{\bowtie} \Omega_2 = \dot{\{}\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1 \wedge \mu_2 \in \Omega_2 \wedge comp(\mu_1, \mu_2) \wedge [\![\mu_1 \cup \mu_2(F)]\!] = true\dot{\}} \cup$$
$$\dot{\{}\mu_1 \mid \mu_1 \in \Omega_1, \forall \mu_2 \in \Omega_2 : ! comp(\mu_1, \mu_2) \vee [\![\mu_1 \cup \mu_2(F)]\!] = false\dot{\}}$$

Graph

Falls an den *Graph*-Operator eine IRI übergeben wird, dann müssen zwei Fälle unterschieden werden. Wenn der Datensatz D einen benannten Graph G_i mit dieser IRI enthält $(< IRI >, G_i) \in D$, dann gilt:

$$[\![Graph(IRI, P)]\!]_{D,G} = [\![P]\!]_{D,G_i}$$

Wenn der Datensatz D keinen benannten Graph G_i mit dieser IRI enthält $(< IRI >, G_i) \notin D$, dann ist die Ergebnismenge leer:

$$[\![Graph(IRI, P)]\!]_{D,G} = \emptyset$$

Wenn der Graph-Operator als Parameter eine Variable enthält, dann erfolgt eine Auflösung nach allen benannten Graphen:

$$[\![Graph(v, P)]\!]_{D,G} = \left([\![Graph(IRI_n, P)]\!]_{D,G} \bowtie \{\mu : v \longmapsto IRI_n\}\right) \cup$$
$$\left([\![Graph(IRI_{n-1}, P)]\!]_{D,G} \bowtie \{\mu : v \longmapsto IRI_{n-1}\}\right) \cup \ldots \cup$$
$$\left([\![Graph(IRI_1, P)]\!]_{D,G} \bowtie \{\mu : v \longmapsto IRI_1\}\right)$$

Wobei $IRI_1, IRI_2, \ldots, IRI_n$ den IRIs der benannten Graphen des Datensatzes D entsprechen. Die Vielfachheit eines spezifischen Lösungs-Mappings μ aus der Ergebnis-Multimenge entspricht dabei der Summe über die Vielfachheiten von μ in den Ergebnis-Multimengen der einzelnen Joins.

ToList

Der *ToList*-Operator wird ausgewertet, indem der übergebene Graph-Pattern-Operator gegen den Standard-Graphen G des Datensatzes D evaluiert wird:

$$[\![ToList(P)]\!]_D = ToList([\![P]\!]_{D,G})$$

Die sich ergebende Multimenge an Lösungs-Mappings wird in eine Sequenz von Lösungs-Mappings umgewandelt:

$$ToList(\Omega) = ToList\left(\dot{\{}\mu_1, \mu_2, \ldots, \mu_n\dot{\}}\right) \coloneqq (\mu_1, \mu_2, \ldots, \mu_n)$$

Distinct
Sei S ein Ausdruck in SPARQL-Algebra beginnend mit einem Ergebnismengen-Modifikator. Die Auflösung des *Distinct*-Operators für diesen Ausdruck ist definiert als:

$$[\![Distinct(S)]\!]_D = Distinct([\![S]\!]_D)$$

Für eine Sequenz von Lösungs-Mappings Ψ ergibt sich folgende Ergebnissequenz:

$$Distinct(\Psi) = Distinct((\mu_1, \mu_2, \dots, \mu_n)) \coloneqq (\mu_1, \mu_2, \dots, \mu_m)$$

Dabei ist $\{\mu_1, \mu_2, \dots, \mu_m\} = \{\mu \mid \mu \in \Psi\}$ wobei für $(\mu_1, \mu_2, \dots, \mu_m)$ die Ordnung in Ψ beibehalten wird.

Reduced
Der *Reduced*-Operator wird folgendermaßen aufgelöst:

$$[\![Reduced(S)]\!]_D = Reduced([\![S]\!]_D)$$

Für eine Sequenz von Lösungen ist das Ergebnis folgendermaßen definiert:

$$Reduced(\Psi) = Reduced((\mu_1, \mu_2, \dots, \mu_n)) \coloneqq (\mu_1, \mu_2, \dots, \mu_m)$$

Dabei ist $\dot{\{}\mu_1, \mu_2, \dots, \mu_m\dot{\}} \subseteq \dot{\{}\mu \mid \mu \in \Psi\dot{\}}$ wobei für $(\mu_1, \mu_2, \dots, \mu_m)$ die Ordnung in Ψ beibehalten wird.

OrderBy
Für die Ordnungsbedingungen $(c_1, c_2, \dots, c_m)$ ist die Evaluierung des der *OrderBy*-Operators definiert als:

$$[\![OrderBy(S, (c_1, c_2, \dots, c_m))]\!]_D = OrderBy([\![S]\!]_D, (c_1, c_2, \dots, c_m))$$
$$OrderBy(\Psi, (c_1, c_2, \dots, c_m)) \coloneqq (\mu_1, \mu_2, \dots, \mu_m)$$

Dabei ist $\dot{\{}\mu_1, \mu_2, \dots, \mu_m\dot{\}} = \dot{\{}\mu \mid \mu \in \Psi\dot{\}}$ wobei $(\mu_1, \mu_2, \dots, \mu_m)$ den Ordnungsbedingungen $(c_1, c_2, \dots, c_m)$ genügt.

Project
Für die Variablen $v_1, v_2, \dots, v_n$ ist die Auflösung des *Project*-Operators definiert als:

$$[\![Project(S, \{v_1, v_2, \dots, v_n\})]\!]_D = Project([\![S]\!]_D, \{v_1, v_2, \dots, v_n\})$$
$$Project(\Psi, \{v_1, v_2, \dots, v_n\}) = Project((\mu_1, \mu_2, \dots, \mu_m), \{v_1, v_2, \dots, v_n\}) \coloneqq (\mu_1', \mu_2', \dots, \mu_m')$$

Dabei gilt für alle $i \in \{1, 2, \dots, n\}$, dass μ_i und μ_i' kompatibel sind ($comp(\mu_i, \mu_i') = true$) und die Definitionsmenge von μ_i' auf die Variablen der Menge $\{v_1, v_2, \dots, v_n\}$ beschränkt sein muss: $dom(\mu_i') \subseteq \{v_1, v_2, \dots, v_n\} \subseteq dom(\mu_i)$.

Slice
Für den Start-Wert s und die Länge l wird der *Slice*-Operator folgendermaßen aufgelöst:

$$[\![Slice(S, s, l\})]\!]_D = Slice([\![S]\!]_D, s, l)$$
$$Slice(\Psi, s, l) = Slice((\mu_1, \mu_2, \dots, \mu_n), s, l) \coloneqq (\mu_s, \mu_{s+1}, \dots, \mu_l)$$

2.3 Open Data Protocol (OData)

Das *Open Data Protocol* (*OData*) ist ein auf den REST-Prinzipien [81] basierendes Protokoll zur Abfrage und Manipulation von Daten. Die OData-Spezifikation wurde von Microsoft unter der *Microsoft Open Specification Promise* veröffentlicht. Zu Beginn dieser Arbeit lag sie in Version 3 vor [3]. Da zu diesem Zeitpunkt sowohl SAP NetWeaver Gateway als auch viele öffentlich verfügbare OData-Schnittstellen Version 2 [137] verwendeten, wird im Rahmen dieser Arbeit auf diese Version Bezug genommen. OData wurde während der Erstellung dieser Arbeit laufend weiterentwickelt und 2014 von der OASIS in Version 4.0 als Standard anerkannt [150, 151, 152].

Wie bereits erwähnt, basiert OData auf den REST-Prinzipien bzw. stellt eine Möglichkeit dar, REST-basierte Web Services umzusetzen. Daher wird im Folgenden zunächst ein Überblick über REST bzw. REST-basierte Web Services[11] gegeben. REST (*Representational State Transfer*) ist ein Architekturstil, der als idealisiertes Modell die Grundlagen der modernen Web Architektur darstellt. Der Begriff wurde von Roy T. Fielding in [81] und [82] eingeführt und beschrieben. Web Services, welche die Prinzipien von REST umsetzen, werden als REST-basierte Web Services bezeichnet. REST-basierte Web Services charakterisieren sich im Wesentlichen durch folgende Eigenschaften [160, 161]:

Ressourcen-orientiert: REST-basierte Web Services ermöglichen die Abfrage und Manipulation von Ressourcen. Eine Ressource kann dabei jede Information sein, der ein Name zugewiesen werden kann [81]. Im Kontext von ERP-Systemen kann insbesondere jedes Geschäftsobjekt als Ressource angesehen werden. Jede Ressource wird über eine URI eindeutig identifiziert.

[11] REST-basierte Web Services werden auch als *RESTful Web Services* bezeichnet.

Einheitliche Schnittstelle: REST-basierte Web Services bieten für jede Ressource eine einheitliche Schnittstelle in Form von vier Operationen: Erzeugen, Abfragen, Aktualisieren und Löschen (CRUD[12]). Diese Operationen werden über die vier HTTP-Methoden POST, GET, PUT und DELETE realisiert. Dabei sind die Operationen GET, PUT und DELETE idempotent.

Zustandslosigkeit: REST-basierte Web Services sind zustandslos. Der Client muss bei jeder Anfrage alle zur Ausführung des Services benötigten Informationen an den Service übertragen.

Unterstützung verschiedener Repräsentationsformen: Der Service unterstützt die Rückgabe einer Ressource in verschiedenen Repräsentationen. Dabei kann es sich z.B. um unterschiedliche Darstellungsformen (z.B. Deutsch oder Englisch) oder um unterschiedliche Formate (z.B. XML oder JSON [66]) handeln. Die Realisierung kann z.B. mittels HTTP Content Negotiation erfolgen. Eine andere Möglichkeit besteht darin, für jede Repräsentation eine eindeutige URI einzuführen.

Verlinkung der Ressourcen: Ressourcen enthalten Links auf andere Ressourcen. Ein Geschäftsobjekt Bestellung kann z.B. einen Link auf ein anderes Geschäftsobjekt Kunde beinhalten, um anzugeben, dass die Bestellung von dem referenzierten Kunden aufgegeben wurde.

Web Services, welche diese Eigenschaften aufweisen, werden als REST-basiert bezeichnet. Es ist anzumerken, dass REST-basierte Web Services einen grundlegend anderen Ansatz verfolgen als SOAP-/WSDL-basierte Web Services, welche nicht Ressourcen-orientiert sind. Zu den Vor- und Nachteilen dieser beiden Ansätze wurde viel geschrieben (siehe z.B. [160]). Auf diese Diskussion soll hier nicht weiter eingegangen werden. Stattdessen sei angemerkt, dass im Gegensatz zu SOAP-/WSDL-basierten Web Services für REST-basierte Web Services unterschiedliche Technologien zu deren Realisierung existieren. Eine mögliche Technologie, mit der REST-basierte Web Services realisiert werden können, ist OData. OData selbst basiert auf einer Reihe bereits existierender Standards. Zur Datenübertragung verwendet OData das Atom Syndication Format (Atom) [145]. Atom ist ein XML-Format, das ursprünglich als Alternative zu RSS [163] für die Übertragung von Informationen von Webseiten, insbesondere von Blogs, entwickelt wurde. Ein Atom-Dokument, auch Feed genannt, setzt sich aus einer Menge von Elementen (*Entries*) und dazugehörigen Metadaten zusammen. Listing 23 zeigt ein Beispiel-Dokument, wie es von einem OData-Service erzeugt werden könnte. Die Angabe der Ressourcen bzw. der Geschäftsobjekte, die von dem OData-Service zurückgegeben werden, erfolgt in Form sogenannter *entry*-Elemente. Das Dokument aus Listing 23 zeigt ein *entry*-Element zur Übertragung der Informationen eines Equipments. Neben verschiedenen Angaben, wie z.B. der eindeutigen URI des Equipments (*id*) und der Referenz auf die mit dem Equipment verknüpften Dokumente (*link*), enthält der Eintrag ein

[12] CRUD = Create, Read, Update, Delete

content-Element, in das die eigentlichen Informationen des Equipments, z.B. Name des Herstellers (*ManufacturerName*), abgebildet werden (mehr dazu weiter unten).

```
1. <feed xmlns="http://www.w3.org/2005/Atom"
2. xmlns:d="http://schemas.microsoft.com/ado/2007/
3.          08/dataservices"
4. xmlns:m="http://schemas.microsoft.com/ado/2007/
5.          08/dataservices/metadata">
6.  <title>OData Test Collection</title>
7.  <link href="http://example.org/"/>
8.  <updated>20012-08-27T17:52:03Z</updated>
9.  <id>http://odata.test.service.org/test/Equipments</id>
10.
11. <entry>
12.   <id>http://odata.test.service.org/test/
13.    Equipments(12349126)</id>
14.   <link href="http://odata.test.service.org/
15.               test/Equipments(12349126)/Documents"/>
16.   <updated>2012-08-27T10:05:02Z</updated>
17.   <content type="application/xml">
18.    <m:properties>
19.     <d:EquipmentID m:type="Edm.Int32">
20.       12349126</D:EquipmentID>
21.     <d:ManufacturerName>XYZ</d:ManufacturerName>
22.     <d:SerialNr m:type="Edm.Int32">34762AC32</d:SerialNr>
23.     ...
24.    </m:properties>
25.   </content>
26. </entry>
27.  ...
28.</feed>
```

Listing 23: Ein Atom Feed Dokument.

Neben Atom, das zur Formatierung der Daten eingesetzt wird, basiert OData auf dem Atom Publishing Protocol (AtomPub) [87]. AtomPub setzt die REST-Prinzipien um, indem es für Web Ressourcen eine einheitliche Programmierschnittstelle in Form der CRUD-Operationen bereitstellt. Diese werden über die HTTP-Operationen (*GET*, *PUT*, *POST*, *DELETE*) realisiert. Des Weiteren ist in der AtomPub-Spezifikation eine Möglichkeit zur Beschreibung einer Service-Schnittstelle definiert. Die Beschreibung erfolgt in Form eines als *Service-Dokuments* bezeichneten XML-Dokuments. Listing 24 zeigt einen Ausschnitt eines Service-Dokuments. Ein Service-Dokument listet sogenannte Kollektionen (*Collections*) auf, die vom Service bereitgestellt werden. Dabei repräsentiert eine Kollektion eine Menge von Ressourcen, die über den Service abgerufen werden können. Die URI dieser Menge von

Ressourcen setzt sich aus der Basis-URI des Services und dem Wert des *href*-Attributes des *collection*-Elements zusammen. In dem dargestellten Beispiel stellt der Service eine Kollektion *Equipments* zur Verfügung, über die Daten über alle im System vorhandenen Equipments abgerufen werden können.

```
1. <service xml:base="http://odata.test.service.org/test/"
2.          xmlns="http://www.w3.org/2007/app">
3.  <workspace>
4.   <atom:title>Default</atom:title>
5.   <collection href="Equipments">
6.    <atom:title>Equipments</atom:title>
7.   </collection>
8.   …
9.  </workspace>
10.</service>
```

Listing 24: Ein Service-Dokument.

Bei dem gezeigten Service-Dokument fällt auf, dass eine Beschreibung der vom Service für ein Equipment zurückgegebenen Daten fehlt. Das Service-Dokument enthält keine Angaben darüber, welche Daten des Equipments als Teil des *properties*-Element (siehe Listing 23) übertragen werden. Um eine formale Beschreibung dieser Daten zu ermöglichen, erweitert OData AtomPub durch die Definition des *Entity Data Model for Data Services Packing Format* (*EDMX*) [139]. Eine EDMX-Datei beschreibt das dem OData-Service zugrundeliegende Datenmodell. Das Datenmodell selbst wird dabei mit der *Conceptual Schema Definition Language* (*CSDL*) [138] spezifiziert. Die CSDL-Spezifikation definiert dazu das Entitätsdatenmodell (*Entity Data Model* (*EDM*)). Dabei handelt es sich um eine Menge von Konzepten zur Definition von Datenstrukturen. Das Entitätsdatenmodell basiert auf dem Entity-Relationship-Modell. Listing 25 zeigt einen Ausschnitt aus einer an Listing 23 angelehnten EDMX-Datei. Eine EDMX-Datei wird in diesem Zusammenhang auch als *Service-Metadaten-Dokument* (*Service Metadata Document*) bezeichnet. Bei den Einträgen (*entry*-Elemente) aus einem Atom Feed Dokument handelt es sich um Elemente sogenannter *Entitätsmengen*, welche innerhalb eines *Service-Metadaten-Dokuments* über das *EntitySet*-Element spezifiziert werden. Die Elemente einer Entitätsmenge werden im Folgenden auch als *Entitäten* bezeichnet. Jede Entität hat einen Typ, den sogenannten *Entitätstypen*. Dieser entspricht entweder dem von der Entitätsmenge referenzierten Entitätstypen oder einem seiner Untertypen. Entitätstypen werden mittels des *EntityType*-Elements definiert. Ein Entitätstyp enthält eine Menge von *Eigenschaften*, die mittels des *Property*-Elements angegeben werden. Eigenschaften können primitiven oder komplexen Typs sein. Für die Definition primitiver Eigenschaften definiert das Entitätsdatenmodell ein abstraktes Typ-System, das gängige Datentypen, wie z.B. Boolean (*Edm.Boolean*), Integer (*Edm.Int32*), String (*Edm.String*) usw., spezifiziert. Komplexe Datentypen werden mit dem *ComplexType*-Element in dem Service-Metadaten-Dokument angegeben. Die Definition eines komplexen Typs unterscheidet sich von der Definition eines Entitätstyps dadurch, dass für den Entitätstyp mit dem *Key*-Element

ein eindeutiger Schlüssel für die Instanzen dieses Typs definiert wird. Dieser setzt sich dabei aus einem oder mehreren Eigenschaften des Entitätstyps zusammen.

```
1. <edmx:Edmx Version="1.0"
2.  xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx"
3.  xmlns:m="http://schemas.microsoft.com/ado/2007/08/
4.           dataservices/metadata">
5.  <edmx:DataServices m:DataServiceVersion="2.0">
6.   <Schema Namespace="BOO" xmlns="http://schemas.microsoft.
7.     com/ado/2008/09/edm">
8.    <EntityType Name="Equipment" m:HasStream="false">
9.     <Key>
10.     <PropertyRef Name="EquipmentID" />
11.    </Key>
12.    <Property Name="EquipmentID" Type="Edm.Int32"
13.              Nullable="false" />
14.    <Property Name="ManufacturerName" Type="Edm.String"
15.       MaxLength="50" Unicode="true" FixedLength="false" />
16.    <Property Name="SerialNr" Type="Edm.String"
17.       MaxLength="60" Unicode="true" FixedLength="false" />
18.     ...
19.    <NavigationProperty Name="Documents"
20.      Relationship="BOO.EquipmentDocuments"
21.      FromRole="Equipment" ToRole="Document" />
22.     ...
23.   </EntityType>
24.   ...
25.   <Association Name="EquipmentDocuments">
26.    <End Role="Equipment" Type="BOO.Equipment"
27.     Multiplicity="1" />
28.    <End Role="Documents" Type="BOO.Document"
29.     Multiplicity="*" />
30.   </Association>
31.   <EntityContainer Name="TestContainer"
32.    m:IsDefaultEntityContainer="true">
33.    <EntitySet Name="Equipments"
34.               EntityType="Boo.Equipments"/>
35.    ...
36.   </EntityContainer>
37.  </Schema>
38. </edmx:DataServices>
39.</edmx:Edmx>
```

Listing 25: Ausschnitt aus einem Service-Metadaten-Dokument.

Das Entitätsdatenmodell unterstützt das Konzept der Vererbung für Entitätstypen und komplexe Typen. Relationen zwischen Entitätstypen werden mittels *Assoziationen* und *Navigationseigenschaften* definiert. Eine Navigationseigenschaft spezifiziert eine Verknüpfung von einem Entitätstypen zu einem anderen Entitätstypen. Die Angabe erfolgt mit dem *NavigationProperty*-Element. Die eigentliche Definition der Relation erfolgt in Form einer Assoziation mit dem *Association*-Element außerhalb der Entitätstyp-Definition. Wobei das *NavigationProperty*-Element das zugehörige *Association*-Element referenziert. In dem dargestellten Beispiel wurde eine 1:n-Beziehung zwischen dem Entitätstyp *Equipment* und dem Entitätstyp *Document* definiert. Eine Instanz dieser Relation wird in einem Atom Feed Dokument durch das Element *link* repräsentiert (siehe Listing 23).

Es sei darauf hingewiesen, dass OData, neben der Realisierung der REST-Prinzipien auf Basis von AtomPub, auch die Bereitstellung von Funktionen im Stil von SOAP-/WSDL-basierten Web Services ermöglicht. CSDL stellt zu deren Beschreibung entsprechende Elemente zur Verfügung. Neben diesen und den bereits vorgestellten Elementen definiert CSDL eine Reihe weiterer Elemente, die hier nicht weiter erläutert werden sollen, da sie für das Verständnis dieser Arbeit nicht von Bedeutung sind.

Neben dem Service-Metadaten-Dokument erweitert OData AtomPub um eine Abfragesprache. Mit dieser Abfragesprache besteht die Möglichkeit, die Rückgabemenge eines Services bei der Abfrage einzuschränken. Im Folgenden wird diese Abfragesprache erläutert. Die URI zum Aufruf eines OData-Services setzt sich aus drei Komponenten zusammen. Der Service-Basis-URI, dem Ressourcen-Pfad und den Abfrageoptionen.

```
1. http://odata.test.service.org/test/
2.         Equipments(12349126)?$select=ManufacturerName
```

Listing 26: Eine URI zur Abfrage des Herstellernamens eines bestimmten Equipments.

Listing 26 zeigt ein Beispiel für eine OData-URI. Wenn diese URI per HTTP *GET* an den entsprechenden OData-Service übertragen wird, gibt dieser Service ausschließlich den Herstellernamen (*ManufacturerName*) des entsprechenden Equipments (*Equipments(12349126)*) zurück. Die URI setzt sich wie folgt zusammen:

- *http://odata.test.service.org/test/*: Service-Basis-URI
- *Equipments(12349126)*: Ressourcen-Pfad
- *$select=ManufacturerName*: Abfrageoptionen

Über die Service-Basis-URI kann das Service-Dokument des OData-Services abgerufen werden. Sie bildet außerdem die Basis jeder Service-Anfrage. Nach der Service-Basis-URI folgt der Ressourcen-Pfad, der die angefragte Ressource eindeutig identifiziert. In der dargestellten Beispiel-URI wird das Equipment mit der ID *12349126* angefragt. Auf den Ressourcen-Pfad folgen die Abfrageoptionen. In diesem Fall wurde mit der *select*-Option die zurückzugebende Datenmenge auf den Herstellernamen eingeschränkt. OData definiert eine Vielzahl von Abfrageoptionen, die im Folgenden erläutert werden sollen.

select: Mit der bereits bekannten *select*-Option kann die Rückgabemenge auf bestimmte Eigenschaften (Properties) eingeschränkt werden.

filter: Mit der *filter*-Anweisung kann die Menge der durch den Ressourcen-Pfad identifizierten Ressourcen auf diejenigen eingeschränkt werden, welche die angegebene Filterbedingung erfüllen. Listing 27 zeigt als Beispiel eine Anfrage, die alle Equipments zurückgibt, die vom Hersteller *XYZ* stammen.

```
1. http://odata.test.service.org/test/
2.         Equipments?$filter=ManufacturerName eq 'XYZ'
```

Listing 27: Die OData *filter*-Anweisung.

Neben dem dargestellten Gleichheits-Operator (*eq*) definiert OData eine Vielzahl weiterer logischer und arithmetischer Operatoren. Dazu gehören z.B. *And*, *Or*, *Not*, *Gt* (greater than), *Lt* (less than), *Add* (Addition), *Sub* (Subtraktion), *Mul* (Multiplikation) usw. Des Weiteren stehen eine Vielzahl von Funktionen zur Verfügung, die im Rahmen der *filter*-Anweisung verwendet werden können. Dazu gehören z.B. String-Funktionen (*substringof*, *length*, *indexof* usw.), mathematische Funktionen (*round*, *floor*, *ceiling* usw.) und Typ-Funktionen (*IsOf*).

format: Die *format*-Anweisung ermöglicht die Angabe des gewünschten Rückgabeformats. Mögliche Werte sind *json* (für JSON [66]) und *atom* (für das weiter oben beschriebene Atom Format). Des Weiteren ist als Wert die Angabe jedes möglichen MIME-Typs [100] erlaubt.

orderby: Ermöglicht die Sortierung einer Menge von Einträgen nach einer bestimmten Eigenschaft.

top: Mit der *top*-Anweisung kann die Rückgabe auf die ersten *n* Einträge beschränkt werden.

skip: Die *skip*-Anweisung ermöglicht bei der Rückgabe die ersten *n* Einträge zu überspringen.

expand: Die *expand*-Anweisung ermöglicht es, miteinander verlinkte Einträge über eine einzige Service-Anfrage abzurufen. Somit ist es beispielsweise möglich, ein Equipment und die dazugehörigen Dokumente mit einem Service-Aufruf abzufragen.

inlinecount: Bei Angabe der *inlinecount*-Anweisung wird als Teil der Rückgabe die Anzahl der zurückgegebenen Einträge angegeben.

Es sei noch darauf hingewiesen, dass OData neben den vorgestellten Abfrageoptionen auch die Übergabe von Service-spezifischen Abfrageoptionen ermöglicht (*custom query options*). Des Weiteren wird der URI-Abschnitt zur Angabe der Abfrageoptionen auch zur Übergabe von Parametern für OData-Services genutzt.

Die Semantik von HTTP *PUT*, wie sie in [87] definiert ist, entspricht eher einem *Ersetzen* als einem *Update*, da alle Werte eines Eintrags, die vom Client nicht übertragen werden, durch die Standard-Werte ersetzt werden. OData erweitert daher HTTP um eine *MERGE*-Anweisung, die ein inkrementelles Update ermöglicht. Alle Werte eines Eintrags, die nicht Teil einer MERGE-Anweisung sind, bleiben unverändert. Um eine ungehinderte Weiterleitung von HTTP *MERGE* im Netzwerk zu ermöglichen, definiert OData einen zusätzlichen HTTP-Header (*XHTTPMethod*), der das Tunneln einer *MERGE*-Anweisung über HTTP *POST* ermöglicht. Es sei darauf hingewiesen, dass in OData 3.0 [3] HTTP *MERGE* durch das mit [75] eingeführte HTTP *PATCH* ersetzt wurde. Aus den bereits genannten Gründen wird in dieser Arbeit OData 2.0 betrachtet, daher wird im Folgenden HTTP *PATCH* nicht weiter berücksichtigt.

Zur Steigerung der Performance erlaubt OData das Ausführen mehrerer Operationen über eine einzige HTTP-Anfrage. Diese Anfragen werden als *Batch-Anfragen* bezeichnet. Dabei werden mehrere *GET*-, *PUT*-, *POST*-, *DELETE*-Operationen in einer einzigen *POST*-Anfrage zusammengefasst. Über sogenannte *ChangeSets* besteht außerdem die Möglichkeit, mehrere Operationen innerhalb einer Batch-Anfrage logisch zu gruppieren.

3 Verwandte Arbeiten

Das Ziel dieser Arbeit besteht darin, ein Verfahren für die Integration von OData-Services in das Semantic Web [44] zu entwickeln. Dazu wird ein Verfahren zur Realisierung von SPARQL-Endpunkten auf Basis von OData-Schnittstellen hergeleitet. Das Verfahren soll dabei insbesondere mit Hinblick auf die Vereinfachung der Abfrage von ERP-Systemen konzipiert werden. Dabei müssen sowohl die Technologien als auch die Anforderungen, die sich durch diesen spezifischen Anwendungsfall ergeben, berücksichtigt werden. Es ergeben sich dabei eine Vielzahl unterschiedlicher Herausforderungen, die verschiedene Forschungsbereiche berühren, in denen teilweise bereits umfangreiche Forschungsarbeiten existieren. Diese Arbeit tangiert hauptsächlich folgende, komplementäre Forschungsbereiche:

- **Semantische Beschreibung von Web Services (Semantic Web Services)**
  Die semantische Beschreibung der OData-Service-Schnittstellen ist eine Voraussetzung für deren Abfrage mittels SPARQL. Es existieren bereits zahlreiche Arbeiten zur semantischen Beschreibung von Service-Schnittstellen.
- **Integration von Web Services in das Semantic Web**
  Basierend auf Technologien zur semantischen Beschreibung von Web Services existieren bereits zahlreiche Verfahren zur Integration von Web Services in das Semantic Web bzw. zu deren Abfrage mittels SPARQL.

Zunächst werden in Abschnitt 3.1 verwandte Arbeiten bzgl. der semantischen Beschreibung von Web Services betrachtet. Anschließend erfolgt in Abschnitt 3.1 ein Vergleich der verwandten Arbeiten, welche die Abfrage von Web Services mittels semantischen Technologien behandeln.

3.1 Semantische Beschreibung von Web Services

Aufbauend auf der von Tim-Berners Lee ausgerufenen Semantic Web Initiative [44] kam die Idee auf, die als Folge dieser Initiative entwickelten Konzepte und Technologien auch zur semantischen Beschreibung von Web Services zu verwenden. Bereits 2001 stellten McIlraith et al. [133] die Vision von semantisch beschriebenen Web Services vor, die ein automatisches Auffinden, Ausführen und eine automatische Komposition der Web Services erlauben. Ausgehend von dieser Vision, wurden erste Ansätze zur ihrer Realisierung entwickelt (siehe z.B. [53, 78]). Seit dieser Zeit sind eine Vielzahl von Forschungsarbeiten in diesem Bereich durchgeführt worden. Zahlreiche Verfahren wurden entwickelt, die sich hinsichtlich der Komplexität, der Ausdrucksstärke und der Zielsetzung unterscheiden. Mit dem Aufkommen SOAP-/WSDL-basierter Web Services Anfang des neuen Millenniums zielten viele Vorhaben auf die semantische Annotation von mit WSDL beschriebenen Web Services ab. Die steigende Popularität von REST-basierten Services hat in den letzten Jahren zur Entwicklung von Ansätzen zur semantischen Beschreibung geführt, die an die auf den REST-Prinzipien aufsetzenden Web Services angepasst sind.

Die in diesem Abschnitt betrachteten Ansätze wurden zunächst unabhängig von dem Ziel entwickelt SPARQL-Endpunkte für Service-Schnittstellen zu realisieren. Sie können jedoch dafür verwendet werden. Im Zusammenhang mit der Betrachtung von Verfahren zum Zugriff auf Web Services mittels semantischer Technologien in Abschnitt 3.2 werden auch Technologien zur semantischen Beschreibung von Web Service Schnittstellen vorgestellt, die speziell mit Hinblick auf dieses Ziel entwickelt wurden. Zunächst werden in diesem Abschnitt verschiedene Ansätze zur Beschreibung von SOAP-/WSDL-basierten Web Services erläutert und dahingehend evaluiert, inwieweit sie zur Beschreibung von OData-basierten Services mit Hinblick auf die Realisierung von SPARQL-Endpunkten verwendet werden können. Anschließend erfolgt mit der gleichen Zielsetzung die Betrachtung der hauptsächlich in den letzten Jahren entstandenen Ansätze zur Beschreibung von REST-basierten Services. Durch einen Vergleich verschiedener Studien zu Semantic Web Services [55, 120, 141] hat sich ergeben, dass SAWSDL [122], OWL-S [132] und WSMO [162] zu den populärsten Technologien zur semantischen Beschreibung von Web Service Schnittstellen zählen. Der Vergleich der verwandten Arbeiten beginnt daher mit der Betrachtung dieser Technologien.

3.1.1 SAWSDL

SAWSDL (*Semantic Annotations for WSDL and XML Schema*) ist eine Empfehlung des W3C zur semantischen Annotation von WSDL und XML-Schema [122]. Es wurde primär mit Hinblick auf WSDL 2.0 [59, 123] entwickelt. Über spezielle Elemente wird aber auch die semantische Erweiterung von WSDL 1.1 Dokumenten [60] unterstützt. SAWSDL definiert verschiedene Attribute und Elemente als Erweiterung von WSDL und XML Schema, die zur semantischen Annotation von Interfaces, Operationen und den Ein- und Ausgabeparametern verwendet werden können. Dabei macht SAWSDL keine Vorgaben hinsichtlich der zu verwendenden Ontologie-Beschreibungssprache.

Die Erweiterung von WSDL besteht aus den drei folgenden, im SAWSDL-Namensraum[13] definierten, Attributen: *modelReference*, *liftingSchemaMapping*, *loweringSchemaMapping*. Das *modelReference*-Attribut ermöglicht die semantische Annotation von Interfaces (*wsdl:interface*), Operationen (*wsdl:operation*) und Fehlern (*wsdl:fault*). Des Weiteren wird die semantische Erweiterung von Elementen (*xsd:element*), Attributen (*xsd:attribute*), einfachen (*xsd:simpleType*) und komplexen Typen (*xsd:complexType*) in XSD unterstützt. Dazu erhält das *modelReference*-Attribut als Wert eine Liste einer oder mehrerer URIs, die auf Konzepte einer oder mehrerer Ontologien verweisen. Die SAWSDL-Spezifikation macht keine genauen Vorgaben wie die semantische Annotation eines Elements zu interpretieren ist. Zum Beispiel könnte die semantische Erweiterung einer Operation als abstrakte Beschreibung zur Kategorisierung dieser Operation interpretiert werden, sie könnte aber auch deren Verhalten beschreiben [122]. Die beiden Attribute *liftingSchemaMapping* und *loweringSchemaMapping* dienen zur Referenzierung von Mapping-Beschreibungen. Beide Attribute sind nur für Typ- und Element-Definition in XSD definiert. Das *loweringSchemaMapping*-Attribut referenziert eine Mapping-Beschreibung, die

[13] http://www.w3.org/ns/sawsdl

spezifiziert, wie die semantischen Daten in das vom Web Service erwartete XML-Format überführt werden. Als Gegenstück dazu verweist das *liftingSchemaMapping*-Attribut auf eine Beschreibung, in der das Mapping der vom Web Service zurückgegebenen XML-Daten in das semantische Modell beschrieben ist. Die Mapping-Beschreibungen können z.B. mit XSLT [61], SPARQL [155] und XQuery [57] erstellt werden. SAWSDL macht diesbezüglich aber keine Vorgaben.

Obwohl SAWSDL als Erweiterung von WSDL konzipiert wurde, stellt sich die Frage, ob die eingeführten Konzepte für CSDL mit Hinblick auf die Bereitstellung eines SPARQL-Endpunktes übernommen werden können. Prinzipiell könnte jede mögliche Abbildung eines Entitätsdatenmodells auf RDF-Graphen mit XSLT beschrieben werden. Diese XSLT-Dokumente könnten dann über das *liftingSchemaMapping*-Attribut aus dem CSDL-Dokument heraus referenziert werden. Dieser Ansatz hätte allerdings verschiedene Nachteile. Zum einen wäre die Erstellung der entsprechenden XSLT-Dokumente sehr aufwändig. Ohne entsprechende Werkzeuge, die noch zu entwickeln wären, wäre die manuelle Erstellung der XSLT-Beschreibungen wahrscheinlich nicht in akzeptabler Zeit durchzuführen. Zum anderen verkompliziert diese Art der Beschreibung die Entwicklung entsprechender Algorithmen zur Auflösung der SPARQL-Anfragen gegen die OData-Services. Da der vom Service bereitgestellte RDF-Graph durch das XSLT-Dokument beschrieben ist, müsste der Algorithmus durch Analyse der XSLT-Dokumente die zur Beantwortung einer SPARQL-Anfrage benötigten Services ermitteln können. Außerdem wäre der sich ergebende Algorithmus wahrscheinlich weniger performant als bei einer speziell mit Hinblick auf die Beantwortung von SPARQL-Anfragen entwickelten semantischen Beschreibung, bei der keine XSLT-Analyse erforderlich ist. Selbst wenn anstatt XSLT eine andere Sprache zur Beschreibung der Abbildung verwendet wird, müssen immer zusätzliche Dokumente verwaltet und analysiert werden. Ein anderer Ansatz bestände darin, eine semantische Abbildung über das *modelReference*-Attribut ohne die Verwendung von XSLT-Beschreibungen zu definieren. Zwar könnten damit einfache Abbildungen (z.B. wie in Abschnitt 4.4 beschrieben) realisiert werden. Komplexere Abbildungen, wie z.B. die Abbildung eines Entitätstyps in Abhängigkeit eines anderen Entitätstyps, können damit nicht oder nur sehr umständlich beschrieben werden.

3.1.2 OWL-S

OWL-S (Web Ontology Language for Web Services) [132] ist eine OWL-basierte Ontologie zur Beschreibung von Web Services. Diese Ontologie besteht hauptsächlich aus drei Teilen:

- **Service Profile:** Das Service Profil dient dazu, den Service mit Hinblick auf das Suchen und Auffinden von Services zu beschreiben. Es enthält menschenlesbare Angaben, wie den Namen des Services, eine textuelle Beschreibung des Services und Kontaktinformationen des Service-Verantwortlichen. Des Weiteren werden funktionale Eigenschaften, wie die Ein- und Ausgabeparameter, Vorbedingungen und Nebeneffekte (IOPE), beschrieben. Neben den funktionalen Eigenschaften können auch Angaben zur Qualität des Services gemacht werden. Außerdem besteht die Möglichkeit, Informationen zur Klassifizierung anzugeben. Das Service Profil kann

auch zur Beschreibung der Anfrage verwendet werden. In diesem Fall wird über das Profil der gesuchte Service spezifiziert.

- **Service Model:** Das Service Model spezifiziert, wie der Client mit dem Service interagieren kann. Dazu gehören auch Angaben zu den Ein-, Ausgabeparametern, Vorbedingungen und Nebeneffekten. Wobei sich diese von den im Service-Profil beschriebenen unterscheiden können. Die möglichen Interaktionsreihenfolgen sind als Prozesse beschrieben. Es wird zwischen atomaren, zusammengesetzten und einfachen Prozessen unterschieden.
- **Service Grounding:** Das Service-Grounding stellt die zum Aufruf eines Services notwendigen Angaben bereit. Insbesondere werden Informationen zum Format der vom Service erwarteten und zurückgegebenen Nachrichten, dem verwendeten Protokoll und der Adressierung gegeben. Dazu werden insbesondere die Nachrichten und die Operationen zu den entsprechenden Instanzen aus dem Service-Model in Beziehung gesetzt. Aufgrund der weiten Verbreitung von WSDL bezieht sich die OWL-S-Spezifikation auf die Verwendung dieser Beschreibungssprache. Wobei darauf hingewiesen wird, dass neben WSDL auch andere Beschreibungssprachen verwendet werden können.

Obwohl es möglich ist, einen OData-Service mit OWL-S zu beschreiben und ein entsprechendes Grounding zu definieren, wäre dieser Ansatz für die Realisierung eines SPARQL-Endpunktes überdimensioniert. Viele der Konzepte, wie Vorbedingungen, Nebeneffekte, die Beschreibung von Prozessen usw., werden nicht benötigt. Des Weiteren ist die Erstellung einer entsprechenden OWL-S-Beschreibung weit aufwendiger als der in dieser Arbeit vorgestellte Ansatz. Außerdem müsste die Abbildung über ein XSLT-Dokument beschrieben werden, da die semantische Annotation mit einzelnen Konzepten nicht ausreichend ist, um alle möglichen Abbildungen beschreiben zu können. Dieser Ansatz würde allerdings wieder zu den bereits in Abschnitt 3.1.1 beschriebenen Problemen führen.

3.1.3 WSMO

Die Web Service Modeling Ontology (WSMO) [72, 162] ist ein Framework zu semantischen Beschreibung von Web Services. Es basiert auf dem Web Service Modeling Framework (WSMF) [77, 78] und erweitert dieses um zusätzliche Konzepte. WSMO setzt sich im Wesentlichen aus folgenden Elementen zusammen [176]:

- **Ontologien:** Die den Web Services zugrundeliegende Domäne wird in Form von Ontologien formalisiert. Dabei kommt in WSMO die Web Service Modeling Language (WSML) [73] zum Einsatz. Bei WSML handelt es sich um eine Ontologiebeschreibungssprache, die speziell für diesen Zweck entwickelt wurde.
- **Web Service Beschreibungen:** Die eigentliche Beschreibung des Web Services umfasst die Beschreibung der funktionalen (*capability*) und nicht-funktionalen Eigenschaften, der Choreographie und der Orchestrierung. Die Beschreibung der

nicht-funktionalen Eigenschaften enthält z.B. Angaben über die Kosten der Benutzung, der Performance, der Skalierbarkeit usw. Die funktionale Beschreibung enthält Informationen zu den Vorbedingungen (*preconditions*), z.B. Eingabeparameter, dem Zustand, in dem sich die Welt befinden muss (*assumptions*), den Nachbedingungen (*postconditions*) und den Nebeneffekten (*effects*). Die Choreographie beschreibt, wie der Web Service vom Client verwendet werden kann. Die Orchestrierung hingegen gibt an, welche anderen Web Services von dem beschriebenen Web Service aufgerufen werden.

- **Ziele (Goals):** Die Ziele werden vom Client definiert. Sie geben an, welche Anforderungen der Client an den gesuchten Service hat. Ziele bilden das Gegenstück zu der Web Service Beschreibung. Sie können Angaben über die gewünschten funktionalen und nicht-funktionalen Eigenschaften enthalten. Des Weiteren können auch Anforderungen bzgl. der erwarteten Kommunikationsmuster (Choreographie) und der Services, die der gesuchte Service selbst im Rahmen der Abarbeitung verwenden darf (Orchestrierung), definiert werden.
- **Mediatoren:** Mediatoren dienen dazu, Unterschiede bei den verwendeten Datenformaten und Protokollen zu überbrücken.

Mit der Web Service Execution Environment (WSMX) [54] existiert eine Ausführungsumgebung für mit WSMO beschriebene Web Services.

Für die Beschreibung von OData-basierten Web Services zur Realisierung eines SPARQL-Endpunktes gelten für WSMO im Wesentlichen die gleichen Kritikpunkte wie sie bereits für OWL-S in Abschnitt 3.1.2 erläutert wurden. Die Beschreibungen sind für den gedachten Einsatzzweck überdimensioniert. Viele der Konzepte, wie z.B. Vorbedingungen, Nebeneffekte usw., werden nicht benötigt. Da WSMO mit der Web Service Modeling Language eine eigene Sprache zur Beschreibung verwendet, ist der Einarbeitungsaufwand außerdem wesentlich höher als bei dem in dieser Arbeit vorgestellten leichtgewichtigeren Ansatz, der zur Beschreibung auf existierenden Standards aufsetzt.

Der Vollständigkeit halber sei angemerkt, dass neben WSMO mit WSMO-Lite [79, 109, 187] auch eine vereinfachte Variante existiert, die auf W3C-Standards, wie z.B. SAWSDL, RDFS und OWL, aufsetzt.

3.1.4 hREST und MicroWSMO

Webseiten bzw. HTML-Dokumente können üblicherweise von Menschen leicht gelesen werden. Die algorithmische Verarbeitung hingegen ist schwierig, da die automatische Bestimmung der Semantik und der Struktur der in einer HTML-Seite dargestellten Daten aktuell nur sehr eingeschränkt möglich ist. Mikroformate [103] wurden geschaffen, um dieses Problem zu beheben. Sie erlauben die Angabe zusätzlicher Informationen über die in einer HTML-Seite enthaltenen Daten. Mikroformate definieren eine Menge von Konzepten, die aus einem HTML-Dokument heraus referenziert werden können. Die Referenzierung erfolgt dabei über HTML-Bordmittel, z.B. über das häufig im Zusammenhang mit CSS verwendete *class*-Attribut. Es existieren Mikroformate für die unterschiedlichsten Domänen,

beispielsweise zur Kennzeichnung von Ereignissen (*hCalender*), Menschen und Namen (*hCard*).

Für REST-basierte Web Services existiert mit der *Web Application Description Language* (WADL) [89] eine Sprache zur formalen, syntaktischen Beschreibung. Des Weiteren besteht auch mit WSDL 2.0 [123] die Möglichkeit, REST-basierte Services zu beschreiben. In der Praxis wird von diesen Möglichkeiten wenig Gebrauch gemacht. Stattdessen werden REST-basierte Web Services häufig umgangssprachlich in Form von HTML-Dokumenten beschrieben [110]. Um solche in HTML-Dokumenten vorhandenen Informationen über einen REST-basierten Web Service der algorithmischen Verarbeitung zugänglich zu machen, wurde mit *hREST* [110] ein Mikroformat für die Beschreibung von REST-basierten Services entwickelt. hRest stellt eine Menge von Konzepten, wie z.B. *service*, *operation*, *input*, *output* usw., zur Verfügung, mit der die entsprechenden Bereiche im HTML-Dokument gekennzeichnet werden können. Aufbauend auf mit hREST annotierten HTML-Dokumenten kann, z.B. unter Verwendung von XSLT und GRDDL [63], eine Abbildung auf RDF realisiert werden. Mit *MicroWSMO* [111] besteht die Möglichkeit, HTML-Service-Beschreibungen, um semantische Annotationen im Sinne von SAWSDL (siehe Abschnitt 3.1.1) zu erweitern. Ausgehend von SAWSDL erweitert MicroWSMO hREST um die drei Konzepte *model*, *lifting* und *lowering*, die den aus SAWSDL bekannten Attributen *modelReference*, *liftingSchemaMapping* und *loweringSchemaMapping* entsprechen. Dabei ist MicroWSMO insbesondere für die Verwendung mit der WSMO-Lite Ontologie gedacht [111]. Mit SWEET [129] existiert ein Tool, das den Benutzer bei der Erweiterung von Service-Beschreibungen mittels hREST und MicroWSMO unterstützt.

Da OData-Services über die zugehörigen Service-Dokumente und Service-Metadata-Dokumente formal beschrieben sind, besteht keine Notwendigkeit, zugehörige textuelle Beschreibungen, sofern sie überhaupt vorhanden sind, mittels hREST zu annotieren. Hinsichtlich der Verwendung von MicroWSMO zur Beschreibung der Abbildung würden sich dieselben Probleme wie bei der Verwendung von SAWSDL ergeben (siehe Abschnitt 3.1.1).

3.1.5 RDFa

Mit RDFa [22] bzw. mit RDFa-Lite [174], als vereinfachte Untermenge von RDFa, können Abbildungen von mittels Auszeichnungssprachen (insbesondere HTML) strukturierter Daten auf RDF beschrieben werden. Dazu definiert RDFa eine Menge von Attributen, die z.B. als Erweiterung in ein HTML-Dokument eingefügt werden können. Das Attribut *resource* erlaubt die Spezifikation der URI der Ressource, welche das Subjekt der nachfolgenden Aussagen bilden soll. Fehlt das *resource*-Attribut, erfolgt die Abbildung auf einen leeren Knoten. Mit dem *typeof*-Attribut kann der Ressource ein Typ zugewiesen werden. Aussagen werden über das Attribut *property* spezifiziert, welches als Wert das Prädikat der Aussage erhält. Das Objekt der Aussage wird aus dem Inhalt des Elements gebildet, das um das *property*-Attribut erweitert wurde. Wenn das Element ein *href*- oder ein *src*-Attribut enthält, bildet der Wert dieses Attributs das Objekt der Aussage. Des Weiteren besteht über das *content*-Attribut die Möglichkeit, den Wert der Aussage explizit anzugeben. Für das Objekt einer Aussage kann mit dem *datatype*-Attribut der Datentyp spezifiziert werden. Weitere Attribute existieren zur

Festlegung der Namensräume (*vocab*, *prefix*) und zur Abbildung auf bestimmte RDF-Konstrukte, wie z.B. Listen (*inlist*).

Obwohl RDFa speziell für Auszeichnungssprachen entwickelt wurde, ist der zugrundeliegende Ansatz ein ähnlicher wie der in dieser Arbeit vorgestellte. Die Abbildung wird über zusätzliche Attribute als Teil des Dokuments spezifiziert. Da aber RDFa insbesondere mit Hinblick auf HTML entwickelt wurde, können viele Konstrukte, wie sie mit CSDL beschrieben werden können (z.B. Typhierarchien, Assoziationen usw.), nicht in die Abbildung mit einbezogen werden. Des Weiteren geht der in dieser Arbeit vorgestellte Ansatz weit über die einfachen, mit RDFa beschreibbaren Abbildungen hinaus, z.B. hinsichtlich der Einbeziehung von XPath-Funktionen in die Abbildung und der Konzepte zur automatischen Generierung der Ressourcen-URIs.

3.1.6 SA-REST

Aus der Motivation heraus, die Erstellung von Mashups zu vereinfachen, stellen Lathem, Gomadam und Sheth in [121] und [170] mit *SA-REST* ein Framework zur semantischen Beschreibung REST-basierter Services vor. SA-REST ist vom Ansatz her mit dem in Abschnitt 3.1.4 beschriebenen MicroWSMO vergleichbar. Die HTML-Beschreibungen werden mittels RDFa semantisch erweitert. Wobei SA-REST Prädikate zur Annotation der Eingabeparameter (*sarest:input*), der Ausgabeparameter (*sarest:output*), der Fehler (*sarest:Fault*) und der Operationen (*sarest:operation*) definiert. Ähnlich wie das aus SAWSDL bzw. MicroWSMO bekannte *modelReference*-Attribut referenzieren Aussagen mit diesen Prädikaten über das Objekt die entsprechenden Konzepte der Ontologie. Die URL, über die der Service aufgerufen werden kann, bildet dabei die Subjekte der Aussagen. Des Weiteren werden Prädikate zur Referenzierung des Lowering- (*sarest:lowering*) und des Lifting-Schemas (*sarest:lifting*) bereitgestellt, die von der Bedeutung ebenfalls mit den entsprechenden Attributen aus SAWSDL bzw. MicroWSMO vergleichbar sind. Neben der Verwendung von RDFa wird auch die Annotation mittels beliebiger Mikroformate in Kombination mit GRDDL [63] unterstützt. Wobei dies dahingehend definiert ist, dass die sich durch die GRDDL-Transformation ergebenden RDF-Tripel mit den sich aus einer äquivalenten RDFa Annotation ergebenden Tripel übereinstimmen müssen. Da SAREST von den grundlegenden Konzepten im Wesentlichen mit SAWSDL bzw. MicroWSMO übereinstimmt, ergeben sich auch die gleichen, bereits beschriebenen, Probleme.

3.1.7 ReLL

Alarcon und Wilde leiten in [25] und [27], aufbauend auf einem Metamodell zur Beschreibung REST-basierter Services, eine Sprache, *ReLL* (*Resource Linking Language*), zu deren Beschreibung her. Mit *RESTler* [26] stellen die Autoren einen Crawler vor, der die von einem mit ReLL beschriebenen REST-Service bereitgestellten Ressourcen abrufen und mittels einer zusätzlichen Komponente nach RDF überführen kann.

Das Metamodell formalisiert einen Service als eine Menge von Ressourcen. Eine Ressource hat verschiedene Eigenschaften, wie eine eindeutige ID, einen Namen, eine Beschreibung und ein URI-Pattern. Das URI-Pattern spezifiziert die Struktur der URI über die

auf die einzelnen Ressourcen zugegriffen werden kann. Eine Ressource kann auf eine oder mehrere Repräsentationen verweisen, welche die möglichen Serialisierungsformate (z.B. XML oder JSON) der Ressource beschreiben. Repräsentationen können Links enthalten, welche Angaben über die Beziehungen der Ressource zu anderen Ressourcen machen. Insbesondere spezifizieren sie sogenannte Selektoren. Diese geben in Abhängigkeit des Repräsentationsformats an, wie die Links aus einer konkreten Ressourcen-Repräsentation ermittelt werden können. Bei der Verwendung von XML als Serialisierungsformat könnte ein Selektor z.B. mittels XPath definiert werden. Links können weitere Angaben, z.B. hinsichtlich des zum Abruf eines Links zu verwendenden Protokolls, enthalten.

Basierend auf diesem Metamodell stellen die Autoren mit ReLL eine XML-basierte Sprache zur Beschreibung von REST-basierten Services vor. Des Weiteren dient dieses Metamodell als Grundlage einer OWL-Ontologie, die als Ausgangspunkt für die Überführung der Ressourcen nach RDF dient. Die eben beschriebenen Konzepte bilden dabei OWL-Klassen, von denen Domain-spezifische Klassen abgeleitet werden. Das bereits erwähnte Tool RESTler parst ReLL-Dokumente und ruft die dort beschriebenen Ressourcen ab. Diese werden an die sogenannte Translator-Komponente weitergeleitet, welche mittels XSLT-Stylesheets die Ressourcen auf Basis der Ontologie nach RDF überführt. Da auch bei diesem Ansatz die eigentliche Abbildung mittels XSLT beschrieben wird, ergeben sich die gleichen Probleme wie bei SAWSDL (siehe Abschnitt 3.1.1).

3.1.8 EXPRESS

In [29] und [28] stellen Alowisheq et al. ein Verfahren, *EXPRESS*, zur Bereitstellung von semantischen, REST-basierten Services auf Basis bereits existierender REST-Services vor. Das dem Service zugrundeliegende Datenmodell wird in Form einer OWL-Ontologie beschrieben. Auf Basis dieser OWL-Ontologie erzeugt eine entsprechende Komponente ein REST-Interface, über welches Instanzen dieser Ontologie verarbeitet werden können. Für jedes Konzept der Ontologie (Klasse und Eigenschaft) wird ein URI-Schema generiert. Der Service-Provider kann für jedes URI-Schema spezifizieren, welche HTTP-Methoden (*GET*, *PUT*, *POST*, *DELETE*) auf die jeweiligen Ressourcen angewendet werden können. Die Abbildung auf die eigentlichen Service-Aufrufe kann z.B. im Code innerhalb der automatisch generierten Stubs, welche das REST-Interface realisieren, angegeben werden. Das so realisierte REST-Interface verarbeitet Instanzen der OWL-Ontologie in Form von RDF-Graphen.

Da die OWL-Ontologie nur das zugrundeliegende Datenmodell spezifiziert und das Mapping zu den Services entweder innerhalb des Codes oder über einen nicht genauer definierten Mapping-Mechanismus definiert wird, wäre die Realisierung eines SPARQL-Endpunktes nur schwierig zu realisieren. Eine Analyse des Codes wäre wesentlich aufwändiger und daher wahrscheinlich auch weniger performant als der in dieser Arbeit vorgestellte Ansatz. Zwar wäre es möglich, einen Algorithmus zur Realisierung eines SPARQL-Endpunktes auf Basis der über diesen Ansatz realisierten semantischen REST-Services zu entwickeln. Allerdings wäre die Erstellung des Mappings auf die dahinterliegenden Services jeweils mit erheblichem Aufwand verbunden.

3.1.9 Zusammenfassung

Wie bereits zu Beginn von Abschnitt 3.1 erwähnt, wurden die in diesem Abschnitt erläuterten Technologien nicht mit Hinblick darauf entwickelt, SPARQL-Endpunkte für Service-Schnittstellen zu realisieren. Obwohl sie dafür verwendet werden könnten, würde dies zu verschiedenen Problemen führen. Wie aus den Erläuterungen der letzten Abschnitte ersichtlich, wird bei den meisten Technologien die Überführung der Ein- und Ausgabenachrichten durch zusätzliche Dokumente beschrieben. Obwohl keine verbindlichen Vorgaben bzgl. der zu verwendeten Sprache zur Beschreibung der Abbildungen gemacht werden, wird in diesem Zusammenhang oftmals auf XSLT verwiesen (SAWSDL, OWL-S, MicroWSMO, SA-REST, ReLL). Wie z.B. bei SAWSDL, hREST/MicroWSMO und SA-REST vorgesehen, kann damit eine Abbildung auf RDF beschrieben werden. Dieser Ansatz hat jedoch verschiedene Nachteile. Es müssen zusätzliche Dokumente erstellt und verwaltet werden. Insbesondere, wenn keine entsprechenden Werkzeuge zur Verfügung stehen, kann dies mit erheblichem Aufwand verbunden sein. Des Weiteren erschwert diese Vorgehensweise die algorithmische Verarbeitung. Da der RDF-Graph, der aus den Daten eines Services erstellt werden kann, implizit durch das XSLT-Dokument beschrieben ist, muss jedes Verfahren zur Realisierung eines SPARQL-Endpunktes die zur Beantwortung einer Anfrage relevanten Services durch Analyse der XSLT-Dokumente bestimmen. Ein ähnliches Problem ergibt sich bei Technologien wie EXPRESS, bei denen die Abbildungen im Code definiert sind. Das im Rahmen dieser Arbeit entwickelte Verfahren zur semantischen Beschreibung von OData-Services (Kapitel 4) umgeht die mit der Erstellung zusätzlicher Dokumente verbundenen Probleme, indem die Beschreibung der Abbildung als Erweiterung von CSDL realisiert wird. Dabei wird die Erstellung der Beschreibung durch Verwendung eines durch SPARQL bekannten Formalismus im Vergleich zu XSLT erheblich vereinfacht. Ein weiterer Vorteil, der sich aus der expliziten Spezifikationen der RDF-Graphen ergibt, besteht darin, dass eine Auflösung von SPARQL-Anfragen gegen die Services auf Basis einer angepassten Evaluierungssemantik realisiert werden kann. Dies vereinfacht die algorithmische Verarbeitung und ermöglicht effiziente Implementierungen auf Basis der Erfahrungen, die sich in den letzten Jahren bei der Implementierung von SPARQL-Prozessoren ergeben haben, wobei auf gängige Optimierungstechniken zurückgegriffen werden kann. Durch die Entwicklung der in Kapitel 4 vorgestellten Konzepte, mit Hinblick auf diesen speziellen Anwendungsfall, wird außerdem ein sehr leichtgewichtiger Ansatz realisiert, der im Vergleich zu sehr schwergewichtigen Ansätzen, wie z.B. OWL-S und WSMO, wesentlich weniger Konzepte umfasst. Dadurch wird der Lernaufwand reduziert und die Erstellung der Beschreibungen vereinfacht. Von der zugrundeliegenden Idee kommt RDFa dem in dieser Arbeit vorgestellten Verfahren zur Beschreibung der Abbildung am nächsten. Jedoch hat es, wie alle anderen in diesem Kapitel vorgestellten Technologien, keinen Bezug zu CSDL und EDM. Das hat zur Folge, dass keine Konzepte zur Handhabung der EDM-spezifischen Konstrukte existieren.

Zusammenfassend kann gesagt werden, dass nach bestem Wissen der in dieser Arbeit vorgestellte Ansatz zur semantischen Beschreibung von OData-Services, das erste Verfahren ist, das speziell mit Hinblick auf OData entwickelt wurde und welches die Konstrukte des OData zugrundeliegenden Entitätsdatenmodells berücksichtigt.

3.2 Service-basierte SPARQL-Endpunkte

3.2.1 ANGIE

Preda et al. stellen in [153] und [154] ein System (ANGIE[14]) vor, das RDF-Datenbanken automatisch mit über Web Services bereitgestellten Daten anreichert. Ausgehend von der Tatsache, dass in den letzten Jahren zahlreiche große auf RDF-basierende Wissensdatenbanken erstellt wurden (siehe z.B. DBpedia [48]), weisen die Autoren darauf hin, dass die in diesen Datenbanken enthaltenen Informationen niemals vollständig sein können. Um die Datenbanken mit fehlenden Informationen zu ergänzen, führen die Autoren das Konzept einer aktiven Wissensdatenbank (*active knowledge base*) ein. Benutzer-Anfragen an eine aktive Wissensdatenbank werden, soweit wie möglich, auf Basis der lokalen Datenbank beantwortet. Wenn die benötigten Informationen nicht in der Datenbank gespeichert sind, werden, für den Benutzer transparent, die gesuchten Daten aus den entsprechenden Web Services abgerufen und in die lokale Wissensdatenbank integriert. Aus technischer Sicht wird dabei folgendes Problem behandelt: Für eine gegebene SPARQL-Anfrage müssen die Web Services ermittelt und aufgerufen werden, die notwendig sind, um die benötigten Daten abzurufen. Um dieses Problem zu lösen, schlagen die Autoren eine auf Datalog[15] basierende Sprache zur Beschreibung der Web Service Schnittstellen vor. Auf Basis dieser Schnittstellenbeschreibungen werden mehrere Algorithmen vorgestellt, welche die notwendigen Serviceaufrufe ermitteln und ausführen. Teilweise werden auch nicht-funktionale Eigenschaften, wie z.B. die Dienstgüte der Services, berücksichtigt. Des Weiteren wird eine Architektur vorgestellt, welche die Grundlage für eine prototypische Realisierung auf Basis von YAGO[16] [177] bildet.

Als Abfragesprache wird eine Teilmenge von SPARQL betrachtet, welche ausschließlich einfache Graph-Muster (BGP) umfasst. Weitere Sprachprimitiven (*FILTER*, *OPTIONAL* usw.) werden nicht berücksichtigt. Das System bzw. die Orchestrierungsalgorithmen, welche auf der vorgestellten SPARQL-Teilmenge basieren, decken daher nur einen sehr kleinen Teil der SPARQL-Funktionalität ab. Die Sprache zur Beschreibung von Web Service Schnittstellen ist Technologie-unabhängig und ermöglicht ausschließlich die Beschreibung der Ein- und Ausgabeparameter. Da keine OData-spezifischen Abfrageoptionen, wie z.B. die Filterung nach bestimmten Parametern, berücksichtigt werden, können sich Orchestrierungen ergeben, welche hinsichtlich der zu übertragenden Datenmenge als auch hinsichtlich der Anzahl der notwendigen Service-Aufrufe nicht optimal sind. Des Weiteren wurden die Algorithmen auf die Anforderung hin entwickelt, für eine gegebene maximale Anzahl an Service-Aufrufen eine möglichst große Antwortmenge zu ermitteln. Hinsichtlich der Abfrage von ERP-Systemen im geschäftlichen Umfeld ergibt

[14] Active Knowledge for Interactive Exploration

[15] Syntaktisch eine Untermenge von Prolog.

[16] Eine semantische Wissensdatenbank, die über 120 Millionen Fakten zu über 10 Millionen Entitäten bereitstellt.

sich hingegen die Anforderung einer Ermittlung der vollständigen Ergebnismenge. Als Format der Web Service Nachrichten wird XML angenommen. Die Erzeugung bzw. Transformation der Service-Nachrichten erfolgt mittels XSLT-Stylesheets [61]. Unklar bleibt, wie die XSLT-Stylesheets erstellt werden.

3.2.2 SEREDASj

Lanthaler und Gütl stellen in [119], mit Vorarbeiten in [118] und [117], eine Sprache mit dem Namen *SEREDASj*[17] (*SE*mantic *RE*stful *DA*ta *S*ervices) zur syntaktischen und semantischen Beschreibung von REST-basierten Services vor. Basierend auf dieser Sprache werden Algorithmen für die Integration REST-basierter Services in das Semantic Web eingeführt. Die Autoren weisen auf eine zunehmende Verbreitung REST-basierter Services hin. Als Problem sehen sie dabei das Fehlen einer allgemein anerkannten maschinenlesbaren Beschreibungssprache an[18]. REST-basierte Services werden häufig mittels umgangssprachlich formulierten Dokumenten beschrieben, die für Menschen gut zu lesen, aber für Maschinen nur sehr schwer oder überhaupt nicht zu verstehen sind. Eine Folge davon ist, nach Ansicht der Autoren, eine enge Kopplung der Systeme bei der Maschinen-zu-Maschinen Kommunikation.

Um dieses Problem zu beheben, wird mit *SEREDASj* eine Sprache zur formalen Beschreibung REST-basierter Services eingeführt. SEREDASj beschränkt sich dabei auf Services, die JSON als Serialisierungsformat verwenden. Eine SEREDASj-Beschreibung ist selbst ein JSON-Dokument, welches einen Service sowohl syntaktisch als auch semantisch beschreibt. Es besteht aus zwei Bereichen: Metadaten und der Element-Beschreibung. Der Metadaten-Bereich enthält Angaben, wie z.B. Präfix-Definitionen und Angaben zu den Links zwischen Ressourcen. Die Element-Beschreibung beschreibt die Struktur der mit JSON serialisierten Service-Nachrichten. Beide Bereiche können dabei zur semantischen Beschreibung Referenzen auf entsprechende Domänen-Ontologien enthalten. Auf Basis dieser Beschreibung ist es sowohl möglich, automatisiert eine menschenlesbare Dokumentation zu erstellen, als auch die JSON-Nachrichten automatisch nach RDF zu überführen.

Darauf aufbauend wird ein Referenzmodell zur Integration REST-basierter JSON-Services mit bestehenden RDF-Datensätzen vorgestellt. Bei den bestehenden RDF-Datensätzen kann es sich dabei um RDF-Dumps, in HTML-Dokumente eingebettete RDF-Daten oder um über SPARQL-Endpunkte abfragbare Datensätze handeln. Das Referenzmodell enthält einen anwendungsunabhängigen Datenzugriffs-Layer, der von Client-Anwendungen über SPARQL oder SPARUL [83] angefragt werden kann. Für eine gegebene Anfrage werden die benötigten Daten von den vorhandenen RDF-Datensätzen und den mit SEREDASj beschriebenen REST-Services abgerufen und in einem lokalen Tripel-Store gespeichert, gegen welchen die Abfrage ausgeführt wird. Neben der Transformation und Integration der Daten unterstützt der Layer auch Caching. Des Weiteren enthält der Layer

[17] Das *j* weist darauf hin, dass *SEREDASj* für JSON-basierte REST-Services [66] entwickelt wurde.

[18] Obwohl von den Autoren als Problem angesehen, wird die Notwendigkeit der formalen Beschreibung REST-basierter Services aktuell kontrovers diskutiert (siehe z.B. [86, 160]).

Informationen darüber, welche Daten verändert werden können. Dabei kann selbst für unveränderliche Datensätze ein Schreib- bzw. Update-Mechanismus realisiert werden, indem der Layer alle Datenänderungen speichert und bei Abfrage die Datensätze aus den Datenquellen entsprechend überschreibt.

Die Autoren stellen zwei Algorithmen zur Überführung von SPARUL-Anfragen in REST-Service-Aufrufe (HTTP PUT und POST) vor. Der erste Algorithmus behandelt die Überführung der SPARUL-Anweisungen INSERT DATA und DELETE DATA in die HTTP-Methoden POST, PUT und DELETE. Dazu werden zunächst alle SEREDASj-Beschreibungen ermittelt, die Konzepte referenzieren, welche Teil der SPARQL-Anfrage sind. Die REST-Anfragen werden aus den Beschreibungen und den Daten der SPARQL-Anfrage bzw. den im lokalen Tripel-Store gespeicherten Daten erzeugt. Die Ergebnisse werden nach RDF überführt und lokal gespeichert. Tripel der SPARUL-Anfrage, die durch die REST-Aufrufe abgedeckt wurden, werden aus der SPARUL-Anfrage entfernt. Des Weiteren werden leere Knoten durch konkrete Werte ersetzt. Dieser Prozess wird wiederholt, bis die gesamte SPARUL-Anfrage abgedeckt wurde. Die Pattern-basierten DELETE und INSERT Anweisungen werden von dem zweiten Algorithmus in REST-Aufrufe transformiert, indem zunächst durch das Ausführen der WHERE-Klausel alle vorkommenden Variablen gebunden werden. Anschließend werden die INSERT und DELETE Anweisungen für die gebundenen Variablen in INSERT DATA und DELETE DATA Anweisungen überführt. Mit dem ersten Algorithmus erfolgt schließlich die Abbildung auf REST-Service-Aufrufe.

Die vorgestellte Lösung beschränkt sich auf REST-Services, welche JSON als Serialisierungsformat verwenden. Ein Algorithmus zur Bestimmung der relevanten Services fehlt. Die Algorithmen zur Überführung von SPARUL-Anfragen in REST-Service-Aufrufe sind hinsichtlich der benötigen Anzahl an Service-Aufrufen nicht optimal. Da keine OData-spezifischen Konstrukte berücksichtigt werden, ergeben sich für OData-basierte Services sowohl mit Hinblick auf die zu übertragenden Datenmengen als auch bzgl. der Anzahl benötigter Services ineffiziente Service-Kompositionen.

3.2.3 Semantic Bridge for Web Services

Battle und Benson stellen in [37] ein System vor (*Semantic Bridge for Web Services*), das die Abfrage von SOAP- und REST-basierten Services mittels SPARQL ermöglicht. Des Weiteren werden unter der Bezeichnung *Semantic REST* Konzepte eingeführt, um die Bereitstellung, Abfrage und Modifikation von REST-basierten Ressourcen in RDF zu ermöglichen. Im Folgenden wird die *Semantic Bridge for Web Services* (SBWS) genauer betrachtet.

Ausgehend von der Beobachtung, dass große Mengen an Daten über REST- und SOAP-basierte Web Services verfügbar sind, weisen die Autoren auf den Nutzen hin, der sich durch eine Integration dieser Services in das Semantic Web ergeben würde. Die Integration der Services erfolgt mittels eines SPARQL-Endpunktes, über den die Daten der Services verfügbar gemacht werden. Der SPARQL-Endpunkt wird durch eine semantische Beschreibung der Service-Schnittstelle realisiert, auf Basis dessen ein Orchestrierungsalgorithmus für eine gegebene SPARQL-Anfrage die zur Beantwortung der Anfrage notwendigen Services ermittelt. Zur semantischen Beschreibung von SOAP-/WSDL-basierten Web Services wird OWL-S [132] verwendet. Die Unterstützung von SAWSDL

[122] ist geplant. Die Überführung der XML-Service-Nachrichten erfolgt mittels XSLT-Dokumenten [61], die über das entsprechende OWL-S Dokument referenziert werden müssen. Neben SOAP-/WSDL-basierten Web Services werden auch auf WADL-basierte REST-Services [89] unterstützt. Dazu erweitern die Autoren WADL um zusätzliche Elemente, welche die semantische Beschreibung der Ein- und Ausgabeparameter bzw. der Ressourcen ermöglichen. Die Transformation der Nachrichten nach RDF wird ebenfalls mittels XSLT beschrieben. Der Orchestrierungsalgorithmus wird informal anhand eines einfachen Beispiels erläutert. Die genaue Funktionsweise bleibt unklar. Die SBWS ist als Teil *Asio Tool Suite* von Raytheon BBN Technologies [158] kommerziell erhältlich.

Da bei der auf WADL aufsetzenden, semantischen Beschreibung der REST-Services XSLT zur Überführung der Daten nach RDF verwendet wird, ergeben sich die bereits mehrfach erläuterten Probleme. Es werden keine Aussagen über die Funktionsweise des Algorithmus gemacht. Daher ist es nicht möglich, Aussagen über den Abdeckungsgrad und die Effizienz des Algorithmus zu treffen. Da der Algorithmus aber für die Orchestrierung WADL-basierter REST-Services entwickelt wurde, werden OData-spezifische Konstrukte bei der Service-Orchestrierung nicht berücksichtigt. Dadurch ergeben sich ineffiziente Orchestrierungen, die sowohl hinsichtlich der Anzahl der Service-Aufrufe als auch hinsichtlich der übertragenen Datenmengen nicht optimal sind. Die Autoren merken außerdem an, dass die SBWS auf SPARQL *SELECT*- und *CONSTRUCT*- Anfragen beschränkt ist. Des Weiteren ist anzumerken, dass eine Architektur für eine mögliche Realisierung des Systems fehlt.

3.2.4 Generic semantic problem-solving platform

Verborgh et al. stellen in [186] und [185] (siehe auch [184]) eine generische, auf semantischen Technologien basierende Plattform zur automatischen Service-Orchestrierung vor. Neben der eigentlichen Orchestrierung überwacht die Plattform auch die Ausführung und führt im Falle von Fehlern entsprechende Gegenmaßnahmen durch. Die Autoren betrachten als Anwendungsfall die automatische Annotation von Multimediainhalten, wie z.B. Bildern und Videos, mit Metadaten. Insbesondere wird der Fall berücksichtigt, Bilder automatisch mit den Namen der auf ihnen abgebildeten Personen zu beschreiben. Dafür sind mehrere Algorithmen (Gesichtserkennungs- und bestimmungs-Algorithmen) notwendig, die in einer bestimmten Reihenfolge ausgeführt werden müssen.

Das Ziel der vorgestellten Plattform besteht darin, für solch eine gegebene Aufgabe eine Komposition von Algorithmen zu ermitteln, welche diese Aufgabe erfüllt. Des Weiteren sollen Daten aus dem Semantic Web (z.B. DBpedia [48] und Freebase [51]) zur Verwaltung des Kontextes und zur semantischen Annotation verwendet werden. Algorithmen werden dabei als Web Services realisiert, auf die mittels SPARQL zugegriffen werden kann. Die Beschreibung der Web Service Schnittstellen erfolgt mittels OWL-S [132], wobei auf Basis von SPARQL Service Description [193] ein Service-Grounding definiert wird. Aufgaben bzw. Anfragen werden ebenfalls als SPARQL-Ausdrücke beschrieben. Zur Ermittlung der Service-Orchestrierung für eine gegebene SPARQL-Anfrage wird zunächst jede Service-Beschreibung in N3Logic Regeln [46] überführt. Jede OWL-S Service-Beschreibung ergibt somit eine Regel, welche die Eingabeparameter auf die Ausgabeparameter abbildet. Die Regeln werden

außerdem mit bestimmten Informationen erweitert, die über die Erzeugung zusätzlicher Aussagen die Nachverfolgung einer Ableitung ermöglichen. Anschließend wird die SPARQL-Anfrage ebenfalls in eine Regel überführt. Auf Basis der verfügbaren Eingabedaten ermittelt nun ein Reasoner, ob die Anfrage-Regel mittels der Service-Regeln hergeleitet werden kann. Die Service-Orchestrierungen ergeben sich aus den bei der Ableitung verwendeten Regeln, welche über die während der Ableitung erzeugten Aussagen nachvollzogen werden können. Ausgehend von verschiedenen Service-Kriterien, wie z.B. Performance, Verfügbarkeit usw., können ein Vergleich und eine Auswahl der Orchestrierungen erfolgen.

Obwohl die vorgestellte Architektur auf dem häufig in der KI verwendeten Blackboard Architektur Modell [64] basiert, entspricht diese, mit Abweichungen in den Bezeichnungen, weitgehend dem im Bereich der Web Service Komposition gängigen Modell, wie es in [157] beschrieben und auch im Rahmen dieser Arbeit verwendet wird. Der Ansatz wurde mit Hinblick auf SPARQL-basierte Web Services entwickelt. Für bestehende Services ist somit fast immer die Entwicklung eines Wrappers erforderlich. Deswegen kann das vorgestellte System nur für bestimmte oder speziell angepasste Web Services verwendet werden. Folglich werden OData-spezifische Features nicht berücksichtigt. Der Vorteil dieser Annahme ist allerdings, dass die Ergebnisse der Web Services direkt in RDF vorliegen (*CONSTRUCT*) und keine separate Überführung nach RDF erforderlich ist. Der Vorteil des Regel-basierten Ansatzes besteht darin, dass beliebige weitere Regeln, die unabhängig von den konkreten Service-Beschreibungen sind, mit in den Ableitungsprozess einbezogen werden können.

3.2.5 SADI + SHARE

Wilkinson et al. stellen in [190, 191, 192], mit Hinblick auf verschiedene Anwendungsfälle aus der Chemie- und Bioinformatik [58, 159], unter der Bezeichnung *SADI* (*Semantic Automated Discovery and Integration*), eine Menge von Empfehlungen für die Bereitstellung von Services mittels semantischer Technologien vor. Darauf aufbauend, zeigen Vandervalk et al. [181, 182, 183] mit *SHARE* (*Semantic Health and Research Environment*) die Abfrage von SADI-basierten Web Services mittels SPARQL.

Ein Großteil der für den Bereich der Bioinformatik relevanten Daten ist nur über Web Services abfragbar. Dabei stellen die Autoren fest, dass diese Services häufig eine Vielzahl von Gemeinsamkeiten aufweisen. Es werden folgende Übereinstimmungen genannt: unabhängig, idempotent, zustandslos, transformativ[19], atomar. Mit Hinblick auf diese Eigenschaften wurde SADI entwickelt. Ein Service ist SADI-konform, wenn er folgende Empfehlungen berücksichtigt [191]:

- Der Service ist atomar und zustandslos.
- Die Beschreibung der Service-Schnittstelle kann mit HTTP GET abgerufen werden.

[19] Bezeichnet die Eigenschaft, dass das Ergebnis eines Services eine Transformation der Eingabeparameter ist.

- Die Beschreibung der Service-Schnittstelle erfolgt mittels OWL-DL in Form von Klassen und Rollen, wobei die Rollen die Ein- und Ausgabeparameter des Services beschreiben.
- Das Ein- und Ausgabeformat des Services ist RDF.
- Der Aufruf des Services erfolgt, indem die RDF-Daten per HTTP POST an den Service-Endpunkt gesendet werden.
- Der Service erweitert die als Eingabe übergebene OWL-Instanz um zusätzliche Rollen. Die URIs der Ein- und Ausgabeinstanzen stimmen somit überein.

Darauf aufbauend realisiert SHARE eine Abfrage-Engine für SADI-konforme Services und SPARQL-Endpunkte. SPARQL-Anfragen werden von SHARE in eine Menge von SADI-Service- und SPARQL-Endpunkt-Aufrufen umgewandelt. Dazu werden die Tripel Pattern innerhalb der WHERE-Klausel der SPARQL-Anfrage mit den SADI-Service-Beschreibungen verglichen. Es werden alle Services ermittelt, die mit den gegebenen Daten aufgerufen werden können und die Werte für das gesuchte Prädikat des Tripel-Patterns liefern. Die Ergebnisse der Service-Aufrufe werden im lokalen Triple Store gespeichert und als Binding für Subjekt- und Objektvariablen der verbleibenden Tripel-Pattern verwendet. Nach diesem Schema werden die Tripel Pattern sukzessiv abgearbeitet. Die sich ergebende Service-Orchestrierung ruft alle zur Beantwortung der Anfrage benötigten Daten ab.

Wie die Autoren selbst anmerken, gelten die für Web Services im Bereich der Bioinformatik gemachten Annahmen außerhalb dieser Domäne üblicherweise nicht. Der Einsatzbereich des vorgestellten Systems ist daher auf eine spezielle Menge von Services beschränkt. Selbst Services, welche diese Annahmen erfüllen, können nicht ohne weiteres eingebunden werden, da zunächst eine Anpassung an die durch SADI gestellten Anforderungen erforderlich ist. Der Vorteil des beschriebenen Ansatzes besteht darin, dass keine Transformation der Rückgabedaten notwendig ist. Des Weiteren können bereits während der Ausführung der Service-Orchestrierung mittels entsprechender Reasoner zusätzliche Fakten erzeugt werden, die bei den Service-Aufrufen verwendet werden können.

3.2.6 Uniform Semantic Data Integration System

Zhao et al. stellen in [196] und [195] ein System, USDIS (*Uniform Semantic Data Integration System*), zur Integration von Web Services mit RDF-Datensätzen vor. Aufbauend auf einer Sprache zur semantischen Beschreibung von Web Services wird ein Algorithmus erläutert, der für eine an SPARQL angelehnte Anfragesprache die zur Abarbeitung der Anfrage benötigten Web Services ermittelt.

Die Autoren weisen mit Hinblick auf dynamische Daten (Sensordaten, Wetterdaten usw.) auf den Nutzen hin, der sich durch die Integration von RDF-Datensätzen mit SOAP-/WSDL-basierten Web Services ergeben würde. Dabei unterscheiden sie zwischen Web Services, welche Informationen bereitstellen (Daten-Services) und Web Services, welche den Zustand der Welt ändern (Web Services mit Seiteneffekten). Als Problem bei der semantischen Beschreibung von Daten-Services mit existierenden Ansätzen, wie WSMO [72],

SAWSDL [122] usw., sehen sie die Annotation der Ein- und Ausgabeparameter mit Listen von Konzepten an. Insbesondere bei der Abfrage von Daten über Daten-Services entsprechen die abgefragten Daten üblicherweise den konkreten Rollen anstatt den Konzepten. Um dieses Problem zu beheben, wird unter der Bezeichnung *Uniform Query* ein neues Modell zur Beschreibung von Web Service Schnittstellen bzw. Datenquellen und zur Formulierung von Abfragen vorgestellt. Das Uniform Query Konzept ist an SPARQL angelehnt. Es werden jedoch zusätzliche Konzepte eingeführt, um z.B. Ein- und Ausgabeparameter eines Services explizit spezifizieren zu können. Der vorgestellte Matching-Algorithmus ermittelt für eine mittels Uniform Query spezifizierte Anfrage die zur Beantwortung der Anfrage benötigten Web Services und Datenquellen durch einen Vergleich der referenzierten Konzepte und Teilgraphen. Dazu wird geprüft, ob der in der Anfrage beschriebene Graph ein Teilgraph des in der Web Service Beschreibung spezifizierten Graphen ist. Anschließend erfolgt ein Vergleich der Wertebereiche und der Ein- und Ausgabeparameter. Während des Vergleichs wird ein Ausführungsplan erstellt, in dem das Binding der Parameter beschrieben wird.

Der größte Nachteil des vorgestellten Systems ist die Verwendung von Uniform Query als Anfragesprache. Obwohl diese an SPARQL angelehnt ist, wird nur ein eingeschränkter Teil der SPARQL-Funktionalität abgedeckt. Des Weiteren kann das System nicht mit anderen Systemen, die auf SPARQL aufsetzen, wie z.B. Systeme zur föderierten Abfrage [156], verwendet werden. Der vorgestellte Algorithmus ist Technologie-unabhängig, beschränkt sich jedoch auf das Matching einzelner Services. Technologie-abhängige Optimierungsmöglichkeiten und Inferenzmechanismen werden nicht berücksichtigt.

3.2.7 Composing Data Providing Services

Barhamgi et al. stellen in [34, 35, 36] Konzepte für die Beschreibung von Daten-Services vor. Darauf aufbauend wird gezeigt, wie mittels eines Mediators ein Abfragesystem realisiert werden kann. Der Mediator ermittelt für eine gegebene Anfrage eine Service-Komposition, welche die zur Beantwortung der Anfrage benötigten Daten abruft.

Die Autoren unterscheiden Daten-Services (*Data-Providing Services* (*DP*)) von Services mit Nebeneffekten (*Effect-Providing Services* (*EP*)). DP-Services haben in Gegensatz zu EP-Services bei Aufruf keine Zustandsänderungen zur Folge. Es wird angemerkt, dass die meisten Verfahren zur automatischen Service-Komposition mit Hinblick auf EP-Services entwickelt wurden. Insbesondere umfassen die Beschreibungen der Services die Spezifikation der Aufrufbedingungen (*preconditions*) und Nebeneffekte (*effects*), welche für die Beschreibung von DP-Services nicht relevant sind. Andererseits können die Beschreibung von komplexen Beziehungen zwischen den Ein- und Ausgabedaten, wie sie bei DP-Services häufig vorkommen, nur sehr umständlich spezifiziert werden.

Ausgehend von dieser Beobachtung wird von den Autoren ein auf einfachen Graph-Mustern basierendes Konzept zur semantischen Beschreibung von Service-Schnittstellen vorgeschlagen. Die Autoren bezeichnen dieses als parametrisierte RDF-Sichten (*RDF Parameterized Views (RPVs)*) Es ermöglicht die Spezifikation der Beziehungen der Ein- und Ausgabeparameter zu den Konzepten der verwendeten Ontologie, sowie deren Beziehungen untereinander. Dabei besteht auch die Möglichkeit, Nebenbedingungen (*constraints*) anzugeben. Als Abfragesprache wird eine Untermenge von SPARQL betrachtet. Diese

unterstützt neben konjunktiven Abfragen auch einfache Filter-Bedingungen. Der Orchestrierungsalgorithmus ist zweistufig. Zunächst werden alle Services ermittelt, die einen Teil der Anfrage abdecken. Dazu wird die Anfrage mit den parametrisierten RDF-Sichten verglichen, wobei geprüft wird, ob eine Abbildung der Anfrage auf die Service-Beschreibungen möglich ist. Die Semantik von RDFS [88] wird bei der Service-Komposition berücksichtigt, indem vor der Ermittlung der Services die Service-Beschreibungen entsprechend der RDFS-Semantik erweitert werden. Im zweiten Schritt wird aus den gefundenen Services die Service-Komposition bestimmt, indem mögliche Ausführungsreihenfolgen berechnet werden. Des Weiteren wird eine Verallgemeinerung des vorgestellten Verfahrens angegeben, um sogenannte parametrisierte Anfragen behandeln zu können. Parametrisierte Anfragen unterscheiden sich zu nicht-parametrisierten Anfragen dadurch, dass die konkreten Werte durch Variablen ersetzt wurden. Damit können Service-Kompositionen ermittelt werden, die unabhängig von konkreten Eingabewerten sind. Dabei können aber zusätzliche Abfragen zur Laufzeit notwendig sein, um anhand der gegebenen Werte die benötigten Services auszuwählen.

Der vorgestellte Ansatz wurde mit Hinblick auf SOAP-/WSDL-basierte Web Services entwickelt. Folglich werden keine OData-spezifischen Optimierungsmechanismen berücksichtigt. Des Weiteren werden nur konjunktive Anfragen mit einfachen Filterbedingungen unterstützt.

3.2.8 Weitere Arbeiten

Neben den bereits vorgestellten Ansätzen existieren verschiedene Arbeiten, die eine ähnliche Ausrichtung haben oder diese Arbeit ergänzen.

Domänen-unabhängig und aus einer anderen Motivation heraus, aber grundsätzlich einen ähnlichen Ansatz wie Wilkinson und Vandervalk (Abschnitt 3.2.5) verfolgen auch Speiser und Harth [171, 172, 173] bzw. Krummenacher et al. [112, 144]. Um eine Integration von Daten-Services mit Linked Data [42, 47, 188] zu ermöglichen, werden Konzepte zur semantischen Beschreibung der Services vorgestellt. Auf Basis der Beschreibung wird eine Schnittstelle realisiert, welche die über den Service abfragbaren Daten als Linked Data zur Verfügung stellt (siehe dazu auch [25, 27]). Auf Basis der Linked Data Schnittstelle kann ein SPARQL-Endpunkt für die Abfrage der entsprechenden Services realisiert werden [93, 95, 113].

Obwohl es möglich ist, auf diese Art und Weise einen SPARQL-Endpunkt für eine Menge von Web Services bereitzustellen, hat dieser Ansatz verschiedene Nachteile. Bei der Abfrage einer Ressource über die Linked Data Schnittstelle werden üblicherweise alle Prädikate dieser Ressource zurückgegeben. Dies geschieht unabhängig davon, welche Daten zur Beantwortung der gegebenen SPARQL-Anfrage tatsächlich benötigt werden. Insbesondere bei Ressourcen mit vielen Prädikaten, wie es z.B. bei Geschäftsobjekten in ERP-Systemen häufig der Fall ist, kann dieses Vorgehen aber zu einer sehr ineffizienten Abfrageverarbeitung führen. Des Weiteren hängt die unterstützte Ausdrucksstärke vom verwendeten Abfrageverfahren ab. Viele der eingesetzten Verfahren, z.B. [93, 95, 113], beschränken sich jedoch auf konjunktive Abfragen (einfache Graph-Muster). Damit wird zwar ein wichtiger, aber nur kleiner Teil der SPARQL-Funktionalität abgedeckt.

Es sei darauf hingewiesen, dass der umgekehrte Weg einer Linked Data Schnittstelle auf Basis eines SPARQL-Endpunktes ebenfalls möglich ist [67]. Auf Grundlage der in dieser Arbeit vorgestellten Konzepte kann daher auch eine Linked Data Schnittstelle für ERP-Systeme bzw. OData-basierte REST-Services realisiert werden.

Als komplementär zu dieser Arbeit können Systeme zur föderierten Abfrage (*federated queries*) von SPARQL-Endpunkten angesehen werden. Systeme, wie *DARQ* (*Distributed ARQ*[20]) [156] und *SemWIQ* (*Semantic Web Integator and Query Engine*) [114, 115, 116] ermöglichen mittels eines Mediator-basierten Ansatzes [189] die transparente Abfrage mehrerer SPARQL-Endpunkte über eine einzige SPARQL-Schnittstelle. Somit kann die transparente Abfrage über mehrere ERP-Systeme bzw. OData-basierte REST-Services verwirklicht werden. Dabei ist mit den in [126, 127, 128] bzw. [65] vorgestellten Verfahren auch die Integration mehrerer Systeme mit unterschiedlichen Ontologien möglich. Wobei mit den in [49] und [165] erläuterten Ansätzen auch Unterschiede bei den Ressourcen-URIs (*coreference* [24]) behandelt werden können.

3.2.9 Zusammenfassung

Die vorgestellten Ansätze unterscheiden sich hinsichtlich der Zielsetzung, der adressierten Services, der unterstützten Anfragesprache und der Umsetzung teilweise erheblich voneinander. Die Ansätze von Zhao et al. (*USDIS*) und von Barhamgi et al. fokussieren sich auf SOAP-/WSDL-basierte Web Services. *ANGIE* und die *Semantic Bridge for Web Services* hingegen betrachten sowohl REST- als auch SOAP-basierte Services. Lanthaler und Gütl adressieren JSON-basierte REST-Services. Die Systeme von Verborgh et al. (*generic semantic problem-solving platform*) und Vandervalk et al. (*SADI + SHARE*) stellen spezielle Anforderungen an die Services. Unter anderem müssen SADI-konforme Web Services RDF als Ein- und Ausgabeformat unterstützen. Das von Verborgh et al. vorgestellte System kann nur Services verarbeiten, die selbst SPARQL-Endpunkte sind.

Keines der Systeme wurde speziell mit Hinblick auf OData-Services entwickelt. Dies hat zur Folge, dass OData-spezifische Abfrageoptionen bei der Erstellung der Anfragen nicht berücksichtigt werden. Dadurch ergeben sich Service-Kompositionen, die sowohl hinsichtlich der Anzahl der Service-Aufrufe als auch der zu übertragenden Datenmengen nicht optimal sind. Im Gegensatz dazu ruft das im Rahmen dieser Arbeit entwickelte Verfahren nur die zur Beantwortung der Anfrage tatsächlich benötigten Daten ab, indem die Menge der adressierten Entitäten und Eigenschaften mittels der Abfrageoptionen entsprechend eingeschränkt werden.

Mit Hinblick auf die Abfragesprache werden häufig nur Teilmengen von SPARQL unterstützt. *ANGIE* beispielsweise beschränkt sich auf einfache Graph-Muster. Das System von Barhamgi et al. unterstützt, neben einfachen konjunktiven Anfragen, auch Filter-Anweisungen. USDIS verwendet eine auf SPARQL basierende Anfragesprache. Verfahren, die nur Teilmengen von SPARQL unterstützen, haben neben der eingeschränkten Ausdrucksstärke immer auch den Nachteil, dass eine Anbindung an andere Systeme, die auf SPARQL basieren, wie z.B. Systeme zur föderierten Abfrage, ggf. nicht möglich ist.

[20] ARQ ist der SPARQL-Prozessor von Apache Jena [178]: http://jena.sourceforge.net/ARQ/

Ähnliches gilt auch für nicht standardkonforme Erweiterungen von SPARQL. Der in dieser Arbeit vorgestellte Ansatz adressiert dieses Problem, indem er auf der SPARQL-Algebra aufsetzt, um eine vollständige Abdeckung von SPARQL zu realisieren.

Die vorgestellten Algorithmen reichen von einem einfachen Matching der Anfragen gegen die Service-Beschreibungen bis zur Ableitung der Service-Orchestrierungen auf Basis von N3Logic Regeln. Teilweise werden auch keine genaueren Angaben zur tatsächlichen Funktionsweise des Algorithmus gemacht (*Semantic Bridge for Web Services*). In dieser Arbeit werden die zur Beantwortung einer Anfrage benötigten Services auf Basis einer angepassten Evaluierungssemantik ermittelt. Dabei werden keine Vorgaben gemacht, wie eine konkrete Implementierung diese umzusetzen hat. Dies hat gegenüber den bereits existierenden Ansätzen den Vorteil, dass bei der Implementierung auf den Erfahrungen aufgesetzt werden kann, die innerhalb der letzten Jahre bei der Implementierung der SPARQL-Prozessoren gewonnen wurden. Des Weiteren können gängige Optimierungsmechanismen [94, 166, 175] eingesetzt werden, wodurch auch die Verarbeitung sehr großer Service-Mengen ermöglicht wird.

Hinsichtlich der semantischen Beschreibungen kommt bei den vorgestellten Verfahren ebenfalls eine große Vielzahl an eingesetzten Technologien zum Einsatz. *ANGIE* verwendet eine Beschreibungssprache auf Basis von Datalog. Wobei XSLT zur Überführung der Daten eingesetzt wird, was zu den bereits beschriebenen Problemen führt. Das gleiche gilt bei der *Semantic Bridge for Web Services*, welche ebenfalls XSLT zur Transformation verwendet. Dabei erfolgt, wie bei dem Verfahren von Verborg, die semantische Beschreibung auf Basis von OWL-S. Die anderen Ansätze verwenden neu entwickelte Beschreibungssprachen, die teilweise an SPARQL angelehnt sind. Keine dieser Sprachen wurde mit Hinblick auf OData-Services und das Entitätsdatenmodell entwickelt.

Nach bestem Wissen ist das in dieser Arbeit vorgestellte Verfahren zur Abbildung von SPARQL-Anfragen auf Service-Aufrufe das erste, welches mit Bezug auf OData-Services entwickelt wurde und die OData-spezifischen Konzepte, wie Abfrageoptionen, bei der Erstellung der Kompositionen berücksichtigt. Des Weiteren ist auch die Bestimmung der zur Beantwortung einer Anfrage relevanten Services auf Basis einer angepassten Evaluierungssemantik eine neuartige Herangehensweise, wodurch sich die bereits erläuterten Vorteile ergeben.

Teil II

Lösungsansatz

4 Semantische Beschreibung von OData-Services

In diesem Abschnitt werden Konzepte zur semantischen Beschreibung von OData-Services hergeleitet (siehe dazu auch [106]). Die semantische Beschreibung erfolgt mit dem Ziel, die zur Abarbeitung einer gegebenen SPARQL-Anfrage notwendigen OData-Service-Aufrufe ermitteln zu können.

Die Herleitung der Konzepte wird anhand von Beispielen demonstriert, die auf dem öffentlich zugänglichen *Northwind*-OData-Service basieren [136]. Dieser Service stellt eine OData-Schnittstelle für die Northwind-Datenbank zur Verfügung. Die Northwind-Datenbank ist eine Beispieldatenbank von Microsoft, welche Daten über die imaginäre Firma *Northwind Traders* enthält. Dazu gehören Informationen über Kunden, Angestellte, Bestellungen usw. Der Service wird als Grundlage für die folgenden Beispiele verwendet, weil er im Gegensatz zu den OData-basierten Schnittstellen des ERP-Systems öffentlich zugänglich ist und daher eine einfache Verifizierung der hier vorgestellten Konzepte ermöglicht. Des Weiteren ist das zugrundeliegende Datenmodell weniger umfangreich und leichter verständlich. Dadurch kann der Fokus der nachfolgenden Erläuterungen auf der semantischen Beschreibung liegen. Eine detaillierte Beschreibung des Datenmodells ist nicht erforderlich, da dieses weitgehend intuitiv ist. Andererseits ist es komplex genug, um anhand des Services viele Problemstellungen erläutern zu können. Ein weiterer Vorteil besteht darin, dass der Service einen geschäftlichen Bezug hat. Alle vorkommenden Konzepte finden sich in abgewandelter und fein-granularer Form auch im internen Datenmodell des ERP-Systems wieder. Eine grafische Darstellung in Form eines UML-Klassendiagrams des dem OData-Service zugrundeliegenden Entitätsdatenmodells ist in Anhang A zu finden. Wobei das Klassendiagramm nicht das vollständige Datenmodell zeigt, sondern nur einen für diese Arbeit relevanten Ausschnitt. Klassen entsprechen dabei den Entitätstypen und Attribute den Eigenschaften der Entitätstypen. Die Darstellung von Entitätstypen als Klassen in UML-Notation wird im Folgenden durchgängig verwendet werden. Soweit nicht anders angegeben, sind alle in diesem Kapitel gezeigten Datenmodelle Teil dieses dem Northwind-Service zugrungeliegenden Datenmodells. Die dazugehörigen CSDL-Beschreibungen sind Ausschnitte aus dem Service-Metadaten-Dokument des Northwind-Services [135]. Als Container für die Konzepte des Northwind-Modells dient im Folgenden ein durch das Namensraumpräfix *northw* gekennzeichneter Namensraum.

4.1 Abbildung konzeptueller Datenmodelle auf RDF-Templates

Um die für die Abarbeitung einer Anfrage relevanten Services bestimmen zu können, muss die semantische Service-Beschreibung die Struktur der RDF-Graphen definieren, die aus den Rückgabedaten der Services erstellt werden. Die Struktur der vom Service zurückgegebenen Daten ist durch das dem Service zugrundeliegende Entitätsdatenmodell (EDM) gegeben. Die semantische Beschreibung muss spezifizieren, wie das Entitätsdatenmodell auf RDF-Graphen abgebildet wird. Abbildung 9 zeigt dieses Prinzip.

Wie in Abschnitt 2.2 beschrieben, ist SPARQL eine Abfragesprache für RDF. Sie hat kein Verständnis von der Semantik von Ontologiesprachen, wie z.B. OWL [97]. Für die

Beantwortung einer SPARQL-Anfrage ist es daher nicht von Bedeutung, ob die im RDF-Graphen verwendeten Konzepte als Teil einer Ontologie beschrieben sind. Somit ist es für die semantische Beschreibung ebenfalls irrelevant, ob die Konzepte des RDF-Graphen Teil einer Ontologie sind. Die vorgestellten Konzepte sind daher unabhängig von einer bestimmten Ontologiesprache, wie z.B. OWL. Es sei aber darauf hingewiesen, dass es in vielen Fällen, z.B. im Rahmen einer Informationsintegration [125], sinnvoll sein kann, die verwendeten Konzepte als Teil einer Ontologie zu beschreiben. Dies steht jedoch nicht im Widerspruch zu dem hier vorgestellten Ansatz.

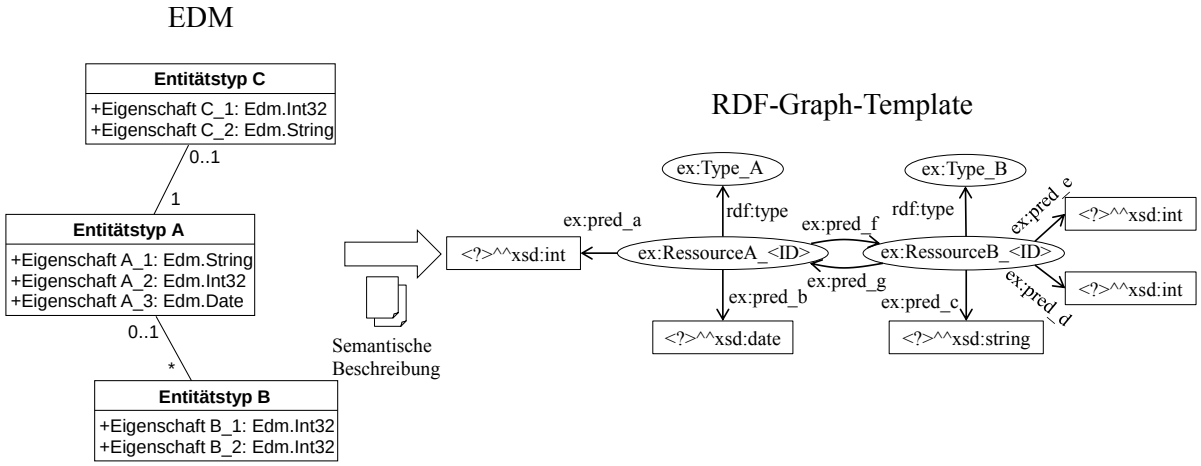


Abbildung 9: Abbildung eines EDM auf ein RDF-Graph-Template durch die semantische Beschreibung.

Die semantische Beschreibung definiert ein *RDF-Graph-Template* (im Folgenden auch als *Graph-Template* bezeichnet), welches die Struktur der RDF-Graphen beschreibt, die aus den Rückgabedaten der Services erstellt werden. Des Weiteren stellt es die Beziehung zu dem Entitätsdatenmodell her, durch das die Struktur der Rückgabedaten spezifiziert ist. Das Graph-Template enthält alle Informationen, die notwendig sind, um die entsprechenden RDF-Graphen aus den vom Service bereitgestellten Daten erzeugen zu können. Abbildung 10 zeigt ein Beispiel, welches an das Entitätsdatenmodell des Northwind-Services [135] angelehnt ist.

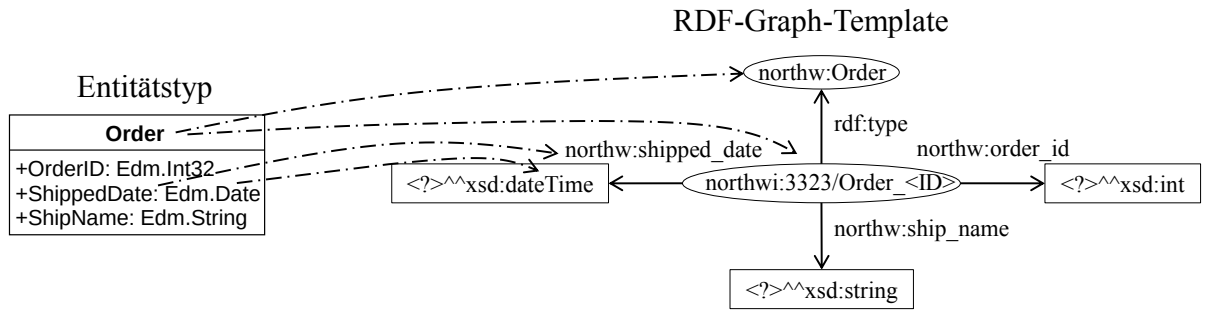


Abbildung 10: Die semantische Beschreibung eines Entitätstyps.

Auf der linken Seite von Abbildung 10 ist als Klasse in UML-Notation der Entitätstyp (*Order*) dargestellt. Aus Gründen der Übersichtlichkeit wird nur ein kleiner Teil des Entitätstyps gezeigt. Der Entitätstyp *Order* beschreibt die Struktur einer Bestellung. Er umfasst unter anderem die drei Eigenschaften *OrderID*, *ShippedDate* und *ShipName*. *OrderID* ist eine ID,

die eine Bestellung innerhalb einer Entitätsmenge eindeutig identifiziert (Entitätsschlüssel). Die Eigenschaft *ShippedDate* gibt das Lieferdatum an und die Eigenschaft *ShipName* enthält den Namen des Empfängers. Der Northwind-Service stellt eine Kollektion (*Orders*) zur Verfügung, über die alle Entitäten des Entitätstyps *Orders* abgerufen werden können. Die rechte Seite von Abbildung 10 zeigt das Template eines RDF-Graphen, das als Vorlage für die Überführung von Entitäten des Entitätstyps *Order* nach RDF dient. Die semantische Beschreibung muss dieses Template definieren und zu dem entsprechenden Entitätstyp in Beziehung setzen. Dem Template entsprechend wird jede Entität des Entitätstyps *Order* in genau eine Ressource des Typs *northw:Order* überführt. Diese Ressource hat die URI *northwi:3323/Order_<ID>*, wobei *northwi* für eine beliebige URI zur Identifizierung der Ressource steht. *<ID>* wird für jede Ressource durch eine ID ersetzt, welche die Eindeutigkeit der Ressourcen-URI garantiert. Der genaue Aufbau und die Struktur der URIs werden in Abschnitt 4.3 erläutert. Die Eigenschaften des Entitätstyps werden für Entitäten auf Aussagen abgebildet. Dabei bildet die URI der entsprechenden Ressource das Subjekt der Aussage. Das Prädikat wird für jede Eigenschaft in der semantischen Beschreibung definiert und das Objekt entspricht dem Wert der Eigenschaft der Entität. In der Abbildung sind die Objekte in Form von Fragezeichen als Variablen dargestellt, da sie von den jeweiligen Entitäten abhängen. Abbildung 11 zeigt auf Basis des in Abbildung 10 angegebenen RDF-Templates die Abbildung einer Entität in einen RDF-Graphen.

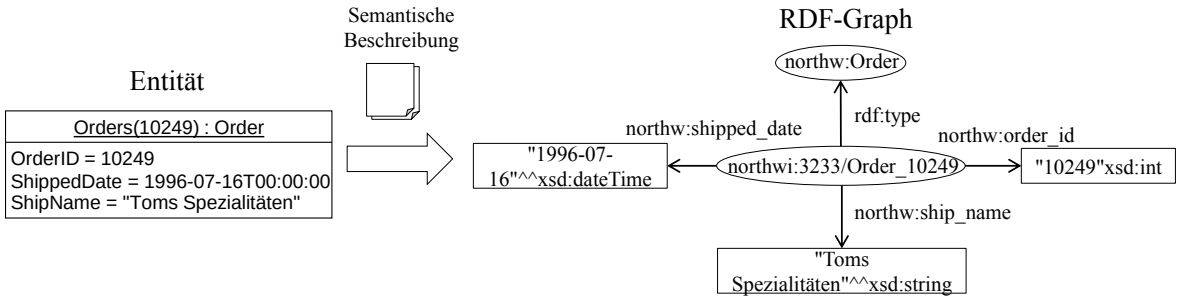


Abbildung 11: Abbildung einer Entität auf einen RDF-Graphen.

Auf der linken Seite ist als Objekt in UML-Notation die Entität *Orders(10249)* des Entitätstyps *Order* dargestellt. Auf der rechten Seite ist der RDF-Graph abgebildet, in der die dargestellte Entität (*Orders(10249)*) anhand des RDF-Graph-Templates, welches in der semantischen Beschreibung definiert ist, überführt wird. Der Graph besteht aus vier Tripeln, wobei die Entität selbst auf die Ressource *northwi:3233/Order_10249* abgebildet wird. Die drei Tripel mit den Prädikaten *northw:order_id*, *northw:shipped_date* und *northw:ship_name* bilden die Eigenschaften der Entität ab. Das Tripel mit dem Prädikat *rdf:type* weist der Ressource den Typ *northw:Order* zu. Die von dem Service zurückgegebene Entität wird somit in genau eine RDF-Ressource überführt, wobei für jede Eigenschaft genau ein entsprechendes Prädikat existiert, auf welches dieses abgebildet wird.

4.2 Semantische Erweiterung von CSDL

Jeder Daten-Service in Form einer Entitätsmenge (*EntitySet*) wird durch eine Kollektion (*Collection*) repräsentiert. Die Referenzierung der zugehörigen Entitätsmenge erfolgt dabei über das *href*-Attribut des *collection*-Elements mittels der URI der Entitätsmenge. Da der Daten-Service durch diese URI eindeutig identifiziert wird, kann es nicht mehrere Kollektionen geben, welche auf die gleiche Entitätsmenge verweisen. Die Semantik der von einem Daten-Service bereitgestellten Daten ist daher unabhängig von den im Service-Dokument gemachten Angaben. Folglich kann auch die Definition der semantischen Beschreibung bzw. der Mapping-Angaben unabhängig von dem Service-Dokument erfolgen. Trotzdem besteht die Möglichkeit, die Abbildung der von den Daten-Services zurückgegebenen Daten nach RDF im Service-Dokument zu definieren. Das hätte aber zur Folge, dass die Entitätstypen, komplexen Typen usw. bei der semantischen Annotation über einen noch zu definierenden Referenzmechanismus, ausgehend von dem Entitätstyp, referenziert werden müssten.

Ein leichtgewichtigerer Ansatz lässt sich realisieren, indem die direkte semantische Annotation des innerhalb des Service-Metadaten-Dokuments bzw. CSDL-Dokuments beschriebenen Entitätsdatenmodells ermöglicht wird. Bei dieser Art der Beschreibung muss allerdings berücksichtigt werden, dass im Rahmen des Entitätsdatenmodells mehrere Entitätsmengen auf die gleichen Entitätstypen verweisen können. Zwar sind zwei Entitätsmengen immer disjunkt, auch dann, wenn sie auf den gleichen Entitätstypen verweisen [134]. Allerdings ist es durchaus vorstellbar, verschiedene Abbildungen auf RDF-Graphen in Abhängigkeit der jeweiligen Entitätsmengen zu definieren. Die zu entwickelnden Konstrukte zur semantischen Beschreibung müssen somit die Definition unterschiedlicher Mappings für bestimmte Entitätstypen in Abhängigkeit der jeweiligen Entitätsmengen erlauben. Für den einfacheren Fall, dass ein bestimmter Entitätstyp von genau einer Entitätsmenge referenziert wird, kann hingegen die semantische Annotation auf Ebene des Entitätstyps erfolgen.

Ausgehend von diesen Überlegungen werden im Folgenden Konzepte zur semantischen Beschreibung hergeleitet. Dabei werden für konkrete Beispiele Abbildungen auf RDF-Graphen definiert, anhand derer die notwendigen Erweiterungen zur semantischen Beschreibung abgeleitet werden. Die Herleitung erfolgt ausgehend von den grundlegenden Konzepten des EDMs, wie Entitätstypen, Eigenschaften und Assoziationen. Diese werden auf RDF-Graphen abgebildet, wobei alle der von RDF bereitgestellten Konstrukte, wie z.B. leere Knoten, Listen usw., berücksichtigt werden. Aufbauend auf den Abbildungen der grundlegenden EDM-Konzepte werden immer umfangreichere Modelle und deren Abbildungen auf RDF-Graphen betrachtet und die dazu notwendigen Erweiterungen von CSDL hergeleitet.

4.3 Aufbau der Ressourcen-URIs

Jede Ressource in einem RDF-Graphen wird über eine URI[21] eindeutig identifiziert. Für jede Entität, die in eine Ressource überführt werden soll, muss daher definiert werden, wie die zugehörige URI erzeugt wird. In dem Beispiel aus Abbildung 11 wurde für die Entität *Orders(10249)* die URI *northwi:Order_10249* erzeugt. Allgemein ist zunächst zu klären, welche Anforderungen an die Struktur der URIs existieren und ob sich daraus Anforderungen an die semantische Beschreibung der Services ableiten lassen. In der RDF-Spezifikation werden verschiedene Angaben zum Aufbau und zur Semantik von URIs gemacht [69, 96, 140]:

- Eine URI hat unabhängig von dem Kontext, in dem sie verwendet wird, immer dieselbe Bedeutung.
- Es werden keine Vorgaben bzgl. der Verwendung von URIs mit einem bestimmten Protokoll, wie z.B. HTTP, gemacht.
- Obwohl die Angabe von relativen URIs in bestimmten Serialisierungsformaten, wie z.B. RDF/XML, erlaubt ist, schreibt RDF die Verwendung absoluter URIs vor, um eine kontextunabhängige Identifikation sicherzustellen.
- Die teilweise syntaktische Übereinstimmung zweier URIs bedeutet nicht, dass die durch diese URIs identifizierten Ressourcen in irgendeiner Beziehung zueinander stehen.

Des Weiteren werden verschiedene Vorgaben hinsichtlich der konkreten Kodierung der URIs gemacht. Es sind Unicode-Zeichen [180] erlaubt, die über den US-ASCII-Zeichensatz hinausgehen. Sie entsprechen somit IRIs [74] bzw. sind zu diesen kompatibel. Die SPARQL-Spezifikation [155] spricht daher auch von IRIs bzw. IRIrefs. Dabei weist die SPARQL-Spezifikation darauf hin, dass bestimmte RDF URIrefs, z.B. URIs, die geschweifte Klammern oder den Backslash enthalten, keine IRIs sind. Die Auswertung von SPARQL-Anfragen gegen RDF-Graphen, die solche URIs enthalten, ist daher undefiniert.

Die Anforderungen an die URIs, die durch die RDF-Spezifikation gemacht werden, müssen bei der Überführung der Daten in einen RDF-Graphen eingehalten werden, andernfalls handelt es sich nicht um korrektes RDF. Um eine von konkreten SPARQL-Prozessoren unabhängige, konsistente Auswertung von SPARQL-Anfragen zu gewährleisten, sollten außerdem die beschriebenen Einschränkungen hinsichtlich der erlaubten Zeichen eingehalten werden. D.h. das System sollte keine in diesem Sinne *ungültigen* URIs erzeugen. Für die semantische Service-Beschreibung ergibt sich damit die Anforderung, dass solche URIs bereits durch die vorgegebenen Konstrukte zur Definition der URI-Erzeugung ausgeschlossen werden sollten.

[21] Wie in Abschnitt 2.1 bereits erwähnt, werden die Begriffe URI und URIref synonym verwendet.

Neben diesen, durch die Spezifikationen vorgegebenen Anforderungen, gibt es verschiedene Empfehlungen hinsichtlich des Aufbaus von URIs in RDF-Graphen. Das W3C empfiehlt in [68], dass es mittels einer gegebenen URI möglich sein sollte, eine Beschreibung über die durch die URI identifizierte Ressource per HTTP abzurufen. In Abhängigkeit des Anfragenden (Mensch oder Maschine) sollte entweder RDF oder eine menschenlesbare Repräsentation, z.B. in HTML, zurückgegeben werden. Auch wenn solch ein Abfragemechanismus außerhalb des Fokus dieser Arbeit liegt, sei darauf hingewiesen, dass dieser, zumindest für RDF, auf Basis des SPARQL-Endpunktes über die *DESCRIBE*-Anweisung sehr leicht realisiert werden kann. Daher ist es sinnvoll, diese Empfehlung bei der Erzeugung der URIs zu berücksichtigen. Dies geschieht, indem als URI-Schema *http* vorgegeben wird und die Host-Angabe frei durch den Service-Provider bestimmt werden kann.

Im Zusammenhang mit der Bereitstellung eines solchen Abfragemechanismus wird in [68] auch auf die sich daraus ergebenden Probleme hingewiesen. Um eine Verwechslung zwischen der Ressource und deren Repräsentation zu vermeiden, sollte eine URI niemals beide repräsentieren. Beispielsweise identifiziert in Abbildung 11 die URI *http://northwind.beispiel.org/Order_10249*[22] eine tatsächlich existierende Bestellung. Wenn über diese URI, z.B. mittels eines Browsers, eine Beschreibung dieser Bestellung in Form einer HTML-Seite abgerufen werden kann, besteht Verwechslungsgefahr. Es ist unklar, ob diese URI die Ressource oder die Webseite, welche diese Ressource beschreibt, identifiziert. Andererseits ist aber dieser Abfragemechanismus, wie bereits erläutert, erwünscht. Um dieses Problem zu umgehen, werden in [68] zwei Lösungen vorgeschlagen.

Die erste Möglichkeit besteht in der Verwendung von Hash-URIs. Bei Hash-URIs wird an die URI mittels einer Raute (#) ein zusätzliches Fragment angehängt. Die URI zur Identifizierung der Bestellung in Abbildung 11 könnte damit z.B. folgendermaßen umgesetzt werden: *http://northwind.beispiel.org/Order#10249*. Da HTTP die Abtrennung des durch die Raute eingeleiteten Fragments fordert, bevor die Anfrage des Servers erfolgen kann, besteht keine Verwechselungsgefahr mehr. Die URI *http://northwind.beispiel.org/Order#10249* identifiziert die Ressource und die URI *http://northwind.beispiel.org/Order* identifiziert das Web-Dokument, das diese Ressource beschreibt. Client-Anwendungen, wie z.B. Browser, entfernen automatisch das Fragment. Daher kann die Abfrage der Beschreibung für den Nutzer transparent über die URI *http://northwind.beispiel.org/Order#10249* erfolgen. Die Angabe des gewünschten Repräsentationsformats (z.B. RDF oder HTML) kann dabei über den HTTP-*Accept*-Header (*content negotiation*) spezifiziert werden.

Als zweite Möglichkeit, dieses Problem zu umgehen, wird in [68] die Verwendung des HTTP-Statuscodes 303 vorgeschlagen. Mit dem Statuscode 303 signalisiert ein Server, dass die angefragte Web-Ressource unter einer anderen URI zu finden ist [80]. Die Idee besteht darin, dass der Server bei der Anfrage nach einer Ressource mit einer 303 antwortet und den Client über eine andere URI auf die Beschreibung der Ressource leitet. Wenn z.B. der Client die URI *http://northwind.beispiel.org/Order_10249* zur Server-Anfrage verwendet, könnte der Server mit dem Statuscode 303 antworten und als Wert des HTTP-*Location*-Headers die URI

[22] Wobei hier das Namensraumpräfix *northw* beispielhaft mit *http://northwind.beispiel.org* aufgelöst wurde.

http://northwind.beispiel.org/doc/Order_10249 übertragen, über welche die Beschreibung der Ressource abgefragt werden kann. Somit besteht keine Verwechslungsgefahr und der Client kann die URI der Ressource zur Abfrage der Beschreibung verwenden. Auch in diesem Fall kann HTTP-Content-Negotiation zur Angabe des gewünschten Repräsentationsformats verwendet werden.

Beide Varianten haben, wie in [68] beschrieben, Vor- und Nachteile . Der Einsatz von Hash-URIs kann dazu führen, dass bei einer Abfrage unnötig viele Daten übertragen werden. Wenn z.B. die bereits erwähnte URI *http://northwind.beispiel.org/Order#10249* zur Identifikation der Bestellung verwendet wird, dann muss der Server bei der Abfrage die Beschreibungen aller Bestellungen zurückgeben, da dieser anhand der URI, die er vom Client erhält, (*http://northwind.beispiel.org/Order*) nicht entscheiden kann, welche Beschreibung benötigt wird. Bei sehr großen oder sehr vielen Datensätzen kann die unnötig übertragene Datenmenge daher sehr hoch sein. Insbesondere in ERP-Systemen, in denen häufig sehr viele Geschäftsobjekte des gleichen Typs verwaltet werden und einzelne Objekte sehr viele Eigenschaften aufweisen, ist die Menge der zu übertragenden Daten bei der Verwendung von Hash-URIs nicht akzeptabel. Andererseits haben Hash-URIs den Vorteil, dass die Anzahl der HTTP-Anfragen bei der Abfrage vieler Ressourcen mit der gleichen Basis-URI viel geringer ist, als bei der Verwendung des 303 Statuscodes. Wenn der Client z.B. neben der Beschreibung der Ressource *http://northwind.beispiel.org/Order#10249* auch die Beschreibungen der Ressourcen *http://northwind.beispiel.org/Order#10250* und *http://northwind.beispiel.org/Order#10251* benötigt, ist nur eine HTTP-Abfrage mit der URI *http://northwind.beispiel.org/Order* erforderlich. Dem gegenüber ständen 6 HTTP-Abfragen bei der Umleitung mittels des 303 Statuscodes. Der Vorteil der Umleitung wäre, dass nur die tatsächlich benötigten Daten übertragen werden, was, wie bereits erwähnt, bei der Verwendung von Hash-URIs nicht notwendigerweise der Fall sein muss.

Um die Vorteile beider Verfahren zu vereinen, wird in [68] eine Kombination beider Ansätze vorgeschlagen. Dazu erhält die Beschreibung einer bestimmten Ressource eine eindeutige URI, wie sie die Ressource bei dem Umleitungsverfahren erhalten würde, z.B.: *http://northwind.beispiel.org/Order_10249*. Die Identifikation der Ressource erfolgt durch das Anhängen eines beliebigen, zusätzlichen Fragments (z.B. *this*) wie beim Hash-Verfahren: *http://northwind.beispiel.org/Order_10249#this*. Obwohl damit die Vorteile beider Ansätze vereint werden, ergibt sich der Nachteil, dass die Struktur der URIs weniger intuitiv ist.

Bei einem Vergleich der drei vorgestellten Verfahren zeigt sich für den speziellen Fall der Abfrage von ERP-Systemen, dass aufgrund der Anzahl und Größe der Datensätze, die in ERP-Systemen typischerweise verwaltet werden, das Hash-Verfahren nicht verwendet werden kann. Stattdessen kommen für diesen speziellen Anwendungsfall nur das Umleitungsverfahren und das kombinierte Verfahren in Betracht. Allerdings kann für andere Anwendungsfälle außerhalb dieser Domäne keine allgemeine Aussage getroffen werden, welcher Ansatz der bessere ist. Um die Unterstützung solcher Anwendungsfälle außerhalb der ERP-Domäne durch die im Rahmen dieser Arbeit entwickelten Konzepte zu gewährleisten, dürfen sich daher hinsichtlich des verwendeten Verfahrens keine Einschränkungen durch die Definition der URI-Erstellung im Rahmen der semantischen Beschreibung ergeben. Somit ergibt sich aus diesen Überlegungen die Anforderung, dass die Konzepte zur Angabe der URI-

Erstellung generisch genug sein müssen, um die Verwendung aller drei Verfahren zu ermöglichen.

Neben diesen Verfahren zur Unterscheidung von URIs zur Identifikation von Ressourcen und deren Beschreibungen werden in [41, 52, 68] weitere Empfehlungen hinsichtlich des Aufbaus von URIs gegeben. URIs sollten so einfach wie möglich sein. D.h. sie sollten möglichst kurz und für den Menschen verständlich sein. Des Weiteren sollten keine Bezeichnungen der Autoren, des Themas, des Status, der technischen Details usw. enthalten sein. URIs sollten zeitlich möglichst stabil und leicht zu verwalten sein. Bei der Definition der Konzepte zur Beschreibung der URI-Erstellung sollten diese Empfehlungen soweit wie möglich berücksichtigt werden.

Eine weitere Anforderung ergibt sich daraus, dass die Entität, auf deren Basis eine Ressource erzeugt wird, durch die URI der Ressource eindeutig bestimmbar sein muss. Wenn z.B. eine SPARQL-Anfrage die URI *http://northwind.beispiel.org/Order_10249* enthält, muss das System in der Lage sein, auf Basis der URI die zugehörige Entität *Orders(10249)* zu ermitteln, um die zur Beantwortung der Abfrage benötigten Daten abrufen zu können. Wie jede RDF-Ressource existiert auch für jede Entität eine URI, welche die Entität eindeutig identifiziert. Es stellt sich daher die Frage, ob die URI der Entität ebenfalls zur Identifikation der zugehörigen RDF-Ressource verwendet werden könnte. Dies würde allerdings wieder zu einer ähnlichen Verwechselungsgefahr führen, wie sie bereits weiter oben für unterschiedliche Repräsentationsformen einer Ressource beschrieben wurde. Des Weiteren muss die Semantik einer Ressource nicht zwangsläufig vollständig mit der Semantik einer Entität übereinstimmen. Stattdessen kann eine Ressource, wie weiter unten noch erläutert wird, auch aus mehreren Entitäten aufgebaut werden. In diesem Fall wäre die Bedeutung der URI kontextabhängig, was den Anforderungen der RDF-Spezifikation [96] widersprechen würde. Daher muss sich die URI der Ressource von der URI der Entität unterscheiden. Allerdings muss die URI der Entität aus der URI der Ressource hergeleitet werden können, um die Abfrage der benötigten Daten zu ermöglichen. Folglich muss die URI der Ressource alle Daten enthalten, die benötigt werden, um die Entität eindeutig zu identifizieren. Dem Entitätsdatenmodell [134, 138] entsprechend, muss jede Entität über einen Entitätsschlüssel verfügen, welcher die Entität in einer Entitätsmenge eindeutig identifiziert. Demzufolge ist der Entitätsschlüssel zwar notwendig, aber nicht ausreichend zur Identifikation einer bestimmten Entität. Zusätzlich ist der Name der Entitätsmenge erforderlich. Die OData-Spezifikation [137] macht zwar keine strikten Vorgaben hinsichtlich des Aufbaus der URIs zur Identifizierung und Abfrage der Entitäten, gibt aber Empfehlungen diesbezüglich ab. Diesen Empfehlungen zufolge wird die URI einer Entität folgendermaßen aufgebaut (Darstellung in erweiterter Backus-Naur-Form):

entityURI = *scheme*, *host*, [":",*port*], *serviceRoot*, "/", *pathPrefix*, "/", [*entityContainer*, "."],
entitySet, "(", *keyPredicate*,")";

Neben dem Entitätsschlüssel (*keyPredicate*) und dem Namen der Entitätsmenge (*entitySet*) muss die URI einer Entität auch den Namen des Entitätscontainers (*entityContainer*) enthalten, indem sich die entsprechende Entitätsmenge befindet, sofern der Entitätscontainer nicht der Standard-Entitätscontainer ist. Die Basis-URI, vor dem Namen des

Entitätscontainers, bestehend aus dem Schema (*schema*), der Host-Angabe (*host*), der Port-Angabe (*port*), der Service-Root-Angabe (*serviceRoot*) und dem Pfad-Präfix (*pathPrefix*), ist für alle Entitäten eines bestimmten Services gleich. Demzufolge muss die URI jeder erzeugten Ressource mindestens folgende Angaben enthalten:

- Wert des Entitätsschlüssels der Entität, aus der die Ressource erzeugt wurde.
- Name der entsprechenden Entitätsmenge.
- Name des entsprechenden Entitätscontainers, sofern es sich nicht um den Standard-Entitätscontainer handelt.
- Basis-URI.

Zusammenfassend ergeben sich folgende Anforderungen und Empfehlungen, die bei der Definition des Templates für die URI-Erzeugung berücksichtigt werden müssen:

1. Globale Eindeutigkeit: Die URI hat unabhängig vom Kontext immer die gleiche Bedeutung [96].
2. Verwendung absoluter URIs [140].
3. Berücksichtigung der erlaubten Zeichen [56] und der zusätzlichen Einschränkungen, die sich diesbezüglich aus der Verwendung von SPARQL ergeben [155].
4. Dereferenzierbarkeit der URI mittels HTTP zum Abruf von Beschreibungen [68].
5. Unterstützung zur Erstellung von URIs für das Hash-Verfahren, das Umleitungsverfahren und das kombinierte Verfahren [68].
6. Berücksichtigung der Empfehlungen nach [41, 52, 68] hinsichtlich Lesbarkeit, Verwaltbarkeit, Einfachheit etc.
7. Die URI der entsprechenden Entität muss aus der Ressourcen-URI hergeleitet werden können.

Nach [45] muss eine absolute URI folgendermaßen aufgebaut sein (Anforderung 2):

URI = scheme ":" *hier-part* ["?" *query*] ["#" *fragment*]

Zunächst muss das Schema der URIs festgelegt werden. Dazu existieren mehrere Möglichkeiten. Entweder wird ein bereits existierendes Schema verwendet oder es wird ein neues Schema definiert. Eine weitere Möglichkeit besteht darin, demjenigen, der die semantische Annotation der Service-Beschreibung vornimmt, die Entscheidung zu überlassen. Wie bereits weiter oben erläutert, wird in [68] die Empfehlung gegeben, dass es mittels einer gegebenen URI möglich sein sollte, eine Beschreibung über die durch die URI identifizierte Ressource per HTTP abzurufen (Anforderung 4). Um die Umsetzung dieser Empfehlung zu unterstützen, wird *http* als Schema für die URIs der RDF-Ressourcen definiert. Als nächstes

muss der *hier-part*-Teil der URI spezifiziert werden. Dieser setzt sich aus der Host-/Port-Angabe und der eigentlichen Pfad-Angabe zusammen. Die Host-Angabe kann nicht fest vorgegeben werden, da sie vom Anwendungsfall und vom Service-Provider abhängt. Sie muss daher vom Nutzer spezifiziert werden. Um das Herleiten der URI der Entität aus der Ressourcen-URI zu ermöglichen (Anforderung 7), muss die Pfad-Angabe die beschriebenen Angaben (Basis-URL, Name des Entitätscontainers, Name der Entitätsmenge, Entitätsschlüssel) enthalten. Allerdings hätte das vollständige Einfügen dieser Daten in die Pfad-Angabe sehr lange und schwer verständliche Ressourcen-URIs zur Folge (Anforderung 6). Dieses Problem kann umgangen werden, indem das System die Kombinationen aus Basis-URIs, Name des Entitätscontainers und Name der Entitätsmenge injektiv auf numerische Werte abbildet (z.B. mittels einer Hash-Funktion) und diese zusammen mit den Elementen, aus denen sie erzeugt wurden, in Form umgekehrter Abbildungen, abspeichert. Diese Werte werden im Folgenden als *Entity-Location-IDs* bezeichnet. Anstatt der Basis-URI, der Name des Entitätscontainers und der Name der Entitätsmenge wird die kürzere Entity-Location-ID in die Ressourcen-URI eingefügt. Für eingehende SPARQL-Anfragen, welche eine Ressourcen-URI enthalten, können die entsprechenden Bezeichnungen der Entitäts-URI mittels dieser ID aus der zugehörigen, umgekehrten Abbildung ermittelt werden.

Aufgrund der Anzahl der typischerweise in ERP-Systemen verwalteten Geschäftsobjekte, ist es nicht möglich, den Entitätsschlüssel mit in die Abbildung auf die *Entity-Location-ID* einzubeziehen. Stattdessen muss der Entitätsschlüssel mit in die Ressourcen-URI aufgenommen werden. Der Entitätsschlüssel setzt sich aus einer Menge von Eigenschaften mit primitiven Typen zusammen [138]. Dabei dürfen die Eigenschaften keine Null-Werte aufweisen (*non-nullable*). Somit müssen die Werte der Entitätsschlüssel der in [45] beschriebenen Kodierung entsprechend in die URI eingefügt werden, wobei bei der Abbildung Anforderung 3 berücksichtigt werden muss.

Um globale Eindeutigkeit (Anforderung 1) auch bei der Erzeugung mehrerer RDF-Ressourcen aus einer Entität sicherzustellen, muss zusätzlich der Name der zugehörigen Ressourcen-Variablen in die URI aufgenommen werden. Die Definition der Ressourcen-Variablen als Teil der semantischen Beschreibung wird in den folgenden Abschnitten erläutert (siehe insbesondere Abschnitt 4.5). Wenn keine Ressourcen-Variablen explizit definiert wurden (siehe dazu Abschnitt 4.4), wird als Variablen-Name der Typ der Ressource ohne Präfix eingesetzt. Zwar ist diese Angabe zur Identifizierung nicht erforderlich, sie erhöht aber die Lesbarkeit (Anforderung 6), da anhand dieser Angabe ersichtlich ist, welcher Typ die Ressource hat. Zusammengefasst ergibt sich folgende URI-Struktur (erweiterte Backus-Naur-Form):

Ressource-URI = "http://", *Host*, [*Path-Prefix*], "/", *Entity-Location-ID*, "/", *Path-Suffix* ;

Der *Path-Prefix* entspricht einer vom Nutzer zur logischen Strukturierung frei wählbaren Teilpfad-Angabe. Das *Path-Suffix* ist in Abhängigkeit des eingesetzten Verfahrens folgendermaßen aufgebaut:

- **Hash-Verfahren**: *Path-Suffix = Ressource_Var*, "#", *Entity-Key*
- **Umleitungsverfahren**: *Path-Suffix = Ressource_Var*, "_", *Entity-Key*
- **Kombiniertes Verfahren**: *Path-Suffix = Ressource_Var*, "_", *Entity-Key*, "#this"

Der *Entity-Key* wird dabei aus den Werten der Eigenschaften des Entitätsschlüssels zusammengesetzt, wobei diese in der Reihenfolge der Auflistung im Service-Metadaten-Dokument aneinander gereiht werden. Die einzelnen Werte werden durch den Schrägstrich getrennt. Mit dieser Definition ergeben sich in Anlehnung an Abbildung 11 folgende Beispiel-URIs:

- *http://northwind.beispiel.org/237421/Order#10249* (Hash-Verfahren)
- *http://northwind.beispiel.org/237421/Order_10249* (Umleitungsverfahren)
- *http://northwind.beispiel.org/237421/Order_10249#this* (kombiniertes Verfahren)

Dabei entspricht der Wert *237421* einer willkürlich gewählten *Entity-Location-ID*. Der Wert *10249* ist der Entitätsschlüssel. Ein *Path-Prefix* ist nicht enthalten.

Da das Schema der URI fest vorgegeben ist und die *Entity-Location-ID* und der Entitätsschlüssel automatisch aus der gegebenen Entität ermittelt werden, können vom Nutzer folgende Elemente frei vergeben werden: *Host*, *Path-Prefix* und *Ressource_Var*. Des Weiteren muss die Möglichkeit bestehen, die Struktur der URI (Hash-, Umleitungs- oder kombiniertes Verfahren) zu spezifizieren. Die semantische Beschreibung muss daher Elemente bereitstellen, um die Angabe dieser Daten zu ermöglichen. Dafür ist zunächst die Definition des entsprechenden Namensraums notwendig, in dem die Elemente und Attribute definiert werden. Zur Identifikation diesen Namensraumes wird folgende URI festgelegt: *http://research.sap.de/OData/SemanticDesc*. Um die abgekürzte Schreibweise zu ermöglichen, wird das zugehörige Namensraumpräfix *sem* definiert.

Die *Host*- und die *Path-Prefix*-Angabe folgen innerhalb der URI aufeinander und sind für alle Ressourcen gleich. Sie können daher über ein einziges Attribut spezifiziert werden: *sem:URI*. Dieses Attribut wird für die CSDL-Elemente *EntityType*, *EntitySet* und *DataServices* definiert. Die URIs der Ressourcen, die aus den Entitäten des entsprechenden Entitätstyps, der Entitätsmenge oder des gesamten Services erzeugt werden, erhalten die als Wert des *sem:URI*-Attributs angegebene *Host*- und *Path-Prefix*-Angabe. Als Beispiel zeigt Listing 28 die Erweiterung des *Order*-Entitätstyps aus dem Service-Metadaten-Dokuments des Northwind-Services mit dem *sem:URI*-Attribut. Die URIs der Ressourcen, die aus den Entitäten des Entitätstyps *Order* erzeugt werden, beginnen daher der Angabe entsprechend mit *http://northwind.beispiel.org*.

Zur Angabe des Typs wird ein weiteres Attribut *sem:Type* für die Elemente *EntityType* und *EntitySet* definiert (siehe Listing 28). Die Struktur der URI wird über das optionale *sem:URI_struc*-Attribut festgelegt, welches die Werte *hash* (Hash-Verfahren), *303* (Umleitungsverfahren) und *combined* (kombiniertes Verfahren) annehmen kann (Anforderung 5). Dieses Element wird ebenfalls für die Elemente *EntityType*, *EntitySet* und

edmx:DataServices definiert. Wenn dieses Element fehlt, wird als Standardwert *hash* angenommen.

```
1. <EntityType Name="Order" sem:Type="northw:Order"
2.  sem:URI="northwind.beispiel.org" sem:URI_struc="303">
3.  …
4. </EntityType>
```

Listing 28: Angabe des Hosts, des Typs und der URI-Struktur für den *Order*-Entitätstyp.

4.4 Abbildung eines Entitätstyps auf ein Ressourcen-Template

Im vorherigen Abschnitt wurde der Aufbau der Ressourcen-URIs hergeleitet und die zur Angabe der frei wählbaren URI-Abschnitte benötigten Attribute definiert. Im Folgenden werden die zur Beschreibung der Abbildung eines Entitätstyps auf ein Ressourcen-Template benötigten Attribute hergeleitet. Der Begriff *Ressourcen-Template* bezeichnet dabei ein RDF-Graph-Template, welches Ressourcen genau eines Typs beschreibt. Die Herleitung erfolgt anhand des bereits aus Abbildung 11 bekannten Beispiels. In diesem Beispiel wird aus jeder Entität des Entitätstyps *Order* eine RDF-Ressource des Typs *northw:Order* erzeugt. Dabei wird jede Eigenschaft der Entität auf genau eine Aussage mit dem entsprechenden Prädikat abgebildet. Somit ergibt sich das in Listing 29 dargestellte Ressourcen-Template (in erweiterter Turtle-Notation), auf den die Entitätsmenge abgebildet wird. Wobei die Variablen durch die konkreten Werte der Eigenschaften der einzelnen Entitäten ersetzt werden. Ein einzelnes Tripel des Ressourcen-Templates wird im Folgenden auch als *Tripel-Template* oder *Aussagen-Template* bezeichnet, da es die Vorlage für eine Menge von RDF-Tripeln bildet.

```
1. ?order rdf:type northw:Order .
2. ?order northw:order_id ?orderID .
3. ?order northw:shipped_date ?shippedDate .
4. ?order northw:ship_name ?shipName .
```

Listing 29: Ein Ressourcen-Template auf Basis des *Order*-Entitätstyps.

4.4.1 Typ-Angaben und konkrete Aussagen

Über die in Abschnitt 4.3 definierten Elemente kann spezifiziert werden, wie die URIs der *Order*-Ressourcen erzeugt werden sollen bzw. wie diese aufgebaut sind. Des Weiteren kann mit dem *sem:Type*-Attribut der Typ der Ressourcen angegeben werden. Damit ist die *order*-Variable und die Typ-Angabe in Zeile 1 aus Listing 29 beschrieben. Es fehlt die Beschreibung der Aussagen in den Zeilen 2 bis 4. In dem dargestellten Beispiel werden die Eigenschaften *OrderID*, *ShippedDate* und *ShipName* (siehe Listing 30) auf Aussagen mit den Prädikaten

order_id, *shipped_date* und *ship_name* abgebildet. Es muss somit angegeben werden, dass diese Aussagen aus diesen Eigenschaften erzeugt werden.

```
1. <EntityType Name="Order" sem:Type="northw:Order"
2.  sem:URI="northwind.beispiel.org" sem:URI_struc="303">
3.  <Key>
4.   <PropertyRef Name="OrderID" />
5.  </Key>
6.  <Property Name="OrderID" Type="Edm.Int32"
7.   sem:Mapping="northw:order_id"/>
8.  <Property Name="ShippedDate" Type="Edm.DateTime"
9.   sem:Mapping="northw:shipped_date"/>
10. <Property Name="ShipName" Type="Edm.String"
11.  sem:Mapping="northw:ship_name"/>
12. …
13.</EntityType>
```

Listing 30: Beschreibung der Abbildung einzelner Eigenschaften auf konkrete Aussagen.

Zur Angabe der Abbildung einzelner Eigenschaften auf Aussagen wird ein neues Attribut *sem:Mapping* für das *Property*-Element definiert (siehe Listing 30). Als Wert erhält dieses Element die URI des entsprechenden Prädikats. Mit diesen Erweiterungen ist die Beschreibung des RDF-Graph-Templates vollständig. Das in Zeile 1 von Listing 30 dargestellte *sem:Type*-Attribut gibt an, dass für jede Entität des entsprechenden Entitätstyps eine Ressource des Typs *northw:Order* erzeugt wird. Die URI der Ressource wird, wie in Abschnitt 4.3 beschrieben, aus dem Wert des *sem:URI*-Attributs, einer automatisch generierten Entity-Location-ID und dem Entitätsschlüssel (*OrderID*) erzeugt: *http://northwind.beispiel.org/237421/Order_10249*. Jede Eigenschaft, die um das Attribut *sem:Mapping* erweitert wurde, wird auf eine Aussage mit dem Wert des Attributs als Prädikat abgebildet. Das Subjekt der Aussage ist dabei jeweils die automatisch generierte Ressourcen-URI. Der Wert der Eigenschaft der jeweiligen Entität bildet das Objekt der Aussage. Für die Abbildung der Werte der Eigenschaften auf Literale muss noch der Typ der Literale definiert werden. Dies wird im folgenden Abschnitt erläutert.

Es sei der Vollständigkeit halber darauf hingewiesen, dass dynamische Eigenschaften (*dynamic properties*) der CSDL-Spezifikation entsprechend nicht im CSDL-Dokument beschrieben werden. Daher können sie auch nicht in die Abbildung auf RDF einbezogen werden.

4.4.2 Datentypen und Sprachangaben

RDF definiert mit Ausnahme des *rdf:XMLLiteral*-Datentyps keine eigenen Datentypen [140]. Es ermöglicht aber die Verwendung von in anderen Namensräumen definierten Datentypen. Daher existieren mehrere Möglichkeiten für die Typisierung der Literale:

- **Untypisierte Literale:** Die Literale sind untypisiert, d.h. ihnen wird kein Datentyp zugewiesen.
- **Datentypen des Entitätsdatenmodells:** Es werden die Datentypen des Entitätsdatenmodells aus dem CSDL-Dokument übernommen.
- **XSD-Datentypen:** Aufgrund ihres Bekanntheitsgrades wird in der RDF-Spezifikation die Verwendung der XSD-Datentypen [130] empfohlen [56].

Der RDF-Primer [140] weist darauf hin, dass die Verwendung von typisierten Literalen den Informationsaustausch zwischen zwei Anwendungen erleichtert und somit die Interoperabilität erhöht. Daher ist die Verwendung untypisierter Literale nicht empfehlenswert. Es bietet sich stattdessen an, die im Namensraum des Entitätsdatenmodells definierten Datentypen der Eigenschaften, wie z.B. *Edm.Int32*, *Edm.DateTime* und *Edm.String* in Listing 30, zu übernehmen. Die RDF-Spezifikation [56, 140] weist aber darauf hin, dass Datentypen zur Verwendung in RDF bestimmte Eigenschaften erfüllen müssen. So muss der Datentyp einen genau definierten Wertebereich und einen genau definierten lexikalischen Bereich aufweisen. Der Wertebereich enthält die Menge aller Werte, die mit diesem Datentyp repräsentiert werden können und der lexikalische Bereich enthält die Menge aller Zeichenketten, mit denen diese Werte ausgedrückt werden können. Des Weiteren muss eine Abbildung existieren, die jeder Zeichenkette aus dem lexikalischen Bereich einen Wert aus dem Wertebereich zuweist (*lexical-to-value mapping*).

Zwar ist für die im Namensraum des Entitätsdatenmodells definierten Datentypen der Wertebereich angegeben [138], es fehlt jedoch die Definition der lexikalischen Bereiche und der entsprechenden Abbildungen. Des Weiteren wird in [134] darauf hingewiesen, dass es sich bei den Datentypen nur um Proxies handelt, welche als Platzhalter für die im System tatsächlich verwendeten Datentypen dienen. Ein weiterer Nachteil besteht darin, dass die Datentypen des Entitätsdatenmodells im Gegensatz zu den XSD-Datentypen von vielen RDF-Datenbanken und APIs, wie z.B. Jena [178] und Sesame [23], nicht unterstützt werden. Aus diesen Gründen können die Datentypen des Entitätsdatenmodells in RDF nicht verwendet werden. Stattdessen werden der Empfehlung der RDF-Spezifikation folgend die XSD-Datentypen eingesetzt. Dazu muss eine Abbildung der Datentypen des Entitätsdatenmodells auf die XSD-Datentypen definiert werden. Dabei muss jeder Zeichenkette des lexikalen Bereichs eines EDM-Datentyps auf die entsprechende Zeichenkette des lexikalen Bereichs des zugehörigen XSD-Datentyps abgebildet werden. Das Problem besteht darin, dass die lexikalen Bereiche der EDM-Datentypen nicht definiert sind. Daher kann nur eine Abbildung unter der Annahme angegeben werden, dass die lexikalen Bereiche der EDM-Datentypen Teilmengen der lexikalen Bereiche der XSD-Datentypen sind und die Abbildung auf die Werte, wie sie in der XSD-Spezifikation definiert ist, den entsprechenden Werten im Sinne der EDM-Datentypen entspricht. Basierend auf dieser Annahme wird die in Tabelle 5 angegebene Abbildung definiert.

EDM-Datentyp	XSD-Datentyp	EDM-Datentyp	XSD-Datentyp
Binary	base64Binary	Double	double
Boolean	boolean	Guid	string
Byte	byte	Int16	short
DateTime	dateTime	Int32	int
Time	time	Int64	long
DateTimeOffset	dateTime	SByte	short
Decimal	decimal	String	string
Single	float		

Tabelle 5: Abbildung der EDM-Datentypen auf die XSD-Datentypen.

Der Datentyp *Binary* aus dem EDM-Namensraum hat keine direkte Entsprechung im XSD-Namensraum. Da aber im Rahmen dieser Arbeit OData-Services betrachtet werden, welche Atom und XML als Nachrichtenformat verwenden, ist es wahrscheinlich, dass die entsprechenden Binärdaten in Base64-Kodierung [102] vorliegen. Anderenfalls muss über das weiter unten erläuterte Verfahren ein anderer Datentyp angegeben werden. Mit der in Tabelle 5 dargestellten Abbildung der Datentypen ergibt sich für die in Abbildung 11 dargestellte Entität der RDF-Graph aus Listing 31.

```
1. <http://northwind.beispiel.org/237421/Order_10249> rdf:type
2.  northw:Order ;
3.   northw:order_id "10249"^^xsd:int ;
4.   northw:shipped_date "1996-07-16T00:00:00"^^xsd:dateTime ;
5.   northw:ship_name "Toms Spezialitäten"^^xsd:string .
```

Listing 31: Aus einer Entität erzeugter RDF-Graph mit typisierten Literalen.

Um die Abbildung auf untypisierte Literale und Literale mit Datentypen außerhalb des XSD-Namensraums zu ermöglichen, wird ein weiteres Attribut *sem:Datatype* definiert. Als Wert erhält dieses Attribut die Typ-Bezeichnung in Form einer URI oder den Wert *untyped* für die Abbildung auf ein untypisiertes Literal. Das Attribut ist für das CSDL-Element *Property* definiert. Um die Abbildung auf untypisierte Literale für einzelne Entitätsmengen, einzelne Entitätstypen und den gesamten Service festlegen zu können, ist das *sem:Datatype*-Attribut für den Wert *untyped* auch für das *EntitySet*-, *EntityType*- und das *edmx:DataServices*-Element definiert.

Wenn der lexikalische Bereich der Eigenschaft nicht Teilmenge des lexikalischen Bereichs des angegebenen Datentyps ist, dann muss von dem System eine entsprechende Konvertierung durchgeführt werden. Das gleiche gilt auch, wenn der lexikalische Bereich zwar eine Teilmenge ist, die Abbildung auf den Wertebereich des Zieldatentyps aber nicht der gewünschten Abbildung entspricht. Listing 32 zeigt als Beispiel die Angabe eines anderen Typs mit dem *sem:Datatype*-Attribut. Als Datentyp des Objektwertes wurde *ownns:date* spezifiziert. Diese URI steht für einen benutzerdefinierten Datentyp zur Repräsentation von Datumsangaben. Im Gegensatz zum *xsd:dateTime*-Datentyp schreibt dieser Datentyp folgende Formatierung für die Datums- und Zeitangabe vor: *hh:mm:ss – tt/mm/jjjj*. Daher muss bei der

Erstellung des RDF-Graphen eine Konvertierung durchgeführt werden. Dieser kann vom System z.B. auf Basis Anwendungsfall-spezifischer Plug-ins durchgeführt werden.

```
1. <Property Name="ShippedDate" Type="Edm.DateTime"
2.  sem:Mapping="northw:shipped_date"
3.  sem:Dataype="ownns:date"/>
```

Listing 32: Spezifikation eines Typs außerhalb des XSD-Namensraums.

Die RDF-Aussage, die auf Basis der Beschreibung in Listing 32 für das bereits bekannte Beispiel erzeugt wird, ist in Listing 33 dargestellt.

```
1. <http://northwind.beispiel.org/237421/Order_10249>
2.  northw:shipped_date "00:00:00 - 16/07/1996"^^ownns:date .
```

Listing 33: Literal mit einem Typ außerhalb des XSD-Namensraums.

Als Alternative zu der in Listing 32 dargestellten Angabe über das *sem:Datatype*-Attribut und in Vorbereitung auf die in den folgenden Abschnitten erläuterte Abbildung einer Eigenschaft auf mehrere Aussagen-Templates wird noch eine weitere Schreibweise definiert. Diese wird an die in Listing 33 dargestellte Turtle-Syntax angelehnt. Wobei anstatt des Objektwertes ein Platzhalter definiert werden muss. Wenn dieser in Anlehnung an die alternative Kennzeichnung von Variablen in der SPARQL-Syntax mit dem Dollarzeichen gekennzeichnet wird, dann ergibt sich für die Typ-Angabe in Listing 32 folgende alternative Schreibweise: *sem:Mapping="northw:shipped_date $^^ownns:date"*.

Wie in Abschnitt 2.1 beschrieben, können Objekten neben Datentypen auch Sprachinformationen zugewiesen werden. Um bei der Beschreibung die Angabe einer Sprache für Objekte bestimmter Aussagen zu ermöglichen, wird ein weiteres Attribut *sem:Lang* für das *Property*-Element definiert. Als Wert dieses Attribut können in Einklang mit der SPARQL-Spezifikation die in [149] definierten Tags verwendet werden. Listing 34 zeigt ein Beispiel.

```
1. <Property Name="ShipName" Type="Edm.String"
2.  sem:Mapping="northw:ship_name" sem:Lang="de"/>
```

Listing 34: Spezifikation der Sprache.

Bei der Angabe muss berücksichtigt werden, dass diese in RDF nur für untypisierte Literale erlaubt ist [99]. Die gleichzeitige Angabe des Datentyps und der Sprache ist daher nicht möglich. Somit impliziert das Vorhandensein des *sem:Lang*-Attributs immer die Abbildung auf ein untypisiertes Literal. Listing 35 zeigt das Ergebnis der Abbildung für das bereits bekannte Beispiel auf Basis der Beschreibung aus Listing 34.

```
1. <http://northwind.beispiel.org/237421/Order_10249>
2.  northw:ship_name "Toms Spezialitäten"@de .
```

Listing 35: Literal mit Sprachangabe.

Aus den gleichen Gründen wie bei der Definition einer alternativen Schreibweise für die Angabe des Datentyps, soll auch zur Spezifikation der Sprachangabe eine alternative Schreibweise definiert werden. Dazu wird in Anlehnung an die Turtle-Syntax mit dem @-Symbol folgend auf das Dollarzeichen als Platzhalter für den Objektwert die Sprache definiert. Damit ergibt sich für die Sprachangabe aus Listing 34 folgende alternative Angabe: *sem:Mapping="northw:ship_name $@de"*.

4.5 Abbildung eines Entitätstyps auf mehrere Ressourcen-Templates

Im vorherigen Abschnitt wurden die Attribute zur semantischen Beschreibung von OData-Services hergeleitet, die notwendig sind, um Entitäten einer Entitätsmenge bzw. eines Entitätstyps auf Ressourcen genau eines Typs abzubilden. Wenn mehrere Ressourcen unterschiedlichen Typs aus einer Entität erzeugt werden sollen, dann ist einer Erweiterung der in Abschnitt 4.4 vorgestellten Konzepte notwendig. Diese Erweiterungen werden im Folgenden anhand des Beispiels aus Abbildung 12 hergeleitet.

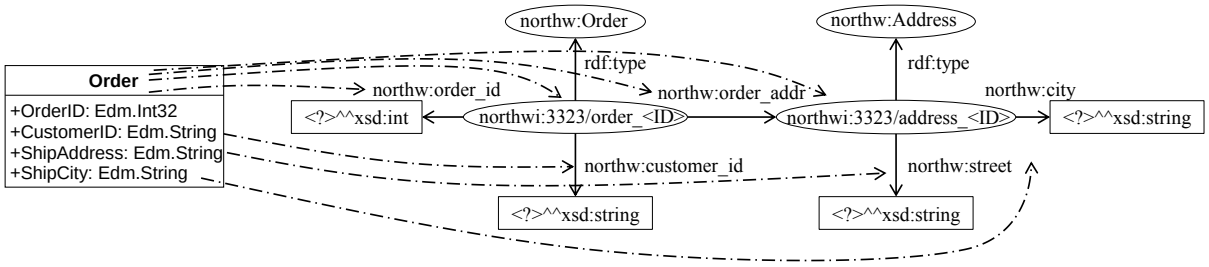


Abbildung 12: Abbildung eines Entitätstyps auf mehrere Ressourcen-Templates.

Abbildung 12 zeigt die Überführung des bereits bekannten *Order*-Entitätstyps auf zwei Ressourcen-Templates, welche miteinander in Beziehung stehen. Der dargestellte Entitätstyp enthält neben der Bestellungs-ID (*OrderID*) und Kunden-ID (*CustomerID*) verschiedene Angaben über den Empfänger der Bestellung (*ShipAddress* und *ShipCity*). Diese Angaben sollen auf ein zweites Ressourcen-Template (*northwi:3323/address_<ID>*) abgebildet werden, welches von der Ressource *northwi:3323/order_<ID>* referenziert wird. Mit den in Abschnitt 4.4 eingeführten Konzepten ist eine solche Abbildung nicht beschreibbar, da diese nur die Abbildung eines Entitätstyps auf genau ein Ressourcen-Template erlauben.

Um eine Abbildung auf mehrere Ressourcen zu ermöglichen, muss die Möglichkeit bestehen, anzugeben, wie die verschiedenen Eigenschaften des Entitätstyps den verschiedenen Ressourcen zugewiesen werden. Dazu wird das Konzept der Ressourcen-Variablen eingeführt. Eine *Ressourcen-Variable* (im Folgenden auch als *Variable* bezeichnet) steht als Platzhalter für die Ressourcen eines bestimmten Typs. Sie wird als Teil einer Typ-Anweisung (*sem:Type-*

Attribut) mit dem aus SPARQL bekannten Formalismus angegeben. Listing 36[23] zeigt, angelehnt an das Beispiel aus Abbildung 12, die Definition zweier Variablen: *?order* und *?address*. Die Variable *?order* bekommt den Typ *northw:Order* zugewiesen und die Variable *?address* erhält den Typ *northw:Address*. Wie an dem Beispiel zu erkennen ist, erfolgt die Zuweisung des Typs über das Gleichheitszeichen. Mehrere Typzuweisungen werden durch einen Punkt voneinander getrennt. Es handelt sich damit um eine Erweiterung des *sem:Type*-Attributs. Wobei die Typ-Anweisung für eine einzelne Ressource, wie sie in Abschnitt 4.4 beschrieben wurde, erhalten bleibt.

```
1. <EntityType Name="Order" sem:Type="?order=northw:Order .
2.  ?address=northw:Address">
3.  …
4. </EntityType>
```

Listing 36: Definition von Ressourcen-Variablen.

Typ-Angaben, wie in Listing 36 dargestellt, beschreiben die Abbildung einer Entität auf zwei Ressourcen. Für jede Typzuweisung wird aus jeder Entität des entsprechenden Entitätstyps eine Ressource des angegebenen Typs erzeugt. Wobei sich die URIs der einzelnen Ressourcen nur durch die Angabe der Ressourcen-Variablen unterscheiden. Die *Entity-Location-ID* und der Entitätsschlüssel bleiben gleich. Damit ist sichergestellt, dass für eine gegebene Ressourcen-URI die URI zum Abruf der entsprechenden Entität ermittelt werden kann, über die alle Daten, die zur Erzeugung der Aussagen über die Ressource notwendig sind, abgerufen werden können.

Neben der Variablen-spezifischen Typ-Zuweisung müssen die Konzepte aus Abschnitt 4.4 so erweitert werden, dass sie die Zuweisung der Eigenschaften zu den verschiedenen Ressourcen ermöglichen. Um das Mapping von Eigenschaften auf RDF-Aussagen zu definieren, wurde in Abschnitt 4.4 das Attribut *sem:Mapping* für das CSDL-Element *Property* eingeführt. Dieses erhält als Wert die URI des Prädikats der Aussage, auf welches die Eigenschaft abgebildet werden soll. Wenn aus einer Entität mehrere Ressourcen erzeugt werden sollen, muss das *sem:Mapping*-Attribut um die Variable des entsprechenden Ressource-Templates, welches in der Typ-Anweisung definiert wurde, erweitert werden. Angelehnt an die erweiterte Turtle-Notation wird die Variablenbezeichnung vor die URI des Prädikats platziert. Listing 37 zeigt, mit Bezug auf die in Abbildung 12 dargestellte Abbildung, die entsprechenden *sem:Mapping*-Attribute. Wie bei der Typ-Angabe bleibt auch für das *sem:Mapping*-Attribut der vereinfachte Formalismus, wie in Abschnitt 4.4 bei der Abbildung auf einen einzelnen Ressourcen-Typ beschrieben, erhalten.

[23] Das *sem:URI*- und das *sem:URI_struc*-Attribut werden in diesem und den folgenden Beispielen aus Gründen der Übersichtlichkeit nicht mehr explizit angegeben.

```
1. <EntityType Name="Order" sem:Type="?order=northw:Order .
2.  ?address=northw:Address"
3.  sem:Mapping="?order northw:order_addr ?address">
4.  …
5.  <Property Name="OrderID" Type="Edm.Int32"
6.   sem:Mapping="?order northw:order_id"/>
7.  <Property Name="CustomerID" Type="Edm.String"
8.   sem:Mapping="?order northw:customer_id" />
9.  <Property Name="ShipAddress" Type="Edm.String"
10.  sem:Mapping="?address northw:street"/>
11. <Property Name="ShipCity" Type="Edm.String"
12.  sem:Mapping="?address northw:city"/>
13. …
14.</EntityType>
```

Listing 37: Semantische Erweiterung der Eigenschaften bei der Abbildung auf mehrere Ressourcen-Templates.

Neben der Abbildung der Eigenschaften auf Aussagen der entsprechenden Ressourcen muss die Beziehung zwischen den Ressourcen des Typs *northw:Order* und des Typs *northw:Address* beschrieben werden. Für jede Entität muss die Aussage *?order northw:order_addr ?address* erstellt werden. Um eine solche Angabe zu ermöglichen, wird das *EntityType*-Element um das *sem:Mapping*-Attribut erweitert. Dabei erlaubt das *sem:Mapping*-Attribut in diesem Fall die Angabe beliebiger Tripel-Templates. Die Vorrausetzung ist, dass die verwendeten Variablen in dem *sem:Type*-Attribut definiert wurden. Für jede Entität werden die im *sem:Mapping*-Attribut des entsprechenden *EntityType*-Element beschriebenen Tripel erstellt, wobei die Variablen durch die automatisch erstellten URIs ersetzt werden. Zeile 3 in Listing 37 zeigt die Angabe des *sem:Mapping*-Attributs für das bereits erwähnte Beispiel. Mit diesen Angaben ist die Abbildung, wie sie in Abbildung 12 dargestellt ist, vollständig beschrieben.

Da entsprechend den Erläuterungen dieses Abschnitts, das *sem:Mapping*-Attribut dahingehend erweitert wird, dass die Angabe beliebiger Tripel-Templates erlaubt ist, kann es auch zur Angabe von Typ-Zuweisungen verwendet werden. Eine Typ-Angabe der Form *sem:Type="?order=northw:Order"* ist eine abkürzende Schreibweise für das Tripel-Template *?order rdf:type northw:Order*. Daher kann dieses Tripel-Template auch über das *sem:Mapping*-Attribut spezifziert werden, was den gleichen Effekt wie die Angabe über das *sem:Type*-Attribut hat.

4.6 Abbildung mehrerer Eigenschaften auf ein Aussagen-Template

Ausgehend von den im vorherigen Abschnitt vorgestellten Konzepten kann eine Erweiterung definiert werden, um die Abbildung mehrerer Eigenschaften eines Entitätstyps auf ein Aussagen-Template zu definieren. Abbildung 13 zeigt ein Beispiel. Die beiden Eigenschaften *LastName* und *FirstName* des Entitätstyps *Employee* werden auf Aussagen mit dem Prädikat

northw:name abgebildet. Dabei soll das Objekt der Aussage aus den durch ein Leerzeichen getrennten Werten der Eigenschaften zusammengesetzt werden.

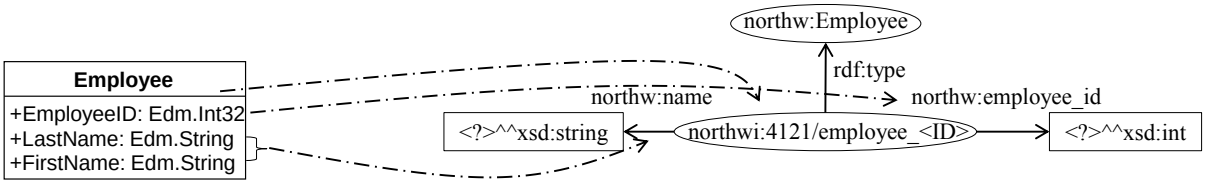


Abbildung 13: Zwei Eigenschaften werden auf eine Aussage abgebildet.

Mit den bereits vorgestellten Konzepten kann eine solche Abbildung nicht beschrieben werden, da diese nur die Abbildung einer einzelnen Eigenschaft auf eine Aussage erlauben. Stattdessen muss eine Erweiterung definiert werden, welche die Angabe mehrerer Eigenschaften als Objekt einer Aussage ermöglicht. Eine Möglichkeit bestände darin, diese Angabe als Teil der semantischen Erweiterung einer einzelnen Eigenschaft zu definieren und dabei über einen noch zu definierenden Referenzmechanismus auf die anderen Eigenschaften zu verweisen. Da für diese Beschreibung jede der Eigenschaften, die Teil der Aussage sein sollen, in Betracht gezogen werden kann, die Beschreibung aber als Teil der semantischen Erweiterung nur für eine Eigenschaft angegeben wird, wäre diese Art der Beschreibung wenig intuitiv und schwer verständlich. Um diese Art der Angabe und die Replikation von Angaben zu vermeiden, kann die Beschreibung der Abbildung im Rahmen der semantischen Erweiterung des jeweiligen Entitätstyps erfolgen. Dazu muss ein Mechanismus definiert werden, welcher die Referenzierung der für die Abbildung relevanten Eigenschaften erlaubt. Listing 38 zeigt als Beispiel die semantische Beschreibung mit Hinblick auf Abbildung 13.

```
1. <EntityType Name="Employee"
2.  sem:Type="?employee=northw:Employee"
3.  sem:Mapping="?employee northw:name '$FirstName $LastName'"
4.  >
5.  <Property Name="EmployeeID" Type="Edm.Int32"
6.   sem:Mapping="?employee northw:employee_id" />
7.  <Property Name="LastName" Type="Edm.String" />
8.  <Property Name="FirstName" Type="Edm.String" />
9. </EntityType>
```

Listing 38: Ein Referenzmechanismus für Eigenschaften.

Die Abbildung der beiden Eigenschaften wird in dem *sem:Mapping*-Attribut des *EntityType*-Elements definiert. Die Referenzierung der Eigenschaften erfolgt, angelehnt an die in SPARQL verwendete erweiterte Turtle-Notation, in Form einer Variablen. Um eine Unterscheidung zu Ressourcen-Variablen zu ermöglichen, wird anstatt des Fragezeichens das Dollarzeichen verwendet, welches dem SPARQL-Standard [155] entsprechend ebenfalls zur Kennzeichnung von Variablen verwendet werden kann. Auf das Dollarzeichen folgt der Name der Variablen, womit eine eindeutige Referenzierung gegeben ist. Da eine Abbildung der

Werte auf Strings erfolgt, werden die durch ein Leerzeichen getrennten Variablen der Turtle-Syntax entsprechend durch Apostrophe eingeschlossen. Hierbei werden die aneinandergereihten Variablen und das Leerzeichen auf eine Konkatenation der entsprechenden Strings abgebildet. Für andere Datentypen, wie z.B. Integer, Float usw., ist eine solche Abbildung in der Regel nicht sinnvoll. Stattdessen müssen andere Arten der Verknüpfung, z.B. Addition, Subtraktion usw., bereitgestellt werden. Dabei ist es sinnvoll, auf bereits existierenden Standard aufzusetzen. Für die XSD-Datentypen existiert mit *XQuery 1.0 and XPath 2.0 Functions and Operators* [131] eine Empfehlung für eine Menge von Funktionen und Operatoren, die viele Standard-Operationen abdecken. Diese Funktionen können für die Beschreibung der Art der Verknüpfung in der semantischen Erweiterung übernommen werden. Damit ist die Angabe in Zeile 3 aus Listing 38 eine abkürzende Schreibweise für: *?employee northw:name fn:concat($FirstName,' ',$LastName)*. Dabei steht das Namensraum-Präfix *fn* für den in [131] definierten Namensraum *http://www.w3.org/2005/xpath-functions*. Die Werte der Eigenschaften werden vor der Ausführung der Operationen den Erläuterungen aus Abschnitt 4.4.2 entsprechend in XSD-Datentypen überführt.

Als Beispiel mit einem anderen Datentyp zeigt Listing 39 die Berechnung des Alters, in dem der Angestellte eingestellt wurde. Zunächst wird das Geburtsdatum (*BirthDate*) von dem Einstellungsdatum (*HireDate*) mittels der Operator-Funktion *op:subtract-dateTimes* subtrahiert. Das *op*-Namensraum-Präfix hat dabei der Spezifikation [131] entsprechend keinen Namensraum zugewiesen. In [131] wird darauf hingewiesen, dass Implementierungen diese Funktionen nicht notwendigerweise direkt zur Verfügung stellen müssen. Sie dienen nur zur Beschreibung der Operatoren. Die Operator-Funktion *op:subtract-dateTimes* entspricht dabei dem Minus-Operator. Somit kann die in den Zeilen 3 und 4 angegebene semantische Beschreibung auch folgendermaßen angegeben werden: *northw:hired_at_age fn:years-from-duration($HireDate-$BirthDate")*. Wenn keine Verwechslungsgefahr besteht, wird im Folgenden die Operator-Schreibweise verwendet. Ansonsten wird die Operator-Funktion explizit angegeben. Das Ergebnis dieses Operators ist die Differenz der beiden Daten in Form des *xsd:dayTimeDuration*-Datentyps. Diese wird an die *fn:years-from-duration*-Funktion übergeben, welche die Jahre zurückgibt, die dem Alter entsprechen, in dem der Angestellte eingestellt wurde.

```
1. <EntityType Name="Employee"
2.  sem:Type="northw:Employee"
3.  sem:Mapping="northw:hired_at_age fn:years-from-
4.   duration(op:subtract-dateTimes($HireDate,$BirthDate))">
5.  <Property Name="BirthDate" Type="Edm.DateTime" />
6.  <Property Name="HireDate" Type="Edm.DateTime" />
7. </EntityType>
```

Listing 39: Berechnung eines Objekts unter Verwendung von XQuery- und XPath-Funktionen.

Die beiden Beispiele in diesem Abschnitt zeigen, wie XQuery- und XPath-Funktionen verwendet werden können, um die Objekte einer Aussage zur Laufzeit berechnen zu können. Es sei darauf hingewiesen, dass eine konkrete Implementierung der hier vorgestellten Konzepte auch andere Funktionen und Operatoren unterstützen können. Die XQuery- und XPath-Funktionen dienen hier nur als Beispiel, da sie für die häufig im Zusammenhang mit RDF verwendeten XSD-Datentypen definiert sind und viele der in Praxis vorkommenden Anwendungsfälle abdecken.

4.7 Abbildung auf leere Knoten

Wie in Abschnitt 2.1 erläutert, erlaubt RDF die Verwendung sogenannter leerer Knoten. Leere Knoten sind Ressourcen, die keine URI zur Identifizierung aufweisen. Sie können als Subjekt oder Objekt, jedoch nicht als Prädikat innerhalb einer RDF-Aussage vorkommen. Sie werden häufig zur Strukturierung von RDF-Graphen eingesetzt, um die Erzeugung zusätzlicher Platzhalter-URIs einzusparen. Um die Abbildung von Entitäten bzw. Teilen von Entitäten auf leere Knoten zu ermöglichen, muss die semantische Beschreibung Konzepte bereitstellen, mit der sich solche Abbildungen beschreiben lassen. Diese sollen im Folgenden hergeleitet werden. Als Beispiel dient die in Abbildung 14 dargestellte Überführung eines Entitätstyps in ein Ressourcen-Template und dem Template eines leeren Knotens, welcher von dem Ressourcen-Template referenziert wird. Abbildung 14 entspricht dem Beispiel aus Abbildung 12 mit dem Unterschied, dass das Template der Ressource zur Repräsentation der Adresse durch einen leeren Knoten ersetzt wurde.

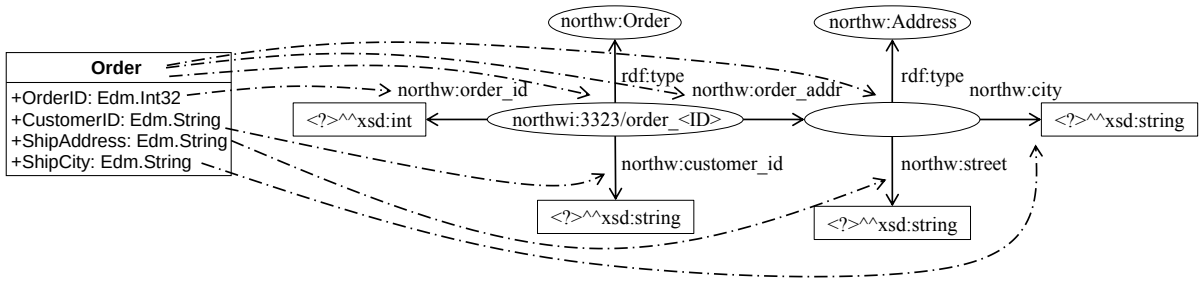


Abbildung 14: Abbildung eines Entitätstyps auf das Template einer Ressource und eines leeren Knotens.

Leere Knoten bekommen keine URIs zur eindeutigen Identifikation zugewiesen. Um sie aber innerhalb eines RDF-Graphen von anderen leeren Knoten unterscheiden zu können, erhalten sie in den meisten Serialisierungsformaten eine Knoten-ID. Neben der Unterscheidung von anderen leeren Knoten dienen die Knoten-IDs dazu, sie innerhalb eines bestimmten RDF-Graphen referenzieren zu können. Um die Referenzierung innerhalb eines Service-Metadaten-Dokuments zu ermöglichen, ist daher im Rahmen der semantischen Beschreibung ebenfalls die Zuweisung einer Knoten-ID notwendig. Die RDF-Spezifikation [56] weist darauf hin, dass die Definition dieser Knoten-IDs von dem verwendeten Serialisierungsformat abhängt und macht keine Vorgaben bzgl. der Struktur. Als Syntax für die Knoten-ID empfiehlt sich, der bisherigen Verwendung entsprechend, die Turtle-Syntax [40]. Wie in Abschnitt 2.1.1 erläutert, werden in Turtle Knoten-IDs durch einen Unterstrich, gefolgt von einem

Doppelpunkt, eingeleitet. Darauf folgt das Label des leeren Knotens. Für die bisher vorgestellten Konzepte und Attribute wird definiert, dass leere Knoten überall dort verwendet werden können, wo auch die Verwendung von Variablen erlaubt ist. Ausgehend von diesen Erläuterungen ist in Listing 40 die semantische Beschreibung für das Beispiel aus Abbildung 14 dargestellt.

```
1. <EntityType Name="Order" sem:Type="?order=northw:Order .
2.  _:orderAddress=northw:Address"
3.  sem:Mapping="?order northw:order_addr _:orderAddress">
4.  …
5.  <Property Name="OrderID" Type="Edm.Int32"
6.   sem:Mapping="?order northw:order_id"/>
7.  <Property Name="CustomerID" Type="Edm.String"
8.   sem:Mapping="?order northw:customer_id" />
9.  <Property Name="ShipAddress" Type="Edm.String"
10.  sem:Mapping="_:orderAddress northw:street"/>
11. <Property Name="ShipCity" Type="Edm.String"
12.  sem:Mapping="_:orderAddress northw:city"/>
13. …
14.</EntityType>
```

Listing 40: Die Beschreibung leerer Knoten.

Auf Basis dieser Beschreibung erzeugt das System für jede Order-Entität einen leeren Knoten. Wobei die Knoten-ID automatisch generiert wird und für eine spezielle Entität immer gleich ist. Im Gegensatz zu den Ressourcen-URIs enthält sie jedoch keine Informationen mit der die entsprechende Entität identifiziert oder abgerufen werden kann.

4.8 Abbildung zweier Entitätstypen auf ein Ressourcen-Template

Bisher wurde ausschließlich die Abbildung einzelner Entitätstypen auf RDF-Graph-Templates mit Ressourcen eines Typs betrachtet. Um die Abbildung zweier, sich gegenseitig über Navigationseigenschaften referenzierender Entitätstypen auf Ressourcen eines Typs zu ermöglichen, ist eine semantische Beschreibung der Navigationseigenschaften notwendig. Die dazu erforderlichen Konzepte sollen anhand des in Abbildung 15 dargestellten Beispiels erläutert werden. Auf der linken Seite von Abbildung 15 sind die beiden Entitätstypen *Order* (Bestellung) und *Shipper* (Spedition) dargestellt. Beide Entitätstypen stehen über eine Assoziation miteinander in Beziehung. Wobei jeweils eine Navigationseigenschaft von einem Entitätstypen zu dem anderen existiert. Eine Bestellung wird von keiner oder genau einer Spedition ausgeliefert. Eine Spedition kann für die Auslieferung beliebig vieler Bestellungen verantwortlich sein. Auf der rechten Seite ist das RDF-Graph-Template dargestellt, das mit den herzuleitenden Elementen beschrieben werden soll.

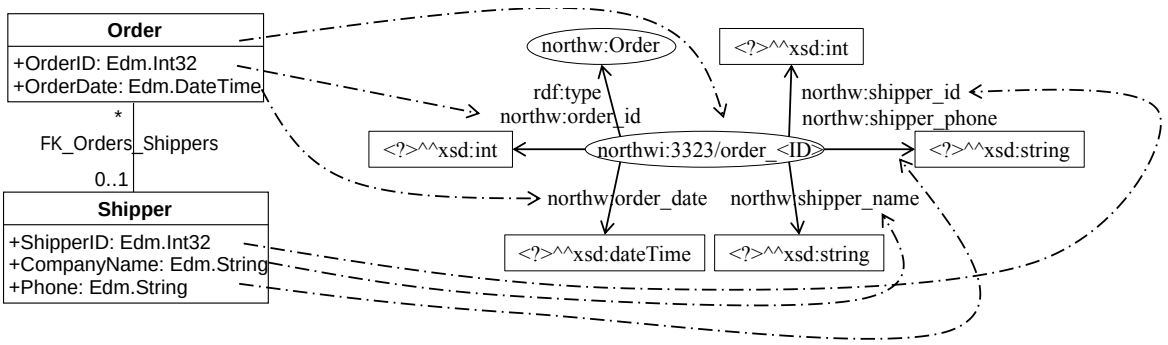


Abbildung 15: Abbildung zweier Entitätstypen auf ein Ressourcen-Template.

Die Abbildung der Eigenschaften des Entitätstyps *Order* kann mit den in Abschnitt 4.4 vorgestellten Attributen beschrieben werden. Zur Abbildung der Eigenschaften des Entitätstyps *Shipper* ist jedoch die Definition weiterer Konzepte bzw. die Erweiterung bereits existierender Konzepte notwendig. Diese müssen die Möglichkeit bieten, neben dem Entitätstyp, aus dessen Instanz die URI der Ressource erzeugt wird, auch andere Entitätstypen, welche mit diesem über eine Navigationseigenschaft in Beziehung stehen, in die Abbildung auf die entsprechende Ressource mit aufzunehmen. Es sind verschiedene Arten der Spezifikation vorstellbar. Listing 41 zeigt mit Hinblick auf Abbildung 15 eine Art der Angabe durch explizite Angabe der Variable ohne Kennzeichnung der Navigationseigenschaft.

```
1. <EntityType Name="Order" sem:Type="?order=northw:Order">
2.  ...
3. </EntityType>
4.
5. <EntityType Name="Shipper">
6.  <Property Name="ShipperID" Type="Edm.Int32"
7.   sem:Mapping="?order northw:shipper_id" />
8.  <Property Name="CompanyName" Type="Edm.String"
9.   sem:Mapping="?order northw:shipper_name" />
10. <Property Name="Phone" Type="Edm.String"
11.  sem:Mapping="?order northw:shipper_phone" />
12.</EntityType>
```

Listing 41: Abbildung mehrerer Entitätstypen durch explizite Angabe der Variable.

Für die Ressourcen des Typs *northw:Order*, die aus dem *Order*-Entitätstyp erstellt werden, wird eine Variable *?order* definiert. Diese Variable kann in den *sem:Mapping*-Attributen der Eigenschaften des Entitätstyps *Shipper* referenziert werden. Der Bezug auf die Navigationseigenschaft erfolgt somit implizit. Wenn eine Variable in der semantischen Beschreibung eines Entitätstyps verwendet wird und nicht in dieser definiert wurde, wird implizit die Beziehung über die Navigationseigenschaft zu dem Entitätstyp, in der die Variable definiert wurde, hergestellt. Die Beschreibung in Listing 41 gibt somit an, dass für eine bestimmte Entität des Entitätstyps *Order* jede Entität des Typs *Shipper*, die mit dieser

Entität in Beziehung steht, Aussagen den Angaben der semantischen Erweiterungen der Eigenschaften entsprechend, mit der *Order*-Ressource als Subjekt erzeugt werden. Allerdings ist eine Beschreibung durch die explizite Angabe der Variablen, wie in Listing 41, nur dann ausreichend, wenn es genau eine Navigationseigenschaft gibt, die beide Entitätstypen verbindet. Die CSDL-Spezifikation [138] schließt nicht explizit aus, dass es in einer CSDL-Service-Beschreibung mehrere Navigationseigenschaften geben kann, welche die gleichen Entitätstypen miteinander verbinden. In diesem speziellen Fall muss die Navigationseigenschaft, die in das Mapping mit einbezogen werden soll, explizit spezifiziert werden. Dies kann ausgehend von dem Entitätstyp erfolgen, welcher auf die Subjekt-Ressource abgebildet wird. Listing 42 zeigt ein Beispiel.

```
1. <EntityType Name="Order" sem:Type="?order=northw:Order">
2.  ...
3.  <NavigationProperty Name="Shipper"
4.   Relationship="NorthwindModel.FK_Orders_Shippers"
5.   FromRole="Orders" ToRole="Shippers" sem:Mapping="?order"
6.   />
7. </EntityType>
8.
9. <EntityType Name="Shipper">
10. <Property Name="ShipperID" Type="Edm.Int32"
11.  sem:Mapping="northw:shipper_ID" />
12.  …
13.</EntityType>
```

Listing 42: Abbildung mehrerer Entitätstypen durch explizite Kennzeichnung der Navigationseigenschaft.

Durch die semantische Erweiterung des *NavigationProperty*-Elements des *Order*-Entitätstyps mit dem *sem:Mapping*-Attribut wird über die Angabe der Ressourcen-Variable die Navigationseigenschaft, welche in die Abbildung mit einbezogen werden soll, eindeutig spezifiziert. Die Eigenschaften des Entitätstyps *Shipper* werden, wie bereits erläutert, semantisch erweitert. Wobei in diesem speziellen Fall im Gegensatz zu dem Beispiel aus Listing 41 die Variable an der Subjekt-Position nicht explizit angegeben werden muss. Da der *Shipper*-Entitätstyp selbst nicht auf Ressourcen eines Typs abgebildet wird, kann die Variable über die entsprechende Navigationseigenschaft hergeleitet werden. Wenn jedoch mehrere Navigationseigenschaften, die auf den *Shipper*-Entitätstyp verweisen, in die Abbildung mit einbezogen werden sollen oder der *Shipper*-Entitätstyp selbst auf ein Ressourcen-Template abgebildet wird, dann ist die explizite Angabe der Variable, wie in Listing 41 gezeigt, erforderlich.

Eine weitere Möglichkeit, die Beschreibung der Abbildung einer Navigationseigenschaft zu definieren, besteht darin, die relevante Navigationseigenschaft in den semantischen Erweiterungen der Eigenschaften des Entitätstyps anzugeben, in der die Ressourcen-Variable nicht definiert wurde. In diesem Fall müsste ein Referenzierungssmechanismus definiert werden, welcher die Angabe der entsprechenden

Navigationseigenschaft erlaubt. Da eine derartige Beschreibung der Abbildung weniger intuitiv ist als der bereits vorgestellte, wird sie nicht weiter betrachtet.

4.9 Abbildung einzelner Navigationseigenschaften auf abstrakte Aussagen

Im vorherigen Abschnitt wurde die Abbildung zweier, über eine Navigationseigenschaft miteinander in Beziehung stehender Entitätstypen auf ein einzelnes Ressource-Template betrachtet. In diesem Abschnitt soll die semantische Beschreibung der Navigationseigenschaften dahingehend erweitert werden, dass die Abbildung einer Navigationseigenschaft auf eine abstrakte Aussage ermöglicht wird. Die dazu erforderlichen Konzepte sollen anhand des in Abbildung 16 dargestellten Beispiels erläutert werden.

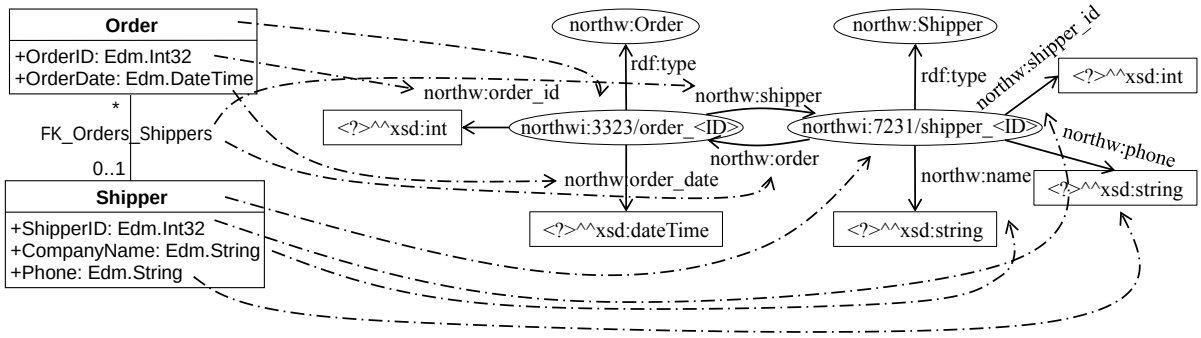


Abbildung 16: Abbildung einer Navigationseigenschaft auf zwei abstrakte Aussagen.

Auf der linken Seite von Abbildung 16 sind die bereits aus Abbildung 15 bekannten Entitätstypen *Order* (Bestellung) und *Shipper* (Spedition) dargestellt. Das RDF-Graph-Template auf der rechten Seite zeigt zwei Ressourcen-Templates. Der Entitätstyp *Order* wird auf ein Ressource-Template des Typs *northw:Order* abgebildet. Das Ressource-Template des Typs *northw:Shipper* wird aus dem *Shipper*-Entitätstyp generiert. Die Navigationseigenschaften, welche auf der Assoziation *FK_Order_Shippers* basieren und beide Entitätstypen miteinander verbinden, werden auf Aussagen-Templates abgebildet, welche beide Ressourcen-Templates miteinander in Beziehung setzen. Somit ist zu spezifizieren, dass Entitäten der Typen *Order* und *Shipper*, die über Navigationseigenschaften der Assoziation *FK_Order_Shippers* verbunden sind, auf zwei Aussagen mit den Prädikaten *northw:shipper* und *northw:order* abgebildet werden. Wobei Subjekte und Objekte dieser beiden Aussagen jeweils den URIs der aus den entsprechenden Entitäten erzeugten Ressourcen entsprechen müssen. Eine mögliche Art der Beschreibung besteht in der Angabe der Aussagen über das *sem:Mapping*-Attribut der entsprechenden *EntityType*-Elemente. Listing 43 zeigt dies mit Hinblick auf das Beispiel aus Abbildung 16.

```
1. <EntityType Name="Order" sem:Type="?order=northw:Order"
2.  sem:Mapping="?order northw:shipper ?shipper">
3.  ...
4. </EntityType>
5.
6. <EntityType Name="Shipper"
7.  sem:Type="?shipper=northw:Shipper"
8.  sem:Mapping="?shipper northw:order ?order">
9.  ...
10.</EntityType>
```

Listing 43: Abbildung einer Navigationseigenschaft auf abstrakte Aussagen über die Angabe der Variablen.

Die für die Abbildung relevante Navigationseigenschaft ist dabei implizit gegeben (siehe dazu die Erläuterungen in Abschnitt 4.8). Wenn die Navigationseigenschaft explizit gekennzeichnet werden soll oder wenn mehrere Navigationseigenschaften existieren, welche die gleichen Entitätstypen miteinander verbinden, ist eine semantische Erweiterung des *NavigationProperty*-Elements mit dem *sem:Mapping*-Attribut erforderlich. Listing 44 zeigt den relevanten Teil der semantisch erweiterten Entitätstypen. Im Gegensatz zu der in Abschnitt 4.8 beschriebenen semantischen Erweiterung der Navigationseigenschaft, erhält bei der Abbildung der selbigen auf abstrakte Aussagen das *sem:Mapping*-Attribut als Wert nicht die Ressourcen-Variable, sondern das Prädikat der abstrakten Aussage. Mit diesen Angaben ist die dargestellte Abbildung vollständig beschrieben.

```
1. <EntityType Name="Order" sem:Type="northw:Order">
2.  …
3.  <NavigationProperty Name="Shipper"
4.   Relationship="NorthwindModel.FK_Orders_Shippers"
5.   FromRole="Orders" ToRole="Shippers"
6.   sem:Mapping="northw:shipper"/>
7. </EntityType>
8.
9. <EntityType Name="Shipper" sem:Type="northw:Shipper">
10. …
11. <NavigationProperty Name="Orders"
12.  Relationship="NorthwindModel.FK_Orders_Shippers"
13.  FromRole="Shippers" ToRole="Orders"
14.  sem:Mapping="northw:order"/>
15.</EntityType>
```

Listing 44: Abbildung einer Navigationseigenschaft auf abstrakte Aussagen durch explizite Kennzeichnung.

Wenn einer der Entitätstypen auf mehrere Ressourcen-Templates abgebildet werden soll, ist die dargestellte Beschreibung nicht ausreichend. Abbildung 17 zeigt ein Beispiel. Im Gegensatz zu dem Beispiel aus Abbildung 16 werden in Abbildung 17 Entitäten des

Entitätstyps *Orders* auf zwei Ressourcen-Templates, *northwi:3323/order_<ID>* und *northwi:3323/address_<ID>*, abgebildet.

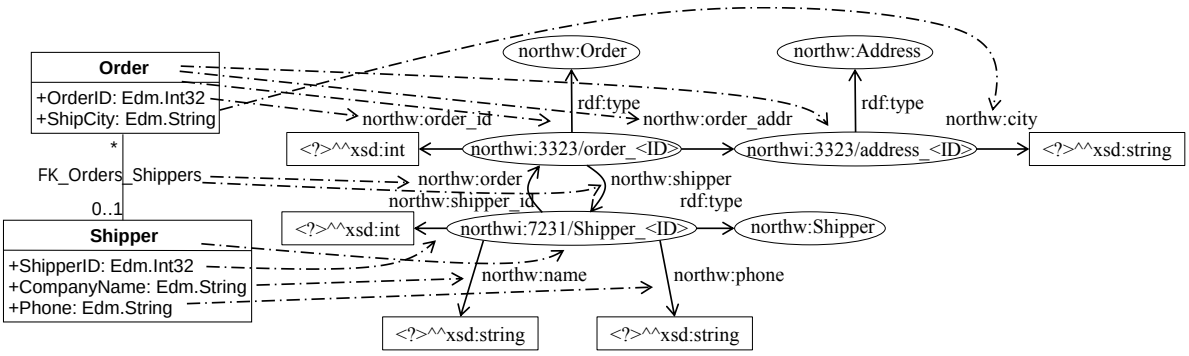


Abbildung 17: Abbildung einer Navigationseigenschaft mit mehreren Ressourcen-Templates.

Die in Listing 44 dargestellte Beschreibung ist in diesem Fall nicht ausreichend, da nicht definiert ist, ob die Ressource *northwi:7231/Shipper_<ID>* mit der Ressource *northwi:3323/order_<ID>* oder der Ressource *northwi:3323/address_<ID>* in Beziehung steht. Das *sem:Mapping*-Attribut des *NavigationProperty*-Elements des *Shipper*-Entitätstyps muss daher um die Angabe der Ziel-Ressource erweitert werden. Listing 45 zeigt die Erweiterung. In Zeile 1 und 2 ist die Ressourcen-spezifische Typ-Angabe dargestellt. Zeile 8 zeigt die Angabe der Ziel-Ressource für die Navigationseigenschaft des *Shipper*-Entitätstyps. Der erweiterten Turtle-Syntax entsprechend, folgt auf das Prädikat (*northw:order*) die Variable der Ressource, welche das Objekt der Aussage bilden soll. Damit ist der Graph aus Abbildung 17 vollständig beschrieben.

```
1. <EntityType Name="Order" sem:Type="?order=northw:Order .
2.  ?address=northw:Address">
3.  …
4. </EntityType>
5.
6. <EntityType Name="Shipper" sem:Type="northw:Shipper">
7.  <NavigationProperty Name="Orders"
8.   sem:Mapping="northw:order ?order"/>
9.  …
10.</EntityType>
```

Listing 45: Abbildung einer Navigationseigenschaft mit Angabe der Ziel-Ressource.

Wenn ein Entitätstyp auf mehrere Ressourcen-Templates abgebildet wird, dann kann bei der Abbildung einer Navigationseigenschaft die Ressource, welche das Subjekt der Aussage bildet, durch die Angabe der entsprechenden Variablen in dem *sem:Mapping*-Attribut an der Subjekt-Position spezifiziert werden. Listing 46 zeigt als Beispiel, mit Hinblick auf Abbildung 17, die semantische Erweiterung der Navigationseigenschaft des Entitätstyps *Order*. Über die Angabe der *?order*-Variablen an der Subjekt-Position ist eindeutig definiert, welche Ressource das Subjekt der Aussage bildet. Aufgrund der zusätzlichen Angabe des Prädikats,

welche für die Abbildung auf abstrakte Aussagen immer erforderlich ist, ist auch weiterhin eine Unterscheidung zu der in Abschnitt 4.8 erläuterten semantischen Erweiterung zur Abbildung auf konkrete Aussagen gewährleistet.

```
1. <EntityType Name="Order" sem:Type="?order=northw:Order .
2.  ?address=northw:Address">
3.  <NavigationProperty Name="Shipper"
4.   Relationship="NorthwindModel.FK_Orders_Shippers"
5.   FromRole="Orders" ToRole="Shippers"
6.   sem:Mapping="?order northw:shipper"/>
7. </EntityType>
```

Listing 46: Abbildung einer Navigationseigenschaft mit Angabe der Subjekt-Ressourcen-Variable.

Die in Listing 45 und Listing 46 vorgestellten Angaben müssen kombiniert werden, wenn jede der beiden Entitätstypen, die sich gegenseitig über Navigationseigenschaften referenzieren, auf mehrere Ressourcen-Templates abgebildet werden. In diesem Fall erhält das *sem:Mapping*-Attribut als Wert ein vollständiges Tripel-Template, mit den entsprechenden Ressource-Variablen an Subjekt- und Objekt-Position. Angelehnt an das obige Beispiel könnte sich z.B. folgendes Tripel-Template ergeben: *?order northw:shipper ?shipper*.

4.10 Abbildung mehrerer Entitätstypen auf ein Ressourcen-Template

In Abschnitt 4.8 wurde gezeigt, wie zwei Entitätstypen, die über eine Assoziation miteinander in Beziehung stehen, auf ein RDF-Graph-Template mit einem einzelnen Ressourcen-Template abgebildet werden können. In diesem Abschnitt sollen die vorgestellten Konzepte dahingehend erweitert werden, dass die Abbildung von mehr als zwei, sich gegenseitig referenzierender Entitätstypen möglich ist. Abbildung 18 zeigt als Beispiel eine Abbildung mehrerer Entitätstypen auf ein Ressourcen-Template.

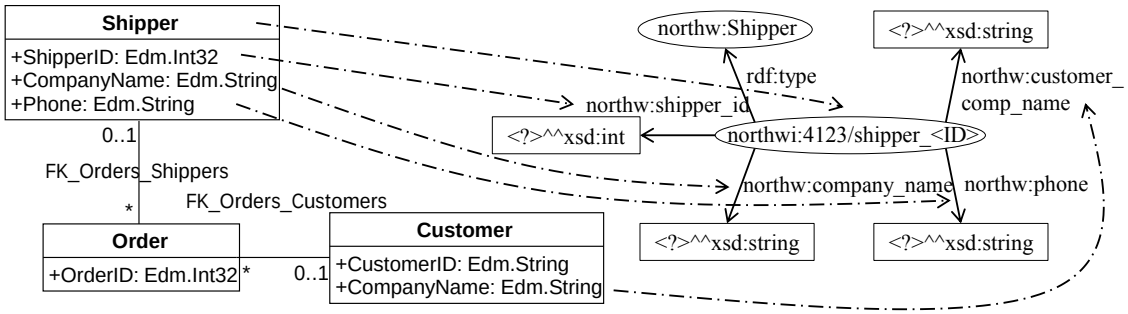


Abbildung 18: Abbildung mehrerer Entitätstypen auf ein Ressourcen-Template.

Das auf der rechten Seite dargestellte Ressourcen-Template basiert auf dem Entitätstyp *Shipper*. Die Eigenschaften *ShipperID*, *CompanyName* und *Phone* werden auf Aussagen mit den Prädikaten *shipper_id*, *company_name* und *phone* abgebildet. Des Weiteren wird für jede

Entität des Typs *Customer*, die mit einer bestimmten Entität des Typs *Shipper* über eine Entität des Typs *Order* verbunden ist, eine Aussage mit dem Prädikat *company_name* erzeugt. Dabei entsprechen die Objekte dieser Aussagen jeweils den Werten der *CompanyName*-Eigenschaften der *Customer*-Entitäten. Um eine solche Abbildung, die sich über mehr als eine Navigationseigenschaft erstreckt, beschreiben zu können, muss die semantische Erweiterung die Spezifikation von Abbildungen über mehrere zusammenhängende Navigationseigenschaften ermöglichen. Wie bei der in Abschnitt 4.8 erläuterten Erweiterung zur Beschreibung der Abbildung einzelner Navigationseigenschaften gibt es mehrere Arten der Beschreibung. Listing 47 zeigt die Beschreibung über die Angabe der Variable, ohne Spezifikation der Navigationseigenschaft.

```
1. <EntityType Name="Shipper"
2.  sem:Type="?shipper=northw:Shipper">
3.  …
4. </EntityType>
5.
6. <EntityType Name="Customer">
7.  <Property Name="CompanyName" Type="Edm.String"
8.   sem:Mapping="?shipper northw:customer_comp_name" />
9. </EntityType>
```

Listing 47: Abbildung über mehrerer Navigationseigenschaften durch explizite Angabe der Variable.

Die Navigationseigenschaften zwischen den Entitätstypen *Shipper* und *Customer*, die bei der Abbildung berücksichtigt werden müssen, sind damit implizit angegeben. Für jede *Customer*-Entität, für die eine *Order*-Entität existiert, so dass die *Shipper*-Entität, welche die Basis für die Abbildung auf die Ressource bildet, mit der *Customer*-Entität über die *Order*-Entität verbunden ist, wird eine Aussage mit dem Prädikat *northw:customer_comp_name* gebildet. Wenn die *Shipper*-Entität mit drei *Order*-Entitäten verbunden ist, welche wiederum jeweils eine *Customer*-Entität referenzieren, werden drei Aussagen erzeugt. Für den Fall, dass mehrere mögliche Beziehungen zwischen zwei Entitätstypen existieren, muss die Semantik dieser Art der Beschreibung dahingehend erweitert werden, dass eindeutig definiert ist, welche Navigationseigenschaften abgebildet werden. In Abbildung 19 ist dieser Fall beispielhaft dargestellt.

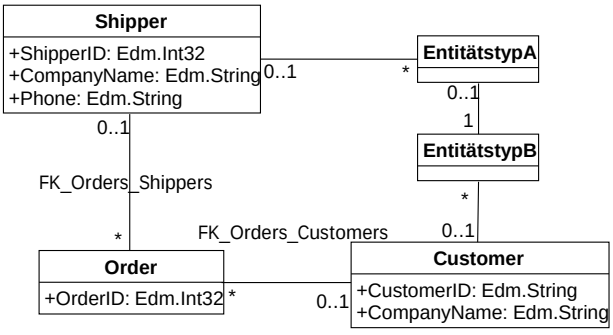


Abbildung 19: Mehrere Verbindungen zwischen zwei Entitätstypen.

Neben der Verbindung über den Entitätstyp *Order* ist der Typ *Shipper* auch über die beiden fiktiven Entitätstypen *EntitätstypA* und *EntitätstypB* mit dem *Customer*-Entitätstyp verbunden. Mit der semantischen Beschreibung aus Listing 47 ist nicht definiert, welche Navigationseigenschaften bei der Abbildung berücksichtigt werden sollen. Um die Semantik der Beschreibung eindeutig zu definieren, wird die Abbildung auf die kürzeste Verbindung festgelegt. Wobei die Länge der Verbindung zwischen zwei Entitätstypen über die Anzahl der Navigationseigenschaften definiert ist, aus der sich die Verbindung zusammensetzt. Dementsprechend hat die Verbindung über den Entitätstyp *Order* aus Abbildung 19 die Länge zwei. Die Länge der Verbindung über *EntitätstypA* und *EntitätstypB* beträgt drei. Somit erfolgt die Abbildung über die Verbindung mit dem Entitätstyp *Order*.

Wenn mehrere Verbindungen mit gleicher Länge existieren, ist die Beschreibung auch mit dieser Festlegung nicht eindeutig. In diesem Fall muss die Verbindung durch die semantische Erweiterung der Navigationseigenschaften angegeben werden. Die Beschreibung erfolgt, wie bereits in Abschnitt 4.8 erläutert, durch die Erweiterung des *NavigationProperty*-Elements mit dem *sem:Mapping*-Attribut. Dieses erhält als Wert die Variable der Ressource, die das Subjekt der Aussage bilden soll. Dabei wird, wie in Listing 48 gezeigt, die vollständige Verbindung durch die semantische Beschreibung aller Navigationseigenschaften, die Teil dieser Verbindung sind, beschrieben.

```
1. <EntityType Name="Shipper"
2.  sem:Type="?shipper=northw:Shipper">
3.  <NavigationProperty Name="Orders" FromRole="Shippers"
4.   ToRole="Orders" sem:Mapping="?shipper"/>
5. </EntityType>
6.
7. <EntityType Name="Order">
8.  <NavigationProperty Name="Customer" FromRole="Orders"
9.   ToRole="Customers" sem:Mapping="?shipper" />
10.</EntityType>
11.
12.<EntityType Name="Customer">
13. <Property Name="CompanyName" Type="Edm.String"
14.  sem:Mapping="?shipper northw:customer_comp_name" />
15.</EntityType>
```

Listing 48: Abbildung über mehrerer Navigationseigenschaft durch explizite Kennzeichnung.

4.11 Abbildung mehrerer Navigationseigenschaften auf abstrakte Aussagen

In diesem Abschnitt sollen die in Abschnitt 4.9 vorgestellten Konzepte zur Abbildung von Navigationseigenschaften auf abstrakte Aussagen dahingehend erweitert werden, dass die Abbildung über mehrere Navigationseigenschaften ermöglicht wird. Zur Herleitung dient das in Abbildung 20 dargestellte Beispiel.

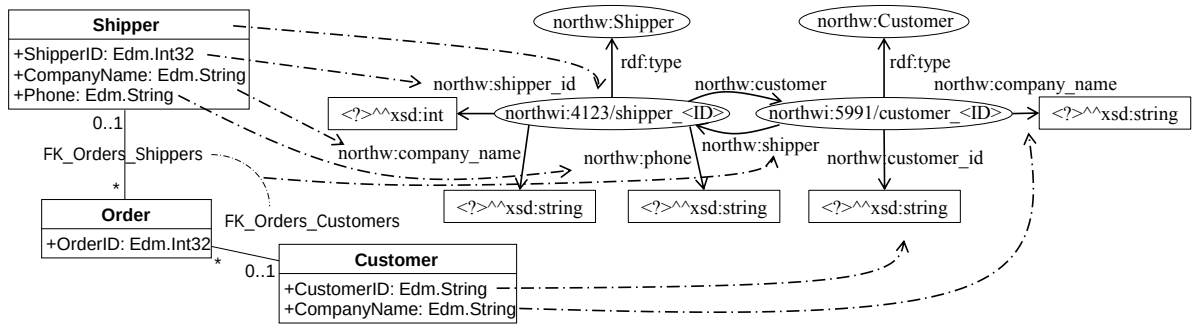


Abbildung 20: Abbildung mehrerer Navigationseigenschaften auf abstrakte Aussagen.

Entitäten der Entitätstypen *Shipper* und *Customer* werden auf Ressourcen der Typen *northw:Shipper* und *northw:Customer* abgebildet. Für jede Verbindung zwischen einer *Shipper*-Entität und einer *Customer*-Entität in Form zweier, über eine *Order*-Entität miteinander in Beziehung stehender Navigationseigenschaften, werden zwei abstrakte Aussagen mit den Prädikaten *northw:customer* und *northw:shipper* erzeugt. Es existieren wieder mehrere Arten der Beschreibung. Eine Möglichkeit besteht darin, die Abbildung auf die abstrakte Aussage, wie in Listing 49 gezeigt, im *sem:Mapping*-Attribut des entsprechenden *EntityType*-Elements zu spezifizieren. Wobei die entsprechenden Navigationseigenschaften nicht explizit gekennzeichnet werden. Stattdessen werden sie automatisch ermittelt.

```
1. <EntityType Name="Shipper"
2.  sem:Type="?shipper=northw:Shipper"
3.  sem:Mapping="?shipper northw:customer ?customer">
4.  …
5. </EntityType>
6.
7. <EntityType Name="Customer"
8.  sem:Type="?customer=northw:Customer"
9.  sem:Mapping="?customer northw:shipper ?shipper">
10. …
11.</EntityType>
```

Listing 49: Abbildung mehrerer Navigationseigenschaften auf abstrakte Aussagen ohne Kennzeichnung der Navigationseigenschaft.

Wie in Abschnitt 4.10 erläutert, kann diese Art der Beschreibung auch dann verwendet werden, wenn mehrere Verbindungen zwischen zwei Entitätstypen existieren. Dabei sind die relevanten Navigationseigenschaften durch die kürzeste Verbindung zwischen den Entitätstypen definiert. Wenn mehrere gleich lange Verbindungen existieren oder die Navigationseigenschaften explizit gekennzeichnet werden sollen, dann besteht die Möglichkeit, die *NavigationProperty*-Elemente mit dem *sem:Mapping*-Attribut zu erweitern. Listing 50 zeigt diese Art der Beschreibung.

```
1. <EntityType Name="Shipper"
2.  sem:Type="?shipper=northw:Shipper">
3.  <NavigationProperty Name="Orders" FromRole="Shippers"
4.   ToRole="Orders" sem:Mapping="?shipper northw:customer"/>
5. </EntityType>
6.
7. <EntityType Name="Order">
8.  <NavigationProperty Name="Customer" FromRole="Orders"
9.   ToRole="Customers" sem:Mapping="?shipper northw:customer"
10. />
11.</EntityType>
12.
13.<EntityType Name="Customer" sem:Type="northw:Customer">
14. <Property Name="CustomerID" Type="Edm.String"
15.  sem:Mapping="customer_id" />
16.</EntityType>
```

Listing 50: Abbildung mehrere Navigationseigenschaften auf eine abstrakte Aussage mittels expliziter Kennzeichnung.

Wie bereits bei der Erläuterung über die Abbildung einer Navigationseigenschaft auf eine abstrakte Aussage in Abschnitt 4.9 beschrieben, erhält das *sem:Mapping*-Attribut als Wert das Prädikat der abstrakten Aussage. Um eine Verwechselung mit der Beschreibung über die Abbildung einer einzelnen Navigationseigenschaft zu vermeiden und um eine effizientere Verarbeitung zu ermöglichen, ist die Angabe der Subjekt-Variablen obligatorisch. Damit die entsprechende Verbindung eindeutig identifiziert werden kann, müssen alle *NavigationProperty*-Elemente einer Verbindung mit dem *sem:Mapping*-Attribut erweitert werden. Wobei diese als Werte alle dieselbe Subjekt- und Prädikat-Angabe erhalten. Die Angabe des Objekts in Form einer Ressourcen-Variablen ist optional, sofern der entsprechende Entitätstyp auf genau ein Ressourcen-Template abgebildet wird. Anderenfalls muss bei der semantischen Beschreibung der Navigationseigenschaften der Verbindung immer die Ressourcen-Variablen an Objekt-Position angegeben werden.

4.12 Mehrfachabbildungen von Eigenschaften

In den vorherigen Abschnitten wurde gezeigt, wie eine aus mehreren Navigationseigenschaften bestehende Verbindung zwischen zwei Entitätstypen auf eine abstrakte Aussage abgebildet wird, welche die aus den Entitätstypen gebildeten Ressourcen-Templates miteinander in Beziehung setzt. Des Weiteren wurde erläutert, wie eine Abbildung von Eigenschaften auf konkrete Aussagen über mehrere Navigationseigenschaften beschrieben werden kann. In diesem Abschnitt wird, anhand des Beispiels aus Abbildung 21, gezeigt, wie beide Abbildungsarten für einen einzelnen Entitätstyp kombiniert werden können.

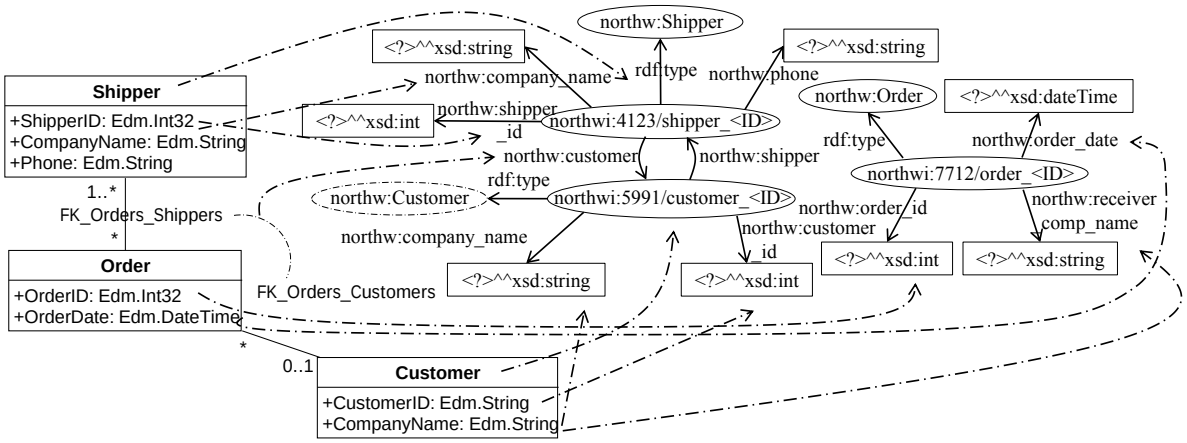


Abbildung 21: Unterschiedliche Abbildungen in Abhängigkeit der Navigationseigenschaften.

Auf der linken Seite in Abbildung 21 sind die drei, bereits aus den vorherigen Abschnitten bekannten Entitätstypen *Shipper*, *Order* und *Customer* dargestellt. Diese werden auf drei Ressourcen-Templates der Typen *northw:Shipper*, *northw:Customer* und *northw:Order* abgebildet. Dabei erfolgt für den Entitätstyp *Customer* eine zweifache Abbildung. Aus jeder *Customer*-Entität wird eine Ressource des Typs *northw:Customer* erzeugt, wobei die Eigenschaften *CustomerID* und *CompanyName* auf konkrete Aussagen mit den Prädikaten *northw:customer_id* und *northw:company_name* abgebildet werden. Des Weiteren werden für jede Entität des Typs *Shipper*, welche mit der *Customer*-Entität über eine *Order*-Entität in Beziehung steht, zwei abstrakte Aussagen mit den Prädikaten *northw:customer* und *northw:shipper* erstellt, welche die entsprechenden Ressourcen miteinander verbinden.

Als zweite Abbildung wird für jede *Order*-Entität, die mit der *Customer*-Entität über die Navigationseigenschaft der *FK_Order_Customers*-Assoziation verbunden ist, die Eigenschaft *CompanyName* der *Customer*-Entität in eine konkrete Aussage mit dem Prädikat *northw:receiver_comp_name* überführt. Wobei das Subjekt durch die URI der entsprechenden *northw:Order*-Ressource gebildet wird. Somit wird die Eigenschaft *CompanyName* der *Customer*-Entität auf zwei Aussagen abgebildet. Um diese Abbildung beschreiben zu können, muss das *sem:Mapping*-Attribut dahingehend erweitert werden, dass es die Angabe zweier Aussagen-Templates ermöglicht. Listing 51 zeigt den relevanten Ausschnitt des semantisch erweiterten CSDL-Dokuments.

```
1. <EntityType Name="Shipper"
2.  sem:Type="?shipper=northw:Shipper">
3.  <NavigationProperty Name="Orders"
4.   Relationship="NorthwindModel.FK_Orders_Shippers"
5.   sem:Mapping="?shipper northw:customer ?customer" />
6. </EntityType>
7.
8. <EntityType Name="Order" sem:Type="?order=northw:Order">
9.  <NavigationProperty Name="Customer"
10.  Relationship="NorthwindModel.FK_Orders_Customers"
11.  sem:Mapping="?shipper northw:customer ?customer .
12.               ?order"/>
13.</EntityType>
14.
15.<EntityType Name="Customer"
16. sem:Type="?customer=northw:Customer">
17. <Property Name="CompanyName" Type="Edm.String"
18.  sem:Mapping="?order northw:receiver_comp_name .
19.               ?customer northw:company_name" />
20.</EntityType>
```

Listing 51: Abbildung einer Eigenschaft auf mehrere Aussagen.

Die semantische Beschreibung der *CompanyName*-Eigenschaft ist in den Zeilen 18 und 19 dargestellt. Dabei erfolgt die Beschreibung der Abbildung auf eine konkrete Aussage der *northw:Order*-Ressource mittels des Aussagen-Templates *?order northw:receiver_comp_name*. Die Abbildung auf die Aussagen der *northw:Customer*-Ressourcen wird über das *?customer northw:company_name* Template beschrieben. Beide Aussagen-Templates werden, der Turtle-Notation entsprechend, durch einen Punkt getrennt. Für die mehrfache semantische Beschreibung der Navigationseigenschaften wird, wie in den Zeilen 11 und 12 dargestellt, die gleiche Formatierung verwendet. Wobei diese, wie in den vorherigen Abschnitten erläutert, definiert ist.

4.13 Beschreibung von komplexen Typen

Bisher wurde ausschließlich die semantische Beschreibung von Entitätstypen erläutert. Neben Entitätstypen können komplexere Datenstrukturen in CSDL mittels sogenannter komplexer Typen ausgedrückt werden. Komplexe Typen setzen sich aus einer Menge von Eigenschaften zusammen. Sie weisen aber laut [138] keine Schlüsselelemente auf. Daher können Instanzen komplexer Typen immer nur in Abhängigkeit von bestimmten Entitäten existieren. Des Weiteren können sie keine Navigationseigenschaften besitzen. Da es sich somit um eine eingeschränkte Form der Entitätstypen handelt, können im Wesentlichen die bereits vorgestellten Konzepte zu deren Beschreibung übernommen werden. Aufgrund der fehlenden

Schlüsselelemente muss aber die Erstellung der URIs angepasst werden. Dies soll anhand des in Abbildung 22 dargestellten Beispiels erläutert werden.

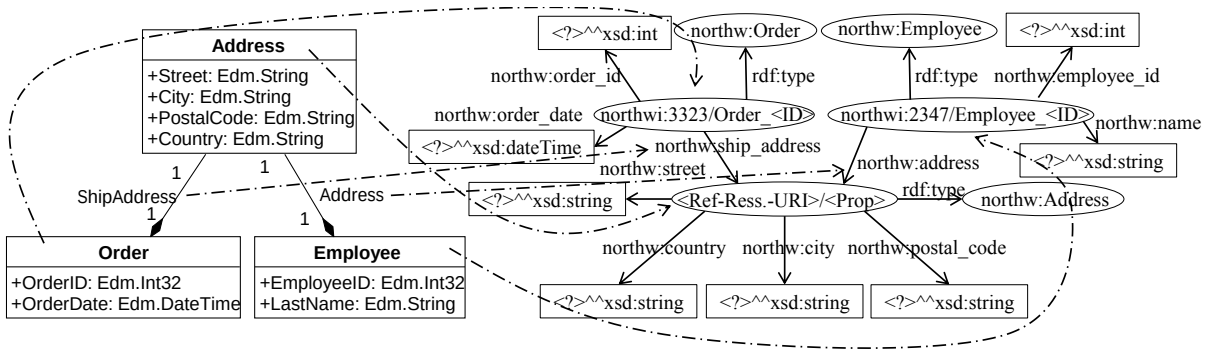


Abbildung 22: Abbildung eines komplexen Typs auf ein Ressourcen-Template.

Auf der linken Seite von Abbildung 22 sind die beiden Entitätstypen *Order* und *Employee* dargestellt. Im Datenmodell des Northwind-Services haben beide Entitätstypen Eigenschaften zur Repräsentation der Adresse, wie z.B. *ShipCountry*, *ShipCity* bzw. *Country*, *City* usw. In Abbildung 22 wurden diese Eigenschaften in einen komplexen Typ *Address* ausgelagert, der von beiden Entitätstypen referenziert wird. Die Referenzierungen, im UML-Klassendiagramm als Kompositionen visualisiert, werden im CSDL-Dokument durch Eigenschaften ausgedrückt, welche als Typ den komplexen Typ *Address* aufweisen. Auf der rechten Seite von Abbildung 22 ist das RDF-Graph-Template dargestellt, auf welches die Entitätstypen und der komplexe Typ abgebildet werden sollen. Aus jeder *Order*-Entität wird eine Ressource des Typs *northw:Order* erstellt und aus jeder Entität des Entitätstyps *Employee* eine Ressource des Typs *northw:Employee*. Jede Eigenschaft des komplexen Typs *Address* wird auf eine Ressource des Typs *northw:Address* abgebildet. Listing 52 zeigt die semantische Beschreibung des komplexen Typs und die Abbildung der entsprechenden Eigenschaft des *Order*-Entitätstyps.

```
1. <EntityType Name="Order" sem:Type="northw:Order">
2.  <Property Name="ShipAddress" Type="NorthwindModel.Address"
3.   sem:Mapping="northw:ship_address" />
4. </EntityType>
5.
6. <ComplexType Name="Address" sem:Type="northw:Address">
7.  <Property Name="Street" Type="Edm.String" Nullable="true"
8.   sem:Mapping="northw:street" />
9.  ...
10.</ComplexType>
```

Listing 52: Beschreibung eines komplexen Typs.

Die semantische Beschreibung erfolgt mit den bereits bekannten Konzepten. Wobei die semantische Beschreibung der Eigenschaft, welche den komplexen Typ referenziert (Zeile 3), wie die semantische Erweiterung einer Navigationseigenschaft interpretiert wird.

Wie bereits erwähnt, haben komplexe Typen keine Schlüsselelemente. Außerdem existieren sie nur in Abhängigkeit von anderen Entitäten. Daher können URIs für Ressourcen auf Basis von komplexen Typen nicht auf dieselbe Art erstellt werden wie Ressourcen, die aus Entitäten erzeugt werden.

Wie auch bei der Definition der URIs von Ressourcen, die aus Entitäten generiert werden (siehe Abschnitt 4.3), müssen URIs von Ressourcen auf Basis von komplexen Typen alle Angaben enthalten, die notwendig sind, um die Daten, die zur Erzeugung der Aussagen über die Ressource benötigt werden, abrufen zu können. Da eine Instanz eines komplexen Typs immer nur in Abhängigkeit einer bestimmten Entität existiert, muss die URI der Ressource die URI der Entität, mit der die Instanz des komplexen Typs in Beziehung steht, enthalten. Diese ist aber bereits in der URI der Ressource enthalten, auf welche die entsprechende Entität abgebildet wird. Daher kann diese URI als Basis verwendet werden. Um, ausgehend von der Entität, den komplexen Typ eindeutig identifizieren zu können, ist der Name der Eigenschaft des Entitätstyps, über die der komplexe Typ referenziert wird, ausreichend. Dieser wird, getrennt durch einen Schrägstrich, an die URI der Entität angehängt.

Komplexe Typen können beliebig tief geschachtelt sein. Sei ein komplexer Typ gegeben, der von einem anderen komplexen Typ über eine Eigenschaft referenziert wird. In diesem Fall ist es erforderlich, dass sowohl der Name der Eigenschaft des Entitätstyps als auch der Name der Eigenschaft des komplexen Typs in der Ressourcen-URI enthalten sind. Somit ist die Anzahl der Namen der Eigenschaften in der Ressourcen-URI abhängig von der Verschachtelungstiefe. Auf Basis der in Abschnitt 4.3 angegebenen Definition ergibt sich folgendes URI-Template (erweiterte Backus-Naur-Form):

Ressource-URI = "http://", *Host*, [*Path-Prefix*], "/", *Entity-Location-ID*, "/", *Path-Suffix*,
("/", *Property-Name*)+ ;

Anders verhält es sich, wenn die Eigenschaft eine Kollektion des komplexen Typs referenziert. Es besteht dann nicht die Möglichkeit, eine bestimmte Instanz des komplexen Typs über den Namen der Eigenschaft eindeutig zu identifizieren. Daher werden in diesem Fall die Ressourcen durch leere Knoten repräsentiert.

Wie in Abschnitt 4.10 erläutert, kann es bei der Abbildung mehrerer Entitätstypen über Navigationseigenschaften vorkommen, dass zwischen zwei Entitätstypen mehr als eine Verbindung existiert. In diesem Fall wird implizit die kürzeste Verbindung zwischen den Entitätstypen in die Abbildung einbezogen. Wenn eine andere als die kürzeste Verbindung bei der Abbildung berücksichtigt werden soll, können die entsprechenden Navigationseigenschaften auf die gezeigte Art und Weise markiert werden. Zwischen einem Entitätstypen und einem komplexen Typen kann es ebenfalls mehrere Verbindungen geben, wenn der komplexe Typ, ausgehend von dem Entitätstyp, direkt oder indirekt über mehrere Eigenschaften referenziert wird.

Sei als Beispiel neben der in Listing 52 dargestellten *ShipAddress*-Eigenschaft des *Order*-Entitätstyps eine weitere Eigenschaft *BillingAddress* gegeben, die ebenfalls den komplexen Typ *Address* aufweist. In diesem Fall werden als Teil einer Entität der *Orders*-Entitätsmenge zwei Instanzen des komplexen Typs *Address* zurückgegeben. Aus einer Entität

werden daher zwei Ressourcen des Typs *northw:Address* erzeugt. Wobei sich die URIs der Ressourcen im Aufbau hinsichtlich der Eigenschaftsnamen unterscheiden. Allgemein formuliert, wird auf Basis einer Ressourcen-Variable, die für einen komplexen Typ spezifiziert wurde, für jede Instanz dieses Typs, welche Teil einer Entität ist, eine extra Ressource erzeugt.

Bei einer Referenz in Form einer abstrakten Aussage auf eine Ressourcen-Variable, welche auf Basis eines komplexen Typs generiert wird, muss die Ressource, die Objekt der sich ergebenden Aussage wird, eindeutig spezifiziert sein. Dies kann, wie in Listing 52 dargestellt, dadurch geschehen, dass die Eigenschaft oder die Eigenschaften durch das Tripel-Template selbst markiert werden. Anderenfalls wird, ähnlich wie bei der Abbildung von Navigationseigenschaften, nur die Instanz des komplexen Typs mit der kürzesten Verbindung zum Entitätstypen berücksichtigt. Wobei die Länge der Verbindung zwischen einem Entitätstypen und einem komplexen Typ durch die Anzahl der Eigenschaften definiert ist, über die der komplexe Typ, ausgehend von dem Entitätstypen, referenziert wird. Das gleiche gilt bzgl. der Verbindung zwischen zwei komplexen Typen, wenn eine abstrakte Aussage zwei Ressourcen-Templates miteinander in Beziehung setzt, die auf verschiedenen komplexen Typen basieren.

Mit Hinblick auf die Einbeziehung von Eigenschaften komplexer Typen in die Abbildung auf konkrete Aussagen von Ressourcen-Variablen, die auf Entitätstypen oder anderen komplexen Typen basieren, können die gleichen Konzepte wie in Abschnitt 4.10 verwendet werden. Wobei die Markierung der in die Abbildung einzubeziehenden Verbindung, wie bei der Markierung von Navigationseigenschaften, durch eine Annotation der Eigenschaft oder der Eigenschaften mit der entsprechenden Ressourcen-Variable erfolgen kann.

Wenn Instanzen des komplexen Typs sowohl auf Ressourcen als auch auf konkrete Aussagen einer Ressource, welche aus einer referenzierenden Entität erzeugt wurde, abgebildet werden sollen, können die Konzepte aus Abschnitt 4.12 verwendet werden.

4.14 Typhierarchien

CSDL ermöglicht die Beschreibung von Typhierarchien. Entitätstypen und komplexe Typen können von anderen Entitätstypen bzw. komplexen Typen abgeleitet werden [138]. Dabei erben sie den Schlüssel und die Eigenschaften des Basistyps. Wobei diese in den abgeleiteten Typen nicht überschrieben werden können. Die Definition neuer Eigenschaften ist aber möglich. Mehrfachvererbung ist nicht erlaubt. In diesem Abschnitt soll die Abbildung von Typhierarchien auf RDF-Graphen erläutert werden.

Es können mehrere Arten der Abbildung unterschieden werden. Die einfachste Art der Abbildung besteht darin, dass alle Entitätstypen einer Typhierarchie auf ein Ressourcen-Template abgebildet werden. Jede Eigenschaft, die ein abgeleiteter Entitätstyp seinem Basistyp hinzufügt, wird dabei auf ein weiteres Aussagen-Template abgebildet. Diese Art der Abbildung kann mit den bereits erläuterten Konzepten beschrieben werden. In der semantischen Erweiterung des obersten Typs der Typhierarchie wird der Typ der Ressource

angegeben und eine entsprechende Variable definiert. In den abgeleiteten Typen werden die Abbildungen unter Bezug auf diese Variable spezifiziert.

Eine komplexere Abbildung ergibt sich, wenn die Entitätstypen einer Typhierarchie auf verschiedene Ressourcen-Templates abgebildet werden sollen. Abbildung 23 zeigt ein Beispiel.

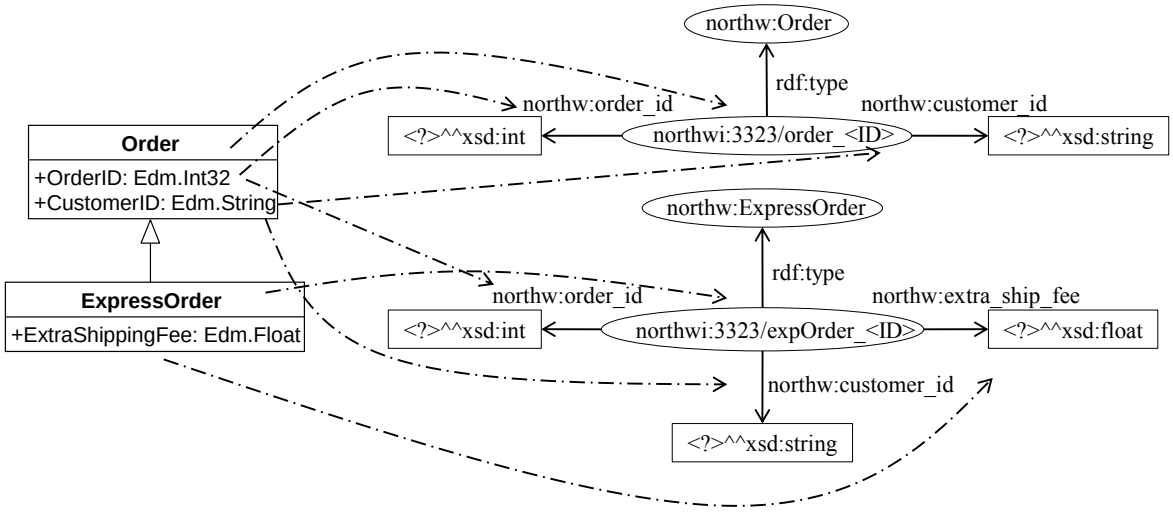


Abbildung 23: Abbildung zweier Entitätstypen einer Typhierarchie auf zwei Ressourcen-Templates.

Auf der linken Seite von Abbildung 23 sind die beiden Entitätstypen *Order* und *ExpressOrder* dargestellt. Der Entitätstyp *ExpressOrder* repräsentiert alle Express-Bestellungen, die innerhalb eines Tages zugestellt werden müssen. Zusätzliche Kosten, welche durch die beschleunigte Zustellung anfallen, werden in der Eigenschaft *ExtraShippingFee* abgespeichert, um die der Basistyp erweitert wird. Die Abbildung erfolgt in Abhängigkeit des Entitätstyps. Entitäten des Entitätstyps *Order* werden auf Ressourcen des Typ *northw:Order* abgebildet. *ExpressOrder*-Entitäten hingegen werden in Ressourcen des Typs *northw:ExpressOrder* überführt. Wobei bei der Abbildung der *ExpressOrder*-Entitäten neben der *ExtraShippingFee*-Eigenschaft auch die Eigenschaften des Basistyps mit einbezogen werden müssen.

Diese Art der Abbildung kann mit den in Abschnitt 4.12 beschriebenen Konzepten spezifiziert werden. Für jeden Entitätstyp der Typhierarchie wird eine Variable definiert, welcher der entsprechende Ressourcen-Typ zugewiesen wird. Die Abbildungen auf die Aussagen-Templates erfolgt in Abhängigkeit der jeweiligen Variablen, wobei in den semantischen Erweiterungen der Basistypen auf die Variablen der abgeleiteten Typen Bezug genommen wird.

Listing 53 zeigt die semantische Beschreibung für das Beispiel aus Abbildung 23. Die mehrfachen Abbildungen der Eigenschaften des *Order*-Entitätstyps sind in den Zeilen 3,4,6 und 7 dargestellt. Dabei wird das Ressourcen-Template des Typs *northw:ExpressOrder* über die Variable *?expOrder* referenziert. Die Angabe erfolgt mit der in Abschnitt 4.12 definierten Syntax.

```
1. <EntityType Name="Order" sem:Type="?order=northw:Order">
2.  <Property Name="OrderID" Type="Edm.Int32"
3.   sem:Mapping="?order northw:order_id .
4.                ?expOrder northw:order_id" />
5.  <Property Name="CustomerID" Type="Edm.String"
6.   sem:Mapping="?order northw:customer_id .
7.                ?expOrder northw:customer_id" />
8. </EntityType>
9.
10.<EntityType Name="ExpressOrder"
11. BaseType="NorthwindModel.Order"
12. sem:Type="?expOrder=northw:ExpressOrder">
13. <Property Name="ExtraShippingFee" Type="Edm.Float"
14.  sem:Mapping="?expOrder northw:extra_ship_fee"/>
15.</EntityType>
```

Listing 53: Semantische Beschreibung einer Typ-Hierarchie.

Aufgrund der voneinander unabhängigen Abbildungen der Eigenschaften auf verschiedene Ressourcen-Templates besteht auch die Möglichkeit, die Eigenschaften auf Aussagen-Templates mit verschiedenen Prädikaten abzubilden oder einzelne Eigenschaften für bestimmte Ressourcen-Templates auszulassen. Die in diesem Abschnitt vorgestellten Konzepte können auch verwendet werden, um Hierarchien von komplexen Typen auf unterschiedlichen Ressourcen-Template abzubilden.

4.15 Abbildungen in Abhängigkeit der Entitätsmenge

Wie bereits erwähnt, können mehrere Entitätsmengen auf den gleichen Entitätstyp verweisen. Allerdings kann eine Entität eines bestimmten Entitätstyps immer nur Teil einer Entitätsmenge sein [134]. Abbildung 24 zeigt als Beispiel die beiden Entitätsmengen *CheapProducts* und *ExpensiveProducts*, welche die Entitäten des Entitätstyps *Product* in billige und teure Produkte unterteilt[24]. Die Entitätsmenge *CheapProducts* enthält alle Produkte, die weniger als 50 $ kosten. Produkte, die mehr als 50 $ kosten, sind in der Entitätsmenge *ExpensiveProducts* zusammengefasst.

[24] Diese Entitätsmengen sind nicht Teil des Northwind-Services. Sie werden hier nur zu Anschauungszwecken eingeführt.

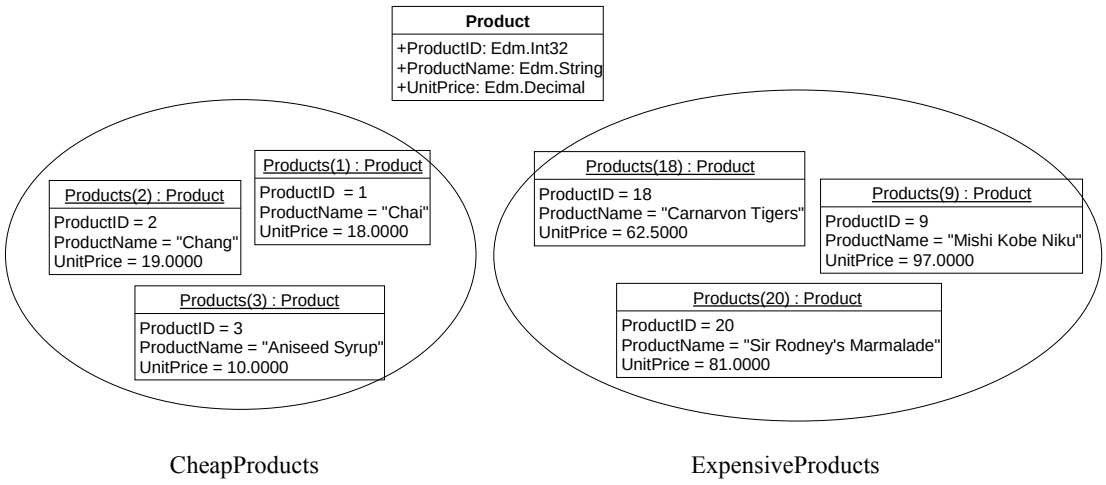


Abbildung 24: Mehrere Entitätsmengen des gleichen Entitätstyps.

Bisher wurden Abbildungen immer auf Ebene der Entitätstypen beschrieben. Um die Abbildung von Entitäten in Abhängigkeit der Entitätsmenge, der sie angehören, zu definieren, müssen die bisher vorgestellten Konzepte erweitert werden. Diese Erweiterungen sollen anhand des Beispiels aus Abbildung 25 hergeleitet werden.

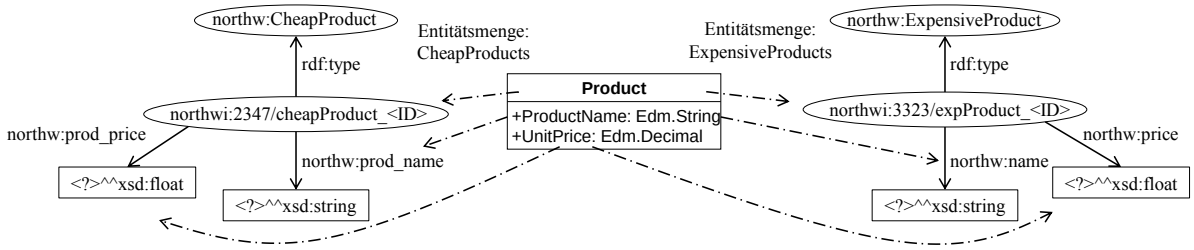


Abbildung 25: Abbildung eines Entitätstyps in Abhängigkeit der Entitätsmengen.

Der Entitätstyp *Product* wird von den bereits vorgestellten Entitätsmengen *CheapProducts* und *ExpensiveProducts* referenziert. Die Abbildung auf die entsprechenden RDF-Graph-Templates erfolgt in Abhängigkeit der Entitätsmengen. Entitäten, die der Entitätsmenge *CheapProducts* angehören, werden in Ressourcen des Typs *northw:CheapProduct* überführt (linke Seite von Abbildung 25). Entitäten der Entitätsmenge *ExpensiveProducts* hingegen werden auf *northw:ExpensiveProduct*-Ressourcen abgebildet (rechte Seite von Abbildung 25). Dabei erfolgt auch die Abbildung der Eigenschaften in Abhängigkeit der Entitätsmengen. Die Eigenschaft *ProductName* z.B. wird, je nach Entitätsmenge, entweder auf eine Aussage mit dem Prädikat *northw:prod_name* oder auf eine Aussage mit dem Prädikat *northw:name* abgebildet.

Um solch eine Abbildung beschreiben zu können, ist die alleinige semantische Erweiterung des Entitätstyps nicht ausreichend. Stattdessen muss als Teil der semantischen Beschreibung eine Referenzierung zwischen dem Entitätstyp und der jeweiligen Entitätsmenge hergestellt werden. Da der Typ der Ressource, auf welche die Entitäten abgebildet werden, abhängig von der jeweiligen Entitätsmenge ist, bietet es sich an, die Typ-Angabe als semantische Erweiterung des *EntitySet*-Elements und nicht des *EntityType*-Elements zu spezifizieren. Listing 54 zeigt dies für das obige Beispiel.

```
1. <EntityType Name="Product">
2.  <Property Name="ProductName" Type="Edm.String"
3.   sem:Mapping="?cheapProduct northw:prod_name .
4.                ?expProduct northw:name" />
5. </EntityType>
6.
7. <EntitySet Name="CheapProducts"
8.  EntityType="NorthwindModel.Product"
9.  sem:Type="?cheapProduct=northw:CheapProduct" />
10.<EntitySet Name="ExpensiveProducts"
11. EntityType="NorthwindModel.Product"
12. sem:Type="?expProduct=northw:ExpensiveProduct" />
```

Listing 54: Semantische Beschreibung zur Abbildung eines Entitätstyps in Abhängigkeit der Entitätsmengen.

In den Zeilen 9 und 12 erfolgt die Typ-Zuweisung für die jeweiligen Entitätsmengen. Wobei für jede Entitätsmenge eine Variable definiert wird (*?cheapProduct* und *?expProduct*). Die unterschiedlichen Abbildungen der Eigenschaften des *Product*-Entitätstyps können unter Verweis auf diese Variablen beschrieben werden. In den Zeilen 3 und 4 ist die semantische Beschreibung für die *ProductName*-Eigenschaft dargestellt.

Diese Art der Beschreibung ist nicht mehr eindeutig, wenn Typhierarchien in Abhängigkeit der Entitätsmengen auf unterschiedliche Ressourcen-Templates abgebildet werden sollen. Abbildung 26 zeigt ein Beispiel.

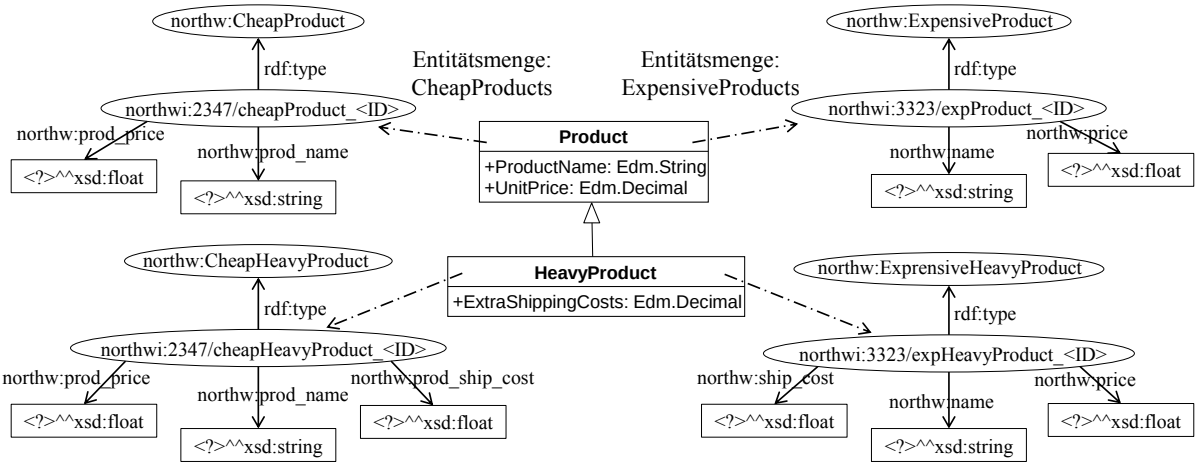


Abbildung 26: Abbildung einer Typhierarchie in Abhängigkeit der Entitätsmengen.

Von dem Entitätstyp *Product* wurde ein weiterer Entitätstyp *HeavyProduct* abgeleitet[25]. Dieser Entitätstyp beschreibt alle Produkte, deren Auslieferung aufgrund ihres hohen Gewichts zusätzliche Kosten verursachen. Diese aus zwei Typen bestehende Typhierarchie wird in Abhängigkeit der Entitätsmengen auf unterschiedliche RDF-Graph-Templates

[25] Dieser zusätzliche Entitätstyp ist nicht Teil des Datenmodells des Northwind-Services.

abgebildet. Entitäten des Entitätstyps *Product* werden für die Entitätsmenge *CheapProducts* auf Ressourcen des Typs *northw:CheapProduct* abgebildet. *HeavyProduct*-Entitäten hingegen werden in Ressourcen des Typs *CheapHeavyProduct* überführt. Für die Entitäten der Entitätsmenge *ExpensiveProducts* erfolgt eine ähnliche Abbildung auf die Typen *ExpensiveProduct* und *ExpensiveHeavyProduct*. Wie in dem vorherigen Beispiel, werden auch bei diesem Beispiel die Eigenschaften der Entitäten in Abhängigkeit der Entitätsmengen und der Ressourcen-Typen auf Aussagen mit unterschiedlichen Prädikaten abgebildet.

Zur semantischen Beschreibung der Abbildung sind die in Listing 54 gezeigten Konzepte nicht ausreichend, da sie die Abhängigkeiten zu den Entitätstypen der Typhierarchie bei der Abbildung nicht berücksichtigen. Da die Entitäten der beiden Entitätstypen *CheapProducts* und *ExpensiveProducts* auf Ressourcen zweier unterschiedlicher Typen abgebildet werden, müssen die Typ-Angaben der *EntitySet*-Elemente zwei Variablen-Definition enthalten. Listing 55 zeigt dies in den Zeilen 19, 20, 23 und 24.

```
1. <EntityType Name="Product"
2.  sem:Type="?cheapProduct . ?expProduct" >
3.  <Property Name="ProductName" Type="Edm.String"
4.   sem:Mapping="?cheapProduct northw:prod_name .
5.                ?expProduct northw:name .
6.                ?cheapHeavyProduct northw:prod_name .
7.                ?expHeavyProduct northw:name" />
8. </EntityType>
9.
10.<EntityType Name="HeavyProduct" BaseType="Product"
11. sem:Type="?cheapHeavyProduct . ?expHeavyProduct">
12. <Property Name="ExtraShippingCosts" Type="Edm.Decimal"
13.  sem:Mapping="?cheapHeavyProduct northw:prod_ship_cost .
14.               ?expHeavyProduct northw:ship_cost" />
15.</EntityType>
16.
17.<EntitySet Name="CheapProducts"
18. EntityType="NorthwindModel.Product"
19. sem:Type="?cheapProduct=northw:CheapProduct .
20.           ?cheapHeavyProduct=northw:CheapHeavyProduct" />
21.<EntitySet Name="ExpensiveProducts"
22. EntityType="NorthwindModel.Product"
23. sem:Type="?expProduct=northw:ExpensiveProduct .
24.           ?expHeavyProduct=northw:ExpensiveHeavyProduct"
25./>
```

Listing 55: Semantische Beschreibung zur Abbildung einer Typhierarchie in Abhängigkeit der Entitätsmengen.

Diese Angaben spezifizieren, welche Entitätsmengen Ressourcen welcher Typen bereitstellen. Allerdings ist unklar, welcher Entitätstyp auf welchen Ressourcen-Typ abgebildet werden soll. Für das Beispiel aus Abbildung 25 besteht dieses Problem nicht, da jede Entitätsmenge nur Entitäten genau eines Typs enthält. Für das Beispiel aus Abbildung 26 hingegen sind zusätzliche Angaben erforderlich. Es muss ein Verweis zwischen den Angaben der Ressourcen-Typen der Entitätsmengen und den entsprechenden Entitätstypen hergestellt werden. Dies kann erreicht werden, indem als Teil der semantischen Beschreibung der Entitätstypen auf die Variablen der Entitätsmengen verwiesen wird. Da es sich um eine Typ-Angabe handelt, bietet es sich an, diesen Verweis als Teil des *sem:Typ*-Attributs des *EntityType*-Elements zu spezifizieren. Dabei ist nur die Angabe der Variablen notwendig, da der Typ für diese Variable bereits in der semantischen Erweiterung des *EntitySet*-Elements definiert wurde. Listing 55 zeigt dies für den Entitätstyp *Product* in der Zeile 2 und für den Entitätstyp *HeavyProduct* in der Zeile 11.

Ausgehend von den Entitätstypen sind damit die Abbildungen der Entitätstypen auf die unterschiedlichen Ressourcen-Templates eindeutig definiert. Wobei die Überführung der Eigenschaften auf die Aussagen-Templates unter Verweis auf die entsprechenden Variablen erfolgt (siehe Zeilen 4 – 7 und Zeilen 13 – 14).

4.16 Beschreibung von eingeschränkten Entitätsmengen

Um die Geschwindigkeit der Abfrageverarbeitung zu erhöhen, ist es sinnvoll, Informationen über die Eigenschaften der in einer Entitätsmenge enthaltenen Entitäten mit in die Beschreibung aufzunehmen. Angenommen, die beiden, aus dem vorherigen Abschnitt bekannten Entitätsmengen *CheapProducts* und *ExpensiveProducts* werden auf das gleiche Ressourcen-Template mit dem Typ *northw:Product* abgebildet. Wenn gegen diesen Graphen eine Anfrage nach allen Produkten durchgeführt werden soll, welche weniger als 20 $ kosten, könnte anhand dieser Informationen ermittelt werden, dass zur Beantwortung der Anfrage die Entitätsmenge *CheapProducts* abgefragt werden muss.

Mit den bisher vorgestellten Konzepten ist eine solche Beschreibung nicht möglich. Daher müssen zusätzliche Attribute definiert werden, welche die Spezifikation solcher einschränkenden Angaben erlauben. Dazu ist zunächst die Definition der Mächtigkeit und des Formats dieser Angaben notwendig, d.h. es muss festgelegt werden, welche Einschränkungen beschrieben werden sollen. Da die semantische Erweiterung von CSDL mit Hinblick auf die Verarbeitung von SPARQL-Anfragen erfolgt, sollte die Ausdrucksstärke dieser Angaben nicht mächtiger sein, als die Ausdrucksstärke der entsprechenden Operatoren in SPARQL selbst. Anderenfalls wären diese irrelevant für die Verarbeitung der Anfragen, da entsprechende Anfragen in SPARQL nicht formuliert werden könnten.

Wie in Abschnitt 2.2 beschrieben, stellt SPARQL mit dem *FILTER*-Operator eine Möglichkeit zur Filterung von Daten bereit. Dabei beschreiben die Filterausdrücke Einschränkungen der Datenmengen. Die Semantik und Syntax der Filterausdrücke ist durch die SPARQL-Spezifikation [62] auf Basis der in [131] definierten Operatoren und Funktionen von XQuery und XPath festgelegt. Es bietet sich daher an, SPARQL-Filterausdrücke auch zur Beschreibung von Entitätsmengen einzusetzen. Damit lassen sich die Entitätsmengen aus

Abbildung 24 mit den folgenden Filterausdrücken beschreiben: *$UnitPrice <= 50.0* (*CheapProducts*) und *$UnitPrice > 50.0* (*ExpensiveProducts*). Die Referenzierung der entsprechenden Eigenschaft erfolgt über den Namen der Eigenschaft, wobei diesem, wie bereits in Abschnitt 4.6 beschrieben, in Anlehnung an die SPARQL-Syntax ein Dollarzeichen vorangestellt wird. Zur Angabe der Einschränkungen im CSDL-Dokument muss ein neues Attribut definiert werden: *sem:Constraint*. Da die Einschränkungen für die jeweiligen Entitätsmengen gelten, wird das Attribut als Erweiterung des *EntitySet*-Elements definiert. Damit ergibt sich die in Listing 56 dargestellte semantische Erweiterung.

```
1. <EntityType Name="Product"
2.  sem:Type="?product=northw:Product">
3.  <Property Name="UnitPrice" Type="Edm.Decimal"
4.   sem:Mapping="?product northw:price"/>
5. </EntityType>
6.
7. <EntitySet Name="CheapProducts"
8.  EntityType="NorthwindModel.Product"
9.  sem:Constraint="$UnitPrice <= 50.0" />
10.<EntitySet Name="ExpensiveProducts"
11. EntityType="NorthwindModel.Product"
12. sem:Constraint="$UnitPrice > 50.0" />
```

Listing 56: Beschreibungen von Entitätsmengen mittels Filterausdrücken.

Da die Operatoren, wie Addition, Subtraktion, Größer, Kleiner usw., nur für XSD-Datentypen definiert sind, bezieht sich die Angabe der Einschränkung auf den Wert der Eigenschaft nach Abbildung auf den entsprechenden XSD-Datentyp, wie diese in Tabelle 5 definiert ist. Wenn, wie in Abschnitt 4.4 beschrieben, der Typ explizit mit dem *sem:Datatype*-Attribut auf einen Typen außerhalb des XSD-Namensraum festgelegt wurde, so ist das Verhalten nicht definiert bzw. implementierungsspezifisch.

Die SPARQL-Spezifikation definiert mehrere Operatoren, mit denen Werte auf verschiedene Eigenschaften abgefragt werden können, z.B. ob der Wert *A* ein Literal (*isLITERAL(A)*) ist. Diese Operatoren können auch zur Beschreibung von Entitätsmengen eingesetzt werden. Listing 57 zeigt ein Beispiel.

```
1. <EntityType Name="Order" sem:Type="northw:Order">
2.  <Property Name="ShippedDate" Type="Edm.DateTime"
3.   Nullable="true" sem:Mapping="shipped_date" />
4. </EntityType>
5.
6. <EntitySet Name="ShippedOrders"
7.  EntityType="NorthwindModel.Order"
8.  sem:Constraint="BOUND($ShippedDate)=true" />
```

Listing 57: Beschreibung der Einschränkung einer Entitätsmenge mit einem SPARQL-Operator.

Die Entitätsmenge *ShippedOrders* enthält alle Bestellungen (*Order*-Entitäten), die bereits versendet wurden, d.h. die für die Eigenschaft *ShippedDate* einen Wert aufweisen. Diese Einschränkung kann, wie in Zeile 8 dargestellt, über das *sem:Constraint*-Attribut mittels des *BOUND*-Operators angegeben werden.

Mit den logischen Operatoren *AND* (*&&*) und *OR* (||) besteht die Möglichkeit, mehrere Angaben miteinander zu verknüpfen. Wenn z.B. eine Entitätsmenge existiert, die alle Bestellungen enthält, die bereits abgeschickt wurden und deren Bestimmungsort innerhalb von Deutschland liegt, so kann dies mit folgendem Filterausdruck beschrieben werden: *BOUND($ShippedDate)=true && $ShipCountry="Germany"*. Mit dem logischen Oder-Operator kann angegeben werden, dass mindestens eine der beiden Bedingungen gilt. Die Beschreibung einer Entitätsmenge, die alle Bestellungen umfasst, welche nach Kanada oder in die USA geliefert werden, kann somit folgendermaßen ausgedrückt werden: *$ShipCountry="USA" || $ShipCountry="Canada"*.

Manche Angaben, die sich mittels der Operatoren ausdrücken lassen, wie z.B. mittels des *DATATYPE*-Operators, sind unnötig, da diese Angaben bereits implizit durch das CSDL-Dokuments gegeben sind. Beispielsweise impliziert die CSDL-Typ-Angabe einer Eigenschaft *A* in Form von *Type="Edm.Int32"* die Einschränkung *DATATYPE($A)=xsd:int*, sofern keine explizite Abbildung auf einen anderen Typen mit dem *sem:Datatype*-Attribut spezifiziert wurde.

Für Funktionen, die einen booleschen Wert zurückliefern, kann eine abkürzende Schreibweise definiert werden. Dabei werden sie zu Aussagen, die angeben, dass die entsprechende Funktion für die Eigenschaftswerte aller Entitäten dieser Entitätsmenge den Rückgabewert *true* liefert. Mit dieser Definition kann die Angabe *BOUND($ShippedDate)=true* abgekürzt als *BOUND($ShippedDate)* angegeben werden. Dadurch wird auch die Verwendung des in der SPARQL-Spezifikation definierten, unären Operators *NOT* (*!*) ermöglicht. Somit könnte die Angabe *!BOUND($ShippedDate)* zur Beschreibung einer Entitätsmenge verwendet werden, welche alle Bestellungen enthält, die noch nicht versandt wurden.

4.17 Reifikation

Als spezieller Anwendungsfall soll im Folgenden gezeigt werden, wie die bisher eingeführten Konzepte im Zusammenhang mit der Reifikation von Aussagen verwendet werden können. RDF stellt Konzepte zur Verfügung, um Aussagen über Aussagen treffen zu können (Reifikation) [140]. Dazu wird eine Ressource des Typs *rdf:Statement* erstellt, welche über die Prädikate *rdf:subject*, *rdf:predicate* und *rdf:object*, das Subjekt, Prädikat und das Objekt der zu reifizierenden Aussage spezifiziert. Um die Erzeugung von Aussagen über Aussagen als Teil der semantischen Erweiterung zu beschreiben, können die im vorherigen Abschnitt vorgestellten Konzepte verwendet werden. Dies soll anhand des Beispiels aus Abbildung 27 demonstriert werden.

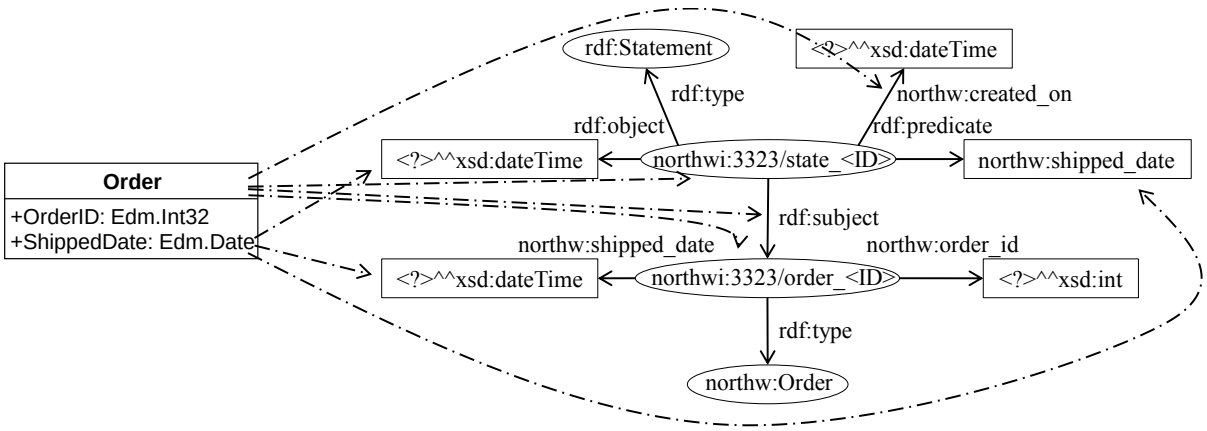


Abbildung 27: Abbildung auf ein Ressourcen- und ein Reifikations-Template.

Die linke Seite der Abbildung zeigt den *Order*-Entitätstyp mit den beiden Eigenschaften *OrderID* und *ShippedDate*. Aus jeder Entität dieses Typs wird eine Ressource des Typs *northw:Order* erstellt. Dabei werden die beiden Eigenschaften auf Aussagen mit den Prädikaten *northw:order_id* und *northw:shipped_date* abgebildet. Für jede Aussage, die über das *northw:shipped_date*-Prädikat das Auslieferungsdatum angibt, soll außerdem mit dem RDF-Reifikationsvokabular das Erstelldatum dieser Aussage spezifiziert werden. Dazu muss ein Ressourcen-Template des Typs *rdf:Statement* beschrieben werden, welches über das *rdf:subjekt*-Prädikat die entsprechende *northw:Order*-Ressource referenziert. Auf das Prädikat *northw:shipped_date* wird über eine Aussage mit dem Prädikat *rdf:predicate* verwiesen. Die Angabe des Objekts über die Aussage mit dem Prädikat *rdf:object* wird über ein Aussagen-Template beschrieben, dessen Wert abhängig von der entsprechenden Eigenschaft der Entitäten ist. Die Angabe des Erstelldatums der Aussage erfolgt als Aussage mit der *rdf:Statement*-Ressource als Subjekt und *northw:created_on* als Prädikat. Dabei kann das aktuelle Datum z.B. mit der XPath-Funktion *fn:current-date()* ermittelt werden (siehe Abschnitt 4.6). Listing 58 zeigt die semantische Beschreibung für dieses Beispiel.

```
1. <EntityType Name="Order" sem:Type="?order=northw:Order .
2.                                    ?state=rdf:Statement"
3.  sem:Mapping="?state rdf:subject ?order .
4.               ?state northw:created_on fn:current-date()">
5.  <Property Name="ShippedDate" Type="Edm.DateTime"
6.   sem:Mapping="?order northw:shipped_date .
7.                ?state rdf:predicate northw:shipped_date .
8.                ?state rdf:object " />
9. </EntityType>
```

Listing 58: Beschreibung der Reifikation.

Die Beschreibung der Referenzierung der *northw:Order*-Ressource durch die *rdf:Statement*-Ressource über das *rdf:subject*-Prädikat erfolgt in Zeile 3. Die Prädikat- und Objektangabe sind in Zeile 7 und 8 beschrieben. Die Beschreibung der Aussage über die Aussage in Form eines Aussagen-Templates mit dem Prädikat *northw:created_on* erfolgt als Teil der semantischen Erweiterung des Entitätstyps in Zeile 4.

4.18 Abbildung auf Listen

Wie in Abschnitt 2.1 erläutert, stellt RDF Konzepte für die Beschreibung von Listen bereit. Dabei wird zwischen offenen (Containern) und geschlossenen (Collections) Listen unterschieden. In diesem Abschnitt soll erläutert werden, wie die Abbildung auf Listen beschrieben werden kann. Abbildung 28 zeigt die Abbildung mehrerer Eigenschaften auf eine offene Liste.

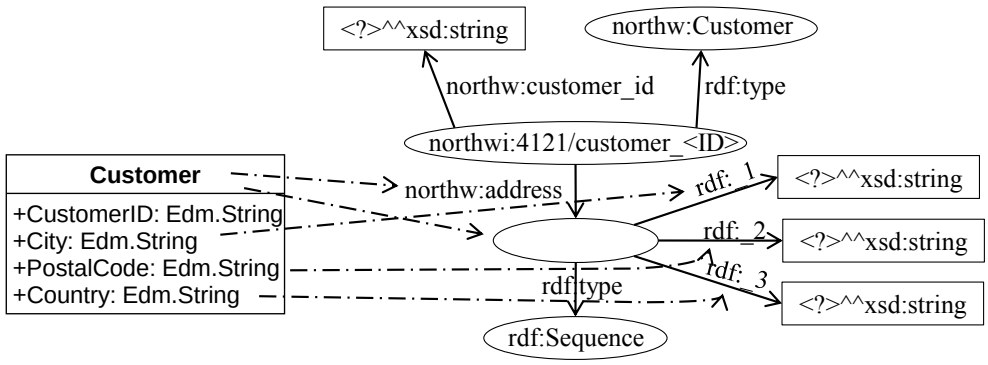


Abbildung 28: Abbildung mehrerer Eigenschaften auf eine offene Liste.

Der Entitätstyp *Customer* enthält mehrere Eigenschaften, welche die Adresse angeben (*City*, *PostalCode*, *Country*). Diese Eigenschaften sollen auf eine offene, geordnete Liste (*Sequence*) abgebildet werden. Die Stadt (*City*) kommt an erster Stelle. Auf diese folgt die Postleitzahl (*PostalCode*). Das Land (*Country*) wird an die dritte Stelle eingefügt. Diese Art der Abbildung lässt sich bereits mit den in den vorherigen Abschnitten vorgestellten Konzepten beschreiben. Listing 59 zeigt die dazugehörige semantische Beschreibung.

```
1. <EntityType Name="Customer"
2.  sem:Type="?customer=northw:Customer . _:cont=rdf:Seq"
3.  sem:Mapping="?customer northw:address _:cont">
4.  <Property Name="City" Type="Edm.String"
5.   sem:Mapping="_:cont rdf:_1" />
6.  <Property Name="PostalCode" Type="Edm.String"
7.   sem:Mapping="_:cont rdf:_2" />
8.  <Property Name="Country" Type="Edm.String"
9.   sem:Mapping="_:cont rdf:_3" />
10.</EntityType>
```

Listing 59: Beschreibung der Abbildung mehrerer Eigenschaften auf eine offene Liste.

Die Liste wird als leerer Knoten beschrieben, der dem Typ *rdf:Seq* zugewiesen wird (Zeile 2). Die Zuweisung der einzelnen Listenelemente erfolgt wie in Abschnitt 4.4 erläutert (Zeilen 5, 7 und 9).

Zur Angabe von geschlossenen Listen (*Collections*) stellt Turtle eine spezielle Syntax zur Verfügung (siehe Abschnitt 2.1.1). Da sich mit dieser Syntax geschlossene Listen sehr kompakt ausdrücken lassen, ohne dass die explizite Angabe von leeren Knoten erforderlich ist, bietet es sich an, diese zur Beschreibung der Abbildung zu übernehmen. Dies soll anhand des in Abbildung 29 dargestellten Beispiels gezeigt werden. Das dargestellte Beispiel zeigt die Abbildung der Eigenschaften des bereits aus dem vorherigen Beispiel bekannten Entitätstypen auf eine geschlossene Liste.

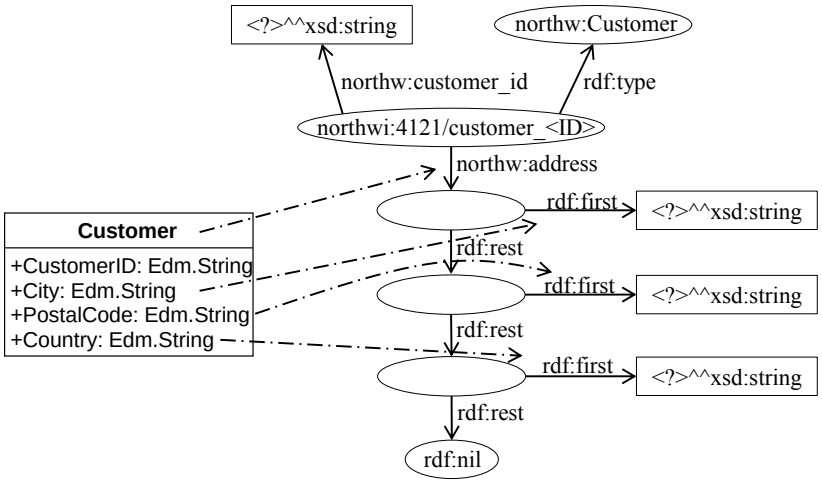


Abbildung 29: Abbildung auf eine geschlossene Liste.

Die Abbildung der Adressen-Angaben auf eine geschlossene Liste kann nun mittels der Turtle-Syntax in Form runder Klammern erfolgen, wobei die entsprechenden Eigenschaften über die Namen referenziert werden. Die Beschreibung der Abbildung wird als Erweiterung des *EntityType*-Elements aus Zeile 1 in Listing 59 über folgenden Ausdruck spezifiziert: *sem:Mapping="?customer northw:address ($City $PostalCode $Country)"*.

4.19 Definitionen

Mit den in diesem Kapitel hergeleiteten Erweiterungen von CSDL kann die Abbildung von OData-Services zugrundeliegenden Entitätsdatenmodellen auf RDF-Graphen beschrieben werden. Eine formale Beschreibung dieser Erweiterungen mit XSD ist in Anhang B zu finden. Die Beschreibung besteht aus zwei Schemata. Das erste Schema definiert die Attribute und deren Datentypen. Das zweite Schema stellt die Beziehung zu den CSDL-Elementen her. Dazu wurden die entsprechenden Typen aus dem CSDL mit dem XSD-Redefine-Mechanismus erweitert. Zwar ist diese Erweiterung nicht zwingend erforderlich, da das CSDL-Schema die Erweiterung der Elemente um beliebige Attribute bereits gestattet, allerdings wird durch diese Erweiterung die Beziehung der Attribute zu den Elementen eindeutig festgelegt.

In Vorbereitung auf die Herleitung der Evaluierungssemantik für Graph-Templates im nächsten Kapitel sollen die Begrifflichkeiten zur Beschreibung der vorgestellten Konzepte definiert und zusammengefasst werden. Die semantischen Annotationen eines CSDL-Dokuments definieren ein **Graph-Template**, das den RDF-Graphen beschreibt, der potentiell aus dem vom OData-Service bereitgestellten Daten erzeugt werden kann. Das Graph-Template eines Services setzt sich aus den Graph-Templates der Entitätsmengen zusammen. Diese wiederum geben an, welche RDF-Graphen potentiell aus den Entitäten der Entitätsmengen erstellt werden können.

Ein Graph-Template besteht aus ein oder mehreren **Ressourcen-Templates**. Ein Ressourcen-Template beschreibt die Abbildung auf Ressourcen eines bestimmten Typs. Als Beispiel zeigt Listing 37 aus Abschnitt 4.5 die semantische Beschreibung eines Entitätstyps, welche zwei Ressourcen-Templates der Typen *northw:Order* und *northw:Address* definiert. Die beiden Ressourcen-Templates bilden dabei ein Graph-Template der Entitätsmenge *Orders*, welche den *Order*-Entitätstyp referenziert.

Ressourcen-Templates setzen sich aus einer beliebigen Menge an **Tripel-Templates** (z.B. *?order northw:order_addr ?address*) zusammen. Ein Tripel-Template ist die Vorlage für eine Menge von RDF-Tripeln. Es wird über das *sem:Mapping*- oder das *sem:Type*-Attribut spezifiziert und beschreibt, in Abhängigkeit seiner Position im EDM-Datenmodell und zu anderen Erweiterungsattributen, wie diese aus den Daten des Services erzeugt werden. Ein Tripel-Template besteht aus einem **Subjekt-** (*?order*), einem **Prädikat-** (*northw:order_addr*) und einem **Objekt-Template** (?*address*). Diese bilden die Vorlage für die Subjekte, Prädikate und Objekte der RDF-Tripel. Bei der Herleitung der Konzepte in diesem Kapitel wurden mehrere abkürzende Schreibweisen definiert, bei der zur Vereinfachung der semantischen Annotation die Subjekt-, Prädikat- oder Objekt-Templates nicht immer explizit angegeben werden müssen (z.B. *?order northw:order_id***)**. In diesem Fall ist das fehlende Template implizit, entsprechend den Erläuterungen dieses Kapitels, gegeben. Ein formaler Algorithmus zur Auflösung der Abkürzungen wird im nächsten Kapitel in Abschnitt 5.2 hergeleitet.

Bei einem Subjekt-Template handelt es sich entweder um ein **URI-Template** oder um ein **Leerer-Knoten-Template**. Das URI-Template (*?order*) ist ein Platzhalter für eine Ressourcen-URI und das Leerer-Knoten-Template (z.B. *_:orderAddress* in Listing 40) steht für die ID eines leeren Knotens. URI-Templates und Leere-Knoten-Templates werden auch als **Ressourcen-Variablen** bezeichnet. Ein Prädikat-Template ist immer eine URI (*northw:order_addr*). Daher sind die Prädikate aller Tripel, die durch ein Tripel-Template beschrieben werden, immer gleich. Das Objekt-Template eines Tripel-Templates kann zur Beschreibung der Referenzierung einer anderen Ressource einem URI-Template oder einem Leerer-Knoten-Template entsprechen.

Wenn die Objekte der Tripel dynamisch berechnet werden sollen, erfolgt die Angabe des Objekt-Templates in Form eines Funktionsausdrucks (z.B. *fn:years-from-duration(op:subtract-dateTimes($HireDate,$BirthDate))* in Listing 39). Ein solcher Funktionsausdruck wird im Folgenden als **Funktions-Template** bezeichnet. Referenzen auf Eigenschaften dienen als Platzhalter für Literale (*$BirthDate*). Sie werden daher als **Literal-Templates** bezeichnet und können ebenfalls das Objekt-Template eines Tripel-Templates bilden. Wobei diese oftmals nicht explizit angegeben werden müssen. Neben den Objekt-Templates, die Objekte beschreiben, welche zur Laufzeit ermittelt werden, kann ein Objekt-

Template auch ein Literal oder eine URI fest vorgeben. Letzteres ist insbesondere bei der Beschreibung von Reifikationen von Bedeutung (z.B. *?state rdf:predicate northw:shipped_date* in Listing 58).

Ausgehend von diesen Erläuterungen erfolgt eine Formalisierung der Begrifflichkeiten. Der Formalismus ist dabei an andere Formalismen für RDF und SPARQL angelehnt, wie sie z.B. von Pérez et al. [32, 148] und von Glimm et al. [85] verwendet werden. Sei U die Menge aller URIs, L die Menge aller RDF-Literale, UT die Menge aller URI-Templates, BT die Menge aller Leerer-Knoten-Templates, FT die Menge aller Funktions-Templates und LT die Menge aller Literal-Templates. Die Menge aller Subjekt-Templates ist gegeben durch $ST = UT \cup BT$. Die Menge der Prädikat-Templates entspricht der Menge der URIs: $PT = U$. Den vorherigen Erläuterungen folgend, ist die Menge der Objekt-Templates definiert durch $OT = UT \cup BT \cup FT \cup LT \cup U \cup L$. Die Menge der Ressource-Variablen setzt sich aus der Menge der URI-Templates und Leerer-Knoten-Templates zusammen: $RV = UT \cup BT$. Ein Tupel $tt = (st, pt, ot) \in TT = ST \times PT \times OT$ ist ein Tripel-Template. Wobei TT der Menge aller Tripel-Templates entspricht. Für ein Tripel-Template $tt \in TT$ ist das Subjekt-, Prädikat- und Objekt-Template gegeben durch $tt.s$, $tt.p$ und $tt.o$. Wenn i ein EDM-Element ist (Entitätsmenge, Entitätstyp, komplexer Typ, Eigenschaft oder Navigationseigenschaft), dann ist $i.TT$ die Menge aller Tripel-Templates, die als Teil der semantischen Annotation für dieses Element definiert wurden. Dazu gehören auch alle Tripel-Templates, die über das *sem:Type*-Attribute spezifiziert wurden, wobei diese entsprechend den Erläuterungen in Abschnitt 5.2 aufgelöst werden.

4.20 Zusammenfassung

In diesem Kapitel wurden Konzepte hergleitet, um Abbildungen von Entitätsdatenmodellen auf RDF-Graphen beschreiben zu können. Dazu wurde CSDL um zusätzliche Attribute erweitert, welche die Spezifikation sogenannter Graph-Templates ermöglichen. Das Graph-Template eines Services beschreibt, den RDF-Graphen, der aus den vom Service zur Verfügung gestellten Daten generiert werden kann.

Die Herleitung erfolgte, ausgehend von den verschiedenen Konstrukten des Entitätsdatenmodells und den Konzepten von RDF, anhand verschiedener Beispiele. Zunächst wurde ein Konzept zur Erzeugung der Ressourcen-URIs hergeleitet, welches sicherstellt, dass die URIs alle Informationen enthalten, die notwendig sind, um die zur Erzeugung der Aussagen über die Ressourcen notwendigen Daten von dem OData-Service abrufen zu können. Des Weiteren wurden mögliche Formate der Ressourcen-URIs diskutiert und ein Erweiterungsattribut (*sem:URI_struc*) eingeführt, über welches das gewünschte Format spezifiziert werden kann.

Es wurde gezeigt, wie die Abbildung von Entitätstypen und deren Eigenschaften auf Ressourcen beschrieben werden kann. Dazu wurden die beiden Attribute *sem:Type* und *sem:Mapping* eingeführt. Das *sem:Mapping*-Attribut erlaubt die Spezifikation von beliebigen Tripel-Templates, welche die zu erzeugenden RDF-Tripel beschreiben. Dabei erfolgt die Angabe der Tripel-Templates mittels einer erweiterten Turtle-Syntax, wie sie auch von SPARQL zur Spezifikation der einfachen Graph-Muster verwendet wird. In der Regel ist es

nicht notwendig, das Tripel-Template vollständig anzugeben. Stattdessen wurden zahlreiche abkürzende Schreibweisen definiert, welche die semantische Annotation eines CSDL-Dokuments vereinfachen. Dazu gehört auch das *sem:Type*-Attribut, das die Angabe von Typ-Angaben in Form von Tripel-Templates mit dem *rdf:type*-Prädikat ermöglicht.

Im Rahmen der Herleitung der Abbildung eines Entitätstyps auf ein Ressourcen-Template wurden Attribute definiert, mit der den Literalen der zu erzeugenden RDF-Tripel ein Datentyp (*sem:Datatype*) und eine Sprachangabe (*sem:Lang*) zugewiesen werden kann. Um die einfache Verwendung der häufig in diesem Zusammenhang verwendeten XSD-Datentypen zu unterstützen, wurde ein Mapping von den einfachen Datentypen des Entitätsdatenmodells auf die XSD-Datentypen definiert. Auf Basis der Konzepte zur Abbildung eines Entitätstypen auf ein Ressourcen-Template wurde gezeigt, wie ein Entitätstyp auf mehrere Ressourcen-Templates abgebildet werden kann. Um die Objekte von Aussagen dynamisch zur Laufzeit berechnen zu können, wurde anhand der XQuery- und XPath-Funktionen gezeigt, wie Funktionen bei der Beschreibung der Abbildung mit einbezogen werden können. Dabei kann eine konkrete Implementierung auch Funktionen außerhalb der XQuery- und XPath-Namensräume unterstützen.

Die in diesem Kapitel vorgestellten Konzepte erlauben die Abbildung auf leere Knoten und RDF-Listen. Es wurde gezeigt, wie als Teil der Abbildung, mittels des RDF-Reifikations-Vokabulars, Aussagen über Aussagen beschrieben werden können. Für komplexe Typen wurde das Konzept der Ressourcen-URIs dahingehend erweitert, dass die Abbildung von Instanzen der komplexen Typen auf Ressourcen ermöglicht wird. Des Weiteren wurde definiert, wie Navigationseigenschaften in die Abbildung einbezogen werden können. Es besteht die Möglichkeit, mehrere, über Navigationseigenschaften miteinander verbundene Entitätstypen auf ein einzelnes Ressourcen-Template abzubilden oder die aus den Entitätstypen erzeugten Ressourcen über Aussagen in Beziehung zu setzen. Dabei können die in die Abbildung involvierten Navigationseigenschaften explizit spezifiziert werden oder sie werden automatisch, anhand der kürzesten Verbindung zwischen den Entitätstypen, ermittelt.

Das Entitätsdatenmodell ermöglicht die Definition von Typhierarchien. Es wurde gezeigt, wie die Abbildung einer Typ-Hierarchie in Abhängigkeit der einzelnen Typen auf verschiedene Ressourcen-Templates beschrieben werden kann. Da mehrere Entitätsmengen auf den gleichen Entitätstypen bzw. auf die gleiche Typ-Hierarchie verweisen können, wurden die Konzepte dahingehend erweitert, um eine Entitätsmengen-abhängige Abbildung zu ermöglichen. Des Weiteren wurde gezeigt, wie über *Constraints* die von einer Entitätsmenge zurückgegebenen Daten genauer beschrieben werden können, um die Geschwindigkeit der Abfrageverarbeitung zu erhöhen. Abschließend wurden die eingeführten Konzepte in Vorbereitung auf die im nächsten Kapitel beschriebene Herleitung der Evaluierungssemantik für Graph-Templates zusammengefasst und formalisiert.

5 Abbildung von SPARQL-Anfragen auf OData-Service-Aufrufe

In diesem Abschnitt wird ein Verfahren hergeleitet, mit dem SPARQL-Anfragen auf OData-Service-Aufrufe abgebildet werden können (siehe dazu auch [107]). Dieses Verfahren ermöglicht für eine gegebene SPARQL-Anfrage, mit Hinblick auf eine Menge von OData-Services, die Ermittlung aller OData-URIs, welche die zur Abarbeitung einer SPARQL-Anfrage benötigten Daten adressieren. Dabei müssen die OData-Services nach den in Kapitel 4 vorgestellten Konzepten semantisch beschrieben sein.

Mit Bezug auf das Verfahren ergeben sich verschiedene Anforderungen. Zum einen muss die Menge der abgerufenen Daten vollständig sein, d.h., es müssen alle Daten abgerufen werden, die potenziell zur Lösungsmenge beitragen könnten. Zum anderen muss die Abarbeitung der Anfragen möglichst effizient erfolgen, da für den praktischen Einsatz des SPARQL-Endpunktes die Performance von entscheidender Bedeutung ist. Die Performance des Gesamtsystems kann bei der Bestimmung der relevanten OData-URIs im Wesentlichen über zwei Punkte beeinflusst werden. Zum einen sollte die Ermittlung der OData-URIs möglichst performant erfolgen, da die relevanten OData-URIs für jede Anfrage zur Laufzeit ermittelt werden müssen. Des Weiteren sollten die ermittelten OData-URIs selbst sowohl hinsichtlich der Anzahl der benötigten Service-Aufrufe als auch hinsichtlich der zu übertragenden Datenmenge minimal sein, d.h. es sollten nur die Daten abgerufen werden, die zur Abarbeitung der SPARQL-Anfrage tatsächlich benötigt werden. Das kann erreicht werden, indem bei der Erstellung der URIs OData-spezifischen Konzepte, wie z.B. Abfrageoptionen, berücksichtigt werden.

Als einleitendes Beispiel zur Überführung einer SPARQL-Anfrage in OData-URIs soll die in Listing 60 dargestellte SPARQL-Anfrage dienen. Soweit nicht anders erläutert, beziehen sich die in diesem Kapitel gezeigten Beispiele auf das in Anhang C als Teil des semantisch erweiterten Service-Metadaten-Dokuments des Northwind-Services angegebenen Mappings. Es sei darauf hingewiesen, dass für den gesamten Service aus Gründen der Übersichtlichkeit über das *sem:Datatype*-Attribut die Abbildung auf untypisierte Literale spezifiziert ist. Die Anfrage in Listing 60 gibt die IDs aller Bestellungen zurück, welche in die Stadt Bern geliefert werden sollen. Dabei wird für jede Bestellung auch der Name des zu beliefernden Kunden zurückgegeben.

```
1. SELECT ?orderID ?customerName
2. WHERE {
3.  ?order rdf:type northw:Order .
4.  ?order northw:ship_city "Bern" .
5.  ?order northw:order_id ?orderID .
6.  ?order northw:customer ?customer .
7.  ?customer northw:company_name ?customerName
8. }
```

Listing 60: Eine Beispiel-SPARQL-Abfrage.

Eine mögliche Orchestrierung, die alle zur Beantwortung der Anfrage benötigten Daten abruft, wäre die folgende: Zunächst werden alle Entitäten der Entitätsmenge *Orders* abgerufen (*http://services.odata.org/Northwind/Northwind.svc/Orders*). Alle *Order*-Entitäten, die für die *ShipCity*-Eigenschaft einen Wert ungleich *Bern* aufweisen, werden verworfen. Für jede der verbleibenden *Order*-Entitäten wird über das entsprechende *link*-Element die URI des mit dieser Bestellung in Beziehung stehenden Kunden ermittelt (z.B. *http://services.odata.org/Northwind/Northwind.svc/Orders(10254)/Customer*). Die so ermittelten Kunden werden anschließend abgerufen. Aus den zurückgegebenen Daten wird, auf Basis des in Anhang C definierten Mappings, der zur Beantwortung der Anfrage benötigte RDF-Graph erzeugt. Dieser ist vollständig in dem Sinne, dass der RDF-Graph alle Daten enthält, die für die Abarbeitung der Anfrage relevant sein könnten.

Obwohl diese Orchestrierung ein korrektes Ergebnis liefert, ist sie unter Berücksichtigung der OData-spezifischen Abfragesprache, weder hinsichtlich der übertragenen Datenmenge noch in Bezug auf die Anzahl der Service-Aufrufe optimal. Es werden sowohl für die Bestellungen (z.B. *ShipCountry*), als auch für die Kunden (z.B. *Phone*) Daten zurückgegeben, die für die Abarbeitung der Anfrage nicht benötigt werden. Des Weiteren steigt die Anzahl der benötigten OData-Service-Aufrufe linear mit der Anzahl der zurückgegebenen *Order*-Entitäten, da für jede *Order*-Entität eine weitere Anfrage durchgeführt werden muss, um die zugehörige *Customer*-Entität abzurufen. Somit würde sich beispielsweise für die Menge von 1000 *Order*-Entitäten 1001 HTTP-Anfragen ergeben (1 Anfrage zur Abfrage der Bestellungen und 1000 Anfragen zur Abfrage der Kunden).

Wie in Abschnitt 2.3 erläutert, stellt OData mit der *filter*-Anweisung die Möglichkeit bereit, die zurückzugebende Datenmenge serverseitig zu filtern. Somit kann die Menge der *Order*-Entitäten bereits auf Seite des Servers auf diejenigen eingeschränkt werden, die nach Bern geliefert werden sollen *($filter=ShipCity eq 'Bern'*). Des Weiteren können über die *expand*-Anweisung miteinander verlinkte Entitäten über eine einzige HTTP-Anfrage abgerufen werden. Die zu den Bestellungen gehörenden Kunden können daher über die Abfrage der Bestellungen mittels der Anweisung *$expand=Customer* abgerufen werden. Mit der *select*-Anweisung besteht außerdem die Möglichkeit, die Menge der zurückgegebenen Eigenschaften auf die tatsächlich für die Verarbeitung der SPARQL-Anfrage benötigten Eigenschaften einzuschränken. Aus der in Anhang C angegebenen Mapping-Definition kann ermittelt werden, dass für die obige SPARQL-Anfrage nur die Eigenschaften *OrderID* und *ShipCity* der *Order*-Entitäten und die Eigenschaft *CompanyName* der *Customer*-Entitäten benötigt werden. Damit ergibt sich folgende *select*-Anweisung: *$select=OrderID,ShipCity,Customer/CompanyName*. Diese Anfrageoptionen können in einer einzigen Anfrage zusammengefasst werden: *http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=ShipCity eq 'Bern'&$expand=Customer&$select=OrderID,ShipCity,Customer/CompanyName*.

Durch die Berücksichtigung der OData-spezifischen Abfragesprache lässt sich somit in diesem Fall eine Reduzierung der Anzahl der benötigten Service-Aufrufe von 1001 Anfragen (bei 1000 *Order*-Entitäten) auf eine einzige Anfrage erreichen. Des Weiteren wird die übertragene Datenmenge durch das Selektieren der Eigenschaften erheblich reduziert.

5.1 Ermittlung der OData-URIs auf Basis der SPARQL-Algebra

Die Semantik von SPARQL ist durch die SPARQL-Algebra formalisiert (siehe Abschnitt 2.2.2). Jede SPARQL-Anfrage kann in einen Ausdruck der SPARQL-Algebra überführt werden. Dabei handelt es sich um einen Baum, der, ausgehend von den Blättern (*BGP*), entsprechend der SPARQL-Evaluierungssemantik (siehe Abschnitt 2.2.3) aufgelöst wird. Um eine vollständige Abdeckung von SPARQL zu erreichen und um Konformität mit der formalen Semantik zu garantieren, basiert das in diesem Kapitel vorgestellte Verfahren auf der SPARQL-Algebra. Abbildung 30 zeigt den generellen Ablauf bei der Überführung einer SPARQL-Anfrage in eine Menge OData-Service-Aufrufe. Die SPARQL-Anfrage wird zunächst in einen Ausdruck der SPARQL-Algebra überführt. Dieser SPARQL-Algebra-Baum dient als Eingabe für die Query Engine, welche auf Basis der semantisch erweiterten Service-Metadaten-Dokumente (siehe Kapitel 4) mittels der in diesem Kapitel vorgestellten Graph-Template-Evaluierungssemantik die relevanten OData-URIs ermittelt.

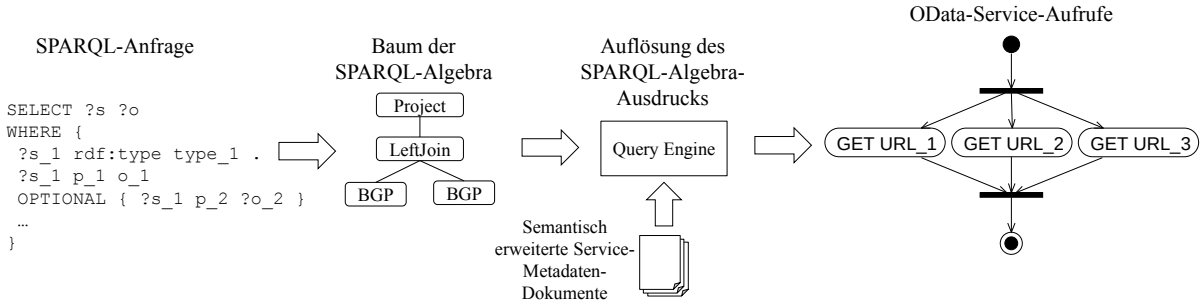


Abbildung 30: Vorgehensweise bei der Überführung von SPARQL-Anfragen nach OData-URIs.

Die mit Bezug auf Graph-Templates angepasste Evaluierungssemantik, welche die Auflösung von SPARQL-Anfragen gegen Graph-Templates definiert, basiert dabei auf der SPARQL-Evaluierungssemantik, wie sie durch die SPARQL-Spezifikation spezifiziert ist. Bezüglich der Berücksichtigung von Inferenzmechanismen, sei darauf hingewiesen, dass die Möglichkeit besteht, die Auswertung von SPARQL-Anfragen hinsichtlich bestimmter Semantiken, wie z.B. der von OWL, durch die Definition sogenannter *Entailment Regimes* zu erweitern [85]. Mittels existierender Verfahren [101] ist es aber möglich, die SPARQL-Anfragen automatisch anzupassen, um eine Auswertung hinsichtlich bestimmter Semantiken zu erreichen, ohne dass eine Änderung des Auswertungsverfahrens notwendig ist. Zum Beispiel könnte mit dem von Jing et al. beschriebenen Ansatz [101] die SPARQL-Anfrage, bevor sie von der Query Engine (siehe Abbildung 2) analysiert wird, der OWL-DL Semantik entsprechend, umgeschrieben werden. Dies würde für den Client transparent erfolgen und zu der Rückgabe aller Daten führen, die für die Ermittlung der vollständigen Ergebnismenge in Bezug auf OWL-DL erforderlich sind. Eine Anpassung des Auswertungsverfahrens ist dabei nicht notwendig. Aus diesem Grund wird die Auswertung hinsichtlich bestimmter Semantiken in dieser Arbeit nicht weiter betrachtet.

Der erste Schritt bei der Ermittlung der OData-URIs besteht darin, die semantisch erweiterten Service-Metadaten-Dokumente zu parsen. Als Teil dieses Prozesses müssen Abkürzungen, wie sie nach Kapitel 4 erlaubt sind, aufgelöst werden. Auf Basis der ermittelten

Tripel-Templates werden im nächsten Schritt die RDF-Graph-Templates der registrierten Services bestimmt. Dabei werden die Tripel-Templates, welche als Teil eines semantisch erweiterten Service-Metadaten-Dokuments spezifiziert wurden, zusammengefasst. Außerdem werden die Informationen darüber, welche vom Service bereitgestellten Daten abgefragt werden müssen, um die durch die Tripel-Templates beschriebenen Tripel erzeugen zu können, als Teil der Tripel-Templates abgespeichert.

Anschließend wird die SPARQL-Anfrage in einen Ausdruck der SPARQL-Algebra überführt. Dieser SPARQL-Algebra-Ausdruck wird auf Basis einer speziellen Evaluierungssemantik gegen die Graph-Templates der Services aufgelöst. Dabei werden sogenannte Lösungs-Template-Mappings ermittelt, welche mögliche Lösungen für die durch die Tripel-Templates beschriebenen Tripel beschreiben. Für jedes Lösungs-Template-Mapping werden, ausgehend von den Tripel-Templates, auf denen das Lösungs-Template-Mapping basiert, OData-URIs erzeugt. Diese OData-URIs adressieren alle Daten, die zur Bestimmung der durch das Lösungs-Template-Mapping beschriebenen Lösungen benötigt werden. Im Rahmen der Auflösung des SPARQL-Algebra-Baums werden alle Lösungs-Template-Mappings eliminiert, für die es entsprechend der SPARQL-Evaluierungssemantik keine Lösungen geben kann. Die nach der vollständigen Auflösung des SPARQL-Algebra-Ausdrucks bestehenden Lösungs-Template-Mappings beschreiben alle möglichen Lösungen. Wobei die assoziierten OData-URIs alle Daten adressieren, die zur vollständigen Bestimmung der Ergebnismenge abgerufen werden müssen.

Zusammenfassend ergeben sich somit folgende Schritte:

1. **Auflösung der semantischen Annotationen:** Die semantischen Erweiterungen müssen aufgelöst und geparst werden.
2. **Ermittlung der RDF-Graph-Templates:** Für jeden semantisch beschriebenen Service wird das zugehörige RDF-Graph-Template ermittelt.
3. **Überführung der SPARQL-Anfrage in einen Ausdruck der SPARQL-Algebra:** Die SPARQL-Anfrage wird entsprechend den Definitionen der SPARQL-Spezifikation in einen Ausdruck der SPARQL-Algebra überführt.
4. **Auflösung des SPARQL-Algebra-Ausdrucks gegen die Graph-Templates:** Der SPARQL-Algebra-Ausdruck wird anhand einer angepassten Evaluierungssemantik gegen die Graph-Templates aufgelöst.
5. **Erzeugung der OData-URIs:** Die im Rahmen der Auflösung der Anfrage ermittelten relevanten Tripel-Templates werden in OData-URIs überführt.

Die Auflösung der semantischen Annotationen und die Ermittlung der RDF-Graph-Templates müssen nur einmalig für jeden registrierten Service durchgeführt werden. Die Schritte 3 bis 5 hingegen sind für jede vom Client gestellte SPARQL-Anfrage erneut auszuführen. Wobei die Schritte 4 und 5 parallel ausgeführt werden.

Der beschriebene Ablauf soll anhand der in Listing 61 gezeigten Beispiel-Anfrage demonstriert werden. Die gezeigte SPARQL-Anfrage wurde mit Hinblick auf das in Anhang C angegebene, semantisch erweiterte Service-Metadaten-Dokument des Northwind-Services formuliert. Sie ruft die Firmennamen (*?companyName*) aller Kunden ab, die in Berlin oder Bern angesiedelt sind. Des Weiteren wird für jede Bestellung, die diese Kunden getätigt haben, das zugehörige Bestelldatum *(?orderDate*) ausgegeben.

```
1. SELECT ?companyName ?orderDate
2. WHERE {
3.  ?cust rdf:type northw:Customer .
4.  { ?cust northw:city "Berlin" } UNION
5.  { ?cust northw:city "Bern" } .
6.  ?cust northw:company_name ?companyName .
7.  ?cust northw:order ?or .
8.  ?or northw:order_date ?orderDate }
```

Listing 61: Eine SPARQL-Anfrage.

Um die RDF-Graph-Templates ermitteln zu können, ist zunächst die Auflösung der semantischen Annotationen erforderlich. Alle semantischen Annotationen, die keine vollständigen, durch Subjekt-, Prädikat- und Objekt-Template definierten, Tripel-Templates beschreiben, müssen, ihrer Position im Datenmodell entsprechend, in vollständige Tripel-Templates überführt werden. Als Beispiel sei die semantische Annotation der *City*-Eigenschaft des *Customer*-Entitätstyps gegeben (siehe Anhang C). Das Tripel-Template ist in Form Prädikat-Templates *northw:city* spezifiziert. Da der *Customer*-Entitätstyp auf die Ressourcen-Variable *?customer* abgebildet wird und die semantische Annotation für die *City*-Eigenschaft definiert ist, wird das Prädikat-Template *northw:city* zu folgendem Tripel-Template aufgelöst: *?customer northw:city $City*. Analog dazu werden für alle in abgekürzter Schreibweise angegebenen Tripel-Templates die fehlenden Templates ermittelt. Die Auflösung der semantischen Annotationen wird in Abschnitt 5.2 im Detail behandelt.

Nachdem alle semantischen Annotationen aufgelöst wurden, werden, wie bereits beschrieben, die Tripel-Templates eines Services zu einem Graph-Template zusammengefasst. Wobei die Informationen darüber, welche Daten von dem Service abgerufen werden müssen, um die, durch das Tripel-Template beschriebenen, Tripel erzeugen zu können, formal gefasst werden. Sei als Beispiel das bereits bekannte Tripel-Template *?customer northw:city $City* gegeben. Um die, durch dieses Tripel-Template beschriebenen, Tripel erzeugen zu können, muss die Eigenschaft *City* abgerufen werden, da die Werte dieser Eigenschaft die Objekte der Tripel bilden. Um die Subjekt-URIs der Tripel erzeugen zu können, wird, entsprechend den Erläuterungen aus Abschnitt 4.3, der Schlüssel der entsprechenden Entitäten benötigt. Folglich muss ebenfalls die *CustomerID*-Eigenschaft Bestandteil der abzufragenden Daten sein. Wie noch in Abschnitt 5.3 beschrieben wird, werden diese Informationen in Form sogenannter *Pfade* formal gefasst. Listing 62 zeigt einen Teil des sich aus dem semantisch erweiterten Service-Metadaten-Dokument in Anhang C ergebenden RDF-Graph-Templates des Northwind-Services.

```
1. ?customer rdf:type northw:Customer .
2. ?customer northw:city $City .
3. ?customer northw:company_name $CompanyName .
4. ?customer northw:order ?order .
5. ?order northw:order_date $OrderDate .
6. ?employee northw:city $City .
7. ?employee northw:order ?order .
```

Listing 62: Ausschnitt aus dem RDF-Graph-Template des Northwind-Services.

Der nächste Schritt besteht darin, die SPARQL-Anfrage anhand des in Abschnitt 2.2.2 angegebenen Algorithmus in einen Ausdruck der SPARQL-Algebra zu überführen. Listing 63 zeigt den sich ergebenden Ausdruck.

```
1. Project(Join(BGP(?cust rdf:type northw:Customer .
2.  ?cust northw:company_name ?companyName .
3.  ?cust northw:order ?or .
4.  ?or northw:order_date ?orderDate),
5.  Union(BGP(?cust northw:city "Berlin"),
6.        BGP(?cust northw:city "Bern"))),
7.  {?companyName,?orderDate})
```

Listing 63: Ein Ausdruck der SPARQL-Algebra.

Dieser Ausdruck der SPARQL-Algebra wird, ausgehend von den einfachen Graph-Mustern (*BGP*), gegen das Graph-Template des registrierten Services aufgelöst. Dabei werden für jedes einfache Graph-Muster alle möglichen Tripel-Templates ermittelt, welche Tripel beschreiben, für die es eine Lösung geben könnte. Sei als Beispiel das in Zeile 5, Listing 63 dargestellte, einfache Graph-Muster *?cust northw:city "Berlin"* gegeben. Eine Lösung für dieses einfache Graph-Muster kann es entsprechend der SPARQL-Spezifikation nur für Tripel geben, die ebenfalls die *northw:city*-URI als Prädikat aufweisen. In dem in Listing 62 dargestellten Graph-Template beschreibt sowohl das in Zeile 2, als auch das in Zeile 6 dargestellte Tripel-Template eine Menge von Tripeln mit diesem Prädikat. Beide Tripel-Templates weisen außerdem mit *$City* ein Literal-Template auf, welches das Objekt des Tripel-Patterns (*"Berlin"*) bereitstellen könnte. Eine Lösung für dieses einfache Graph-Muster würde die *?cust*-Variable auf eine durch die Ressourcen-Variablen *?customer* und *?employee* repräsentierten URIs abbilden.

Zur Repräsentation dieser möglichen Lösungen werden zwei sogenannte Lösungs-Template-Mappings erzeugt. Wobei das erste Lösungs-Template-Mapping die *?cust*-Variable auf die Ressourcen-Variable ?customer abbildet. Analog dazu bildet das zweite Lösungs-Template-Mapping die *?cust*-Variable auf die *?employee*-Ressourcen-Variable ab. Um die durch diese Lösungs-Template-Mappings repräsentierten Lösungen ermitteln zu können, müssen alle Daten abgefragt werden, um die durch die Tripel-Templates beschriebenen Tripel erzeugen zu können. Das Tripel-Template aus Zeile 2 basiert, entsprechend des semantisch

erweiterten Service-Metadaten-Dokument in Anhang C, auf den Entitäten der *Customers*-Entitätsmenge. Folglich müssen die Entitäten dieser Entitätsmenge abgefragt werden. Wobei, wie weiter oben bereits erläutert, zur Erzeugung des Ressourcen-URIs der Schlüssel (*CustomerID*) benötigt wird, der als Teil der *select*-Abfrageoption mit in die URI aufgenommen wird. Das Objekt-Template referenziert die *City*-Eigenschaft, weshalb diese ebenfalls Bestandteil der *select*-Abfrageoption wird. Allerdings kann es Lösungen nur für Tripel geben, welche das Literal *"Berlin"* als Objekt aufweisen. Daher wird die abzufragende Datenmenge über die *filter*-Abfrageoption auf die Menge der Entitäten eingeschränkt, welche für die Eigenschaft *City* den Wert *Berlin* aufweisen. Es ergibt sich folgende URI: *http://services.odata.org/V2/Northwind/Northwind.svc/Customers ?$select=CustomerID,City &$filter=City eq 'Berlin'*. Aus den gleichen Überlegungen heraus ergibt sich für das Tripel-Template aus Zeile 6, Listing 62 die URI *http://services.odata.org/V2/Northwind/Northwind.svc/Employees?$select=EmployeeID,City &$filter=City eq 'Berlin'*.

Die Auflösung des einfachen Graph-Musters aus Zeile 6, Listing 63 führt zu zwei Lösungs-Template-Mappings, die, hinsichtlich der Abbildung mit den zuvor beschriebenen Lösungs-Template-Mappings, übereinstimmen. Hinsichtlich der URIs zum Abruf der relevanten Daten unterscheiden sich diese nur mit Hinblick auf die Filter-Anweisung. Entsprechend des Objekts des Tripel-Patterns werden die Entitäten auf solche eingegrenzt, die für die Eigenschaft *City* den Wert *Bern* aufweisen.

Für die in den Zeilen 1, 2, 3 und 4 aus Listing 63 dargestellten Tripel-Patterns des ersten einfachen Graph-Musters werden den vorherigen Erläuterungen folgend, die dazu relevanten Tripel-Templates aus den Zeilen 1, 3, 4 und 5 aus Listing 62 ermittelt. Darauf aufbauend, wird ein Lösungs-Template-Mapping definiert, das die *?cust*- und *?or*-Variable auf die *?customer*- und *?order*-Ressourcen-Variable abbildet. Die *?companyName*- und *?orderDate*-Variable werden auf die Literal-Templates *$CompanyName* und *$OrderDate* abgebildet. Für die ermittelten Tripel-Templates wird eine OData-URI generiert, die alle relevanten Daten adressiert. Für das in den Zeilen 1 - 3 dargestellte Teil-Muster ist die *Customers*-Entitätsmenge anzufragen. Dabei wird zur Ermittlung möglicher Lösungen nicht der vollständige RDF-Graph benötigt, der aus den Entitäten der Entitätsmenge erzeugt werden kann. Stattdessen sind ausschließlich die Daten erforderlich, aus denen die Tripel generiert werden können, die durch die Tripel-Templates in den Zeilen 1, 3 und 4 (Listing 62) beschrieben werden. Durch einen Vergleich mit dem semantisch erweiterten Service-Metadaten-Dokument in Anhang C ergibt sich, dass die, durch das Template in Zeile 3 beschriebenen Tripel, aus der Eigenschaft *CompanyName* des *Customer*-Entitätstyps erzeugt werden. Die Referenz zur *northw:Order*-Ressource in Zeile 4 wird auf Grundlage der Navigationseigenschaft zwischen dem *Customer*- und dem *Order*-Entitätstyp erstellt. Daher ist es ausreichend, diese Navigationseigenschaft und die *CompanyName*-Eigenschaft abzurufen. Zur Erzeugung der Ressourcen-URIs wird der Schlüssel der *Customer*-Entitäten benötigt. Für das Teil-Muster in den Zeilen 1 bis 3 (Listing 63) ergibt somit unter Verwendung der OData-spezifischen Abfrageoptionen folgende URI: *http://services.odata.org/Northwind/Northwind.svc/Customers?$select=CustomerID,CompanyName,Orders/OrderID&$expand=Orders*. Um die zur Auflösung des in Zeile 4 (Listing 63) dargestellten Tripel-Patterns benötigten Tripel erzeugen zu können, wird die *OrderDate*-

Eigenschaft des *Order*-Entitätstyps benötigt: *http://services.odata.org/Northwind/Northwind.svc/Orders?$select=OrderID,OrderDate*

Nach Auflösung der einfachen Graph-Muster werden die verbleibenden SPARQL-Operatoren von innen her aufgelöst. Dabei bildet der *Union*-Operator den am weitesten innen liegenden Ausdruck. Als Eingabeparameter erhält dieser zwei Mengen von Lösungs-Template-Mappings. Jede Menge umfasst dabei die zwei beschriebenen Lösungs-Template-Mappings der einfachen Graph-Muster. Der *Union*-Operator wird, der SPARQL-Spezifikation entsprechend, durch die Vereinigung der übergebenen Multimengen an Lösungen aufgelöst. Für das Beispiel aus Listing 63 bedeutet dies, dass sowohl jedes Lösungs-Template-Mapping für das einfache Graph-Muster aus Zeile 5, als auch jede Lösungs-Template-Mapping für das einfache Graph-Muster aus Zeile 6 Teil der von dem *Union*-Operator zurückgegebenen Menge ist. Die Menge der abzufragenden Daten, die für die Verarbeitung relevant sein könnten, ist daher ebenfalls durch die Vereinigung der entsprechenden URIs gegeben.

Nach Auflösung des *Union*-Operators bildet die *Join*-Anweisung den innersten verbleibenden Operator. Der *Join*-Operator wird für zwei Multimengen an Lösungen aufgelöst, indem nur die Lösungen übernommen werden, für die in der jeweils anderen Multimenge eine Lösung existiert, die zu der Lösung kompatibel ist. Eine Lösung ist zu einer anderen Lösung kompatibel, wenn sie die Variablen, die auch Teil der Domäne der anderen Lösung sind, auf dieselben Werte wie die andere Lösung abbildet. Für die Lösungs-Template-Mappings bedeutet dies, dass ausschließlich die Lösungs-Template-Mappings für die Abarbeitung der Anfrage relevant sind, welche die Variablen, über die der Join ausgeführt wird, auf Templates abbilden, die potentiell gemeinsame Werte beschreiben. In dem Beispiel aus Listing 63 wird der Join über die *?customer*-Variable ausgeführt. D.h. es werden nur die Lösungs-Template-Mappings in die Ergebnismenge übernommen, die diese Variable auf dieselbe Ressourcen-Variable abbilden.

Der verbleibende *Project*-Operator hat keinen weiteren Einfluss auf die ermittelten Lösungs-Template-Mappings. Zwar könnte die Auflösung des *Project*-Operators, entsprechend der Semantik der SPARQL-Spezifikation, so definiert werden, dass der Definitionsbereich auf die entsprechenden Variablen eingeschränkt würde. Allerdings hätte dies keine Auswirkungen auf die assoziierten OData-URIs.

Die nach Auflösung des SPARQL-Algebra-Ausdrucks ermittelten Lösungs-Template-Mappings beschreiben mögliche Lösungen. Wobei die mit diesen Lösungs-Template-Mappings assoziierten OData-URIs die Daten adressieren, die benötigt werden, um diese Lösungen berechnen zu können. Für das gezeigte Beispiel ergeben sich zusammenfassend die folgenden URIs (es sind jeweils der Ressourcen-Pfad und die Abfrageoptionen dargestellt):

1. *Customers?$select=CustomerID,CompanyName,Orders/OrderID&$expand=Orders*
2. *Orders?$select=OrderID,OrderDate*
3. *Customers?$select=CustomerID,City&$filter=City eq 'Berlin'*
4. *Customers?$select=CustomerID,City&$filter=City eq 'Bern'*

Zur Optimierung der Performance werden diese URIs soweit wie möglich miteinander verschmolzen. Die URIs 3 und 4 adressieren unterschiedliche Entitäten der gleichen Entitätsmenge, wobei jeweils die gleichen Eigenschaften abgefragt werden. Es ist möglich, beide zusammenzufassen indem die *select*-Anweisungen vereinigt und die *filter*-Anweisungen durch ein logisches *Oder* verknüpft werden: *Customers? $select=CustomerID,City& $filter=City eq 'Berlin' or City eq 'Bern'*. Eine Vereinigung mit der ersten URI ist hingegen nicht möglich, ohne die zu übertragende Datenmenge zu erhöhen.

Die Auflösung von SPARQL-Algebra-Ausdrücken und die Erzeugung der OData-URIs werden in den Abschnitten 5.4, 5.5 und 5.6 im Detail erläutert. Die ermittelten OData-URIs sind ausreichend und mit Hinblick auf das Service-Metadaten-Dokument aus Anhang C auch vollständig, um alle für die Abarbeitung der SPARQL-Anfrage aus Listing 61 benötigten Daten abzurufen.

5.2 Auflösung der semantischen Annotationen

Wie in dem vorherigen Abschnitt erläutert, besteht der erste Schritt bei der Registrierung eines Services am System in der Auflösung der semantischen Annotationen. Um die Beschreibung der Abbildung von Entitätsdatenmodellen auf Graph-Templates zu vereinfachen, wurden bei der Herleitung der entsprechenden Konzepte in Kapitel 4 eine Vielzahl abkürzende Schreibweisen definiert. Diese müssen vor der Ermittlung der RDF-Graph-Templates (siehe Abschnitt 5.3) auf Tripel-Templates abgebildet werden. Listing 64 zeigt ein Beispiel.

```
1. <EntityType Name="Customer" sem:Type="northw:Customer">
2.  <Property Name="Phone" Type="Edm.String"
3.   sem:Mapping="northw:phone" sem:Datatype="ownns:phonenr"/>
4.  <Property Name="Address" Type="NorthwindModel.Address"
5.   sem:Mapping="northw:address"/>
6.  <NavigationProperty Name="Orders"
7.   sem:Mapping="northw:order" />
8. </EntityType>
9.
10.<ComplexType Name="Address"
11. sem:Mapping="?address rdf:type northw:Address">
12.</ComplexType>
13.
14.<EntityType Name="Order" sem:Type="?order=northw:Order">
15.</EntityType>
```

Listing 64: Semantische Annotationen in vereinfachter Schreibweise.

Der Entitätstyp *Customer* enthält in Zeile 1 die semantische Annotation *sem:Type="northw:Customer"*. Diese Angabe beschreibt ein Tripel-Template der Form *?customer rdf:type northw:Customer*. Der Name der Ressourcen-Variablen wird vom System festgelegt. Wobei außer der Eindeutigkeit des Namens keine weiteren Vorgaben erforderlich

sind. Mit dem Namen der Ressourcen-Variablen können die verbleibenden semantischen Annotationen des Entitätstyps in den Zeilen 3, 5 und 7 auf Tripel-Templates abgebildet werden. Ein mittels des *sem:Mapping*-Attributs spezifiziertes Teil-Template kann dabei auf mehrere Tripel-Templates abgebildet werden. Die Anzahl der sich ergebenden Tripel-Templates hängt von der Anzahl der für das Element relevanten Ressourcen-Variablen ab. Die Ermittlung dieser Ressourcen-Variablen wird weiter unten erläutert. Für die verbleibenden semantischen Annotationen der Eigenschaften des *Customer*-Entitätstyps ist *?customer* die einzige relevante Ressourcen-Variable. Somit bildet diese das Subjekt-Template der sich ergebenden Tripel-Templates. Das Objekt-Template wird in Abhängigkeit des annotierten Elements bestimmt. Teil-Templates mit fehlendem Objekt-Template, die als Teil der semantischen Erweiterung von Eigenschaften mit einfachem Typ definiert wurden, erhalten, der Intuition entsprechend, als Objekt-Template eine Referenz auf die entsprechende Eigenschaft. Aus der semantischen Erweiterung der *Phone*-Eigenschaft in Zeile 3 ergibt sich somit unter Berücksichtigung des *sem:Datatype*-Attributs die Abbildung auf folgendes Tripel-Template: *?customer northw:phone $Phone^^owns:phone*.

Für Eigenschaften mit komplexen Typen ist die Bestimmung des Objekt-Templates abhängig von der Abbildung der referenzierten komplexen Typen. In Listing 64 wird der komplexe Typ *Address* auf Ressourcen des Typs *northw:Address* abgebildet, welche durch die Ressourcen-Variable *?address* repräsentiert werden. Diese bildet somit das Objekt-Template für das sich aus der semantischen Annotation der *Address*-Eigenschaft (Zeile 5) ergebende Tripel-Template: *?customer northw:address ?address*. Ähnlich verhält es sich für Navigationseigenschaften, für die das Objekt-Template durch die Standard-Ressourcen-Variablen des referenzierten Entitätstyps definiert ist. Für die *Orders*-Navigationseigenschaft ergibt sich somit die Abbildung auf folgendes Tripel-Template: *?customer northw:order ?order*.

Ausgehend von diesem Beispiel soll im Folgenden eine Abbildung für alle in Kapitel 4 definierten abkürzenden Schreibweisen auf vollständige Tripel-Templates, bestehend aus Subjekt-, Prädikat- und Objekt-Template, definiert werden. Per Definition können Tripel-Templates mittels der Attribute *sem:Mapping* und *sem:Type* spezifiziert werden. Diese sind, der in Anhang B formalisierten Erweiterung von CSDL entsprechend, für die Elemente *EntityType*, *ComplexType*, *EntitySet*, *Property* und *NavigationProperty* definiert. Die Abbildung der Werte des *sem:Type*-Attributs auf Tripel-Templates erfolgt unabhängig von dem erweiterten Element. Entsprechend den Erläuterungen in Kapitel 4 kann die Definition einer Ressourcen-Variablen auf zwei Arten erfolgen[26]. Entweder es wird nur der Typ spezifiziert (*sem:Type="northw:Customer"*) oder es wird neben dem Typ auch die Ressourcen-Variable angegeben (*sem:Type="?order=northw:Order"*). Im ersten Fall wird der Name der Ressourcen-Variable automatisch vom System bestimmt. Wobei dieser, wie bereits erläutert, eindeutig innerhalb des Service-Metadaten-Dokuments sein muss. Die Ressourcen-Variable bildet das Subjekt-Template des sich ergebenden Tripel-Templates. Da es sich um

[26] Bei dem in Abschnitt 4.15 beschriebene Fall bei dem nur die Ressourcen-Variable angegeben wird, handelt es sich nicht um eine Variablen-Definition sondern um eine Referenz auf diese. Daher wird sie hier nicht weiter berücksichtigt.

eine Typ-Angabe handelt, ist per Definition das Prädikat-Template immer *rdf:type*. Das Objekt-Template ist durch die explizit als Wert des *sem:Type*-Attributs angegebene Typ-URI gegeben. Somit wird diese Angabe, wie bereits erwähnt, auf folgendes Tripel-Template abgebildet: *?customer rdf:type northw:Customer*. Allgemein ergibt sich folgende Abbildung:

<Typ-URI> $\Longrightarrow$ *<Ressourcen-Var> rdf:type <Typ-URI>*

Wenn die Ressourcen-Variable, wie in dem zweiten Fall, bereits explizit gegeben ist, bildet diese das Subjekt-Template des sich ergebenden Tripel-Templates *(?order rdf:type northw:Order*):

<Ressourcen-Var>=<Typ-URI> $\Longrightarrow$ *<Ressourcen-Var> rdf:type <Typ-URI>*

Bei der Abbildung der mittels des *sem:Mapping*-Attributs gemachten Angaben auf Tripel-Templates müssen mehrere Fälle unterschieden werden. Ein Tripel-Template besteht aus einem Subjekt-, einem Prädikat- und einem Objekt-Template. Unabhängig von den in Kapitel 4 gemachten Angaben sind somit generell folgende Kombinationen als abkürzende Schreibweise bei der semantischen Annotation über das *sem:Mapping*-Attribut vorstellbar:

1. Subjekt-Template
2. Prädikat-Template
3. Objekt-Template
4. Subjekt- und Prädikat-Template
5. Subjekt- und Objekt-Template
6. Prädikat- und Objekt-Template
7. Subjekt-, Prädikat- und Objekt-Template

Der erste Fall kann nur bei der semantischen Erweiterung von Navigationseigenschaften, Eigenschaften komplexen Typs (siehe Abschnitt 4.8) und bei der Annotation mittels des *sem:Type*-Attributs (siehe Abschnitt 4.15) auftreten. Die Angabe eines Subjekt-Templates als Teil der semantischen Erweiterung einer Navigationseigenschaft nimmt eine Sonderstellung ein, da sie keine Abbildung auf ein Tripel-Template beschreibt. Sie dient nur zur Markierung der Navigationseigenschaften, die als Teil einer Verbindung zwischen zwei Entitätstypen bei der Abbildung berücksichtigt werden müssen. Das gleiche gilt für Eigenschaften, die einen komplexen Typ aufweisen. In diesem Fall markiert das Subjekt-Template die Eigenschaft als Teil einer Verbindung zu einem komplexen Typ. Ähnliches gilt für die Angabe über das *sem:Type*-Attribut, die dazu dient, die Beziehung zur Typ-Angabe der entsprechenden Entitätsmenge herzustellen. Somit sind diese Fälle zwar für die Ermittlung der Service-Aufrufe bzw. zur Erzeugung der RDF-Daten relevant, da sie aber keine Tripel-Templates beschreiben, ist auch keine Abbildung auf diese notwendig. Daher werden sie diesbezüglich nicht weiter berücksichtigt.

Die ausschließliche Angabe des Prädikat-Templates (2.) über das *sem:Mapping*-Attribut ist für Eigenschaften und Navigationseigenschaften definiert. Der dritte Fall ist per Definition nur für das *sem:Type*-Attribut (siehe oben) und nicht für das *sem:Mapping*-Attribut möglich. Die Angabe des Subjekt- und Prädikat-Templates ist, wie bei dem zweiten Fall, für Eigenschaften und Navigationseigenschaften definiert. Der fünfte Fall ist den Erläuterungen aus Kapitel 4 entsprechend, wieder nur als Wert des *sem:Type*-Attributs möglich. Die Kombinationen 6 und 7 sind als Teil der semantischen Erweiterungen aller oben erwähnten Elemente möglich. Wobei Fall 7 bereits ein vollständiges Tripel-Template beschreibt. Zusammenfassend müssen somit für die Fälle 2, 4 und 6 Abbildungen definiert werden. Dabei ist für die Fälle 2 und 6 die Bestimmung des Subjekt-Templates erforderlich. Für Fall 2 und für Fall 4 muss das Objekt-Template bestimmt werden. Im Folgenden wird daher die Abbildung, ausgehend von der Bestimmung des Subjekt- bzw. Objekt-Templates, in Abhängigkeit der Entitätsdatenmodell-Konstrukte definiert.

5.2.1 Bestimmung der Subjekt-Templates

Fall 6 ist als Teil der semantischen Erweiterung für folgende EDM-Konstrukte definiert: *EntityType*, *ComplexType*, *EntitySet*, *Property* und *NavigationProperty*. Da Fall 2 für eine Untermenge dieser Elemente definiert ist, muss die Bestimmung der Subjekt-Templates für alle diese Elemente spezifiziert werden. Dies soll, ausgehend von dem *Property*-Element, erfolgen. Ein *Property*-Element kann als Unterelement eines *EntityType*- oder eines *ComplexType*-Elements definiert sein. Der einfachste Fall besteht darin, dass im Rahmen der semantischen Annotation der Eigenschaft selbst Ressourcen-Variablen definiert wurden. Die Menge der relevanten Ressourcen-Variablen besteht dann nur aus diesen Ressourcen-Variablen. Wenn, wie anhand des Beispiels aus Listing 64 demonstriert, für die Eigenschaft keine Ressourcen-Variablen definiert wurden, die Eigenschaft aber Teil eines Entitätstyps ist und der Entitätstyp eine Ressourcen-Variablen Definition enthält, dann besteht die Menge der relevanten Subjekt-Templates aus der als Teil der semantischen Erweiterung des Entitätstyps definierten Ressourcen-Variablen.

Wenn der Entitätstyp Teil einer Typ-Hierarchie ist, kommen gegebenenfalls weitere Ressourcen-Variablen hinzu. Listing 65 zeigt als Beispiel eine aus drei Entitätstypen bestehende Typ-Hierarchie[27]. Der Entitätstyp *Manager* ist vom Entitätstyp *Employee* abgeleitet, welcher wiederum den Untertyp des Entitätstyps *Human* bildet. Die drei Entitätstypen werden der semantischen Annotation entsprechend, auf drei Ressourcen-Templates mit den Typen *northw:Manager*, *northw:Employee* und *northw:Human* abgebildet. Die Abbildung der Eigenschaft *EmployeeID* des *Employee*-Entitätstyps ist durch das Prädikat-Template *northw:employee_id* beschrieben. Den bisherigen Erläuterungen entsprechend, enthält die Menge der relevanten Subjekt-Templates die *?employee*-Ressourcen-Variable. Da der Entitätstyp *Manager* von dem *Employee*-Entitätstyp abgeleitet ist, erbt der Entitätstyp *Manager* alle Eigenschaften des *Employee*-Entitätstyps. Bei der Abbildung der *Manager*-Entitäten auf Ressourcen des Typs *northw:Manager* sollten daher auch die Eigenschaften der

[27] Diese Typ-Hierarchie ist nicht Bestandteil des Entitätsdatenmodells des Northwind-Services.

Basistypen mit in die Abbildung einbezogen werden, insofern diese nicht durch explizite Angabe des Subjekt-Templates auf bestimmte Ressourcen-Templates beschränkt wurden. Für das Beispiel aus Listing 65 ergibt sich somit für die *EmployeeID*-Eigenschaft als weiteres relevantes Subjekt-Template die als Teil der semantischen Erweiterung des *Manager*-Entitätstyps definierte Ressourcen-Variable *?manager*. Ausgehend von diesen Erläuterungen, ergeben sich für die Menge der relevanten Subjekt-Templates für das durch das *northw:last_name*-Prädikat-Template beschriebene Tripel-Template der *LastName*-Eigenschaft die drei Ressourcen-Variablen *?human*, *?employee* und *?manager*. Die Menge der relevanten Subjekt-Templates der *Level*-Eigenschaft besteht hingegen nur aus der im *Manager*-Entitätstyps definierten Ressourcen-Variable *?manager*.

```
1. <EntityType Name="Human" sem:Type="?human=northw:Human">
2.  <Property Name="LastName" Type="Edm.String"
3.   sem:Mapping="northw:last_name" />
4. </EntityType>
5.
6. <EntityType Name="Employee" BaseType="Human"
7.  sem:Type="?employee=northw:Employee">
8.  <Property name="EmployeeID" Type="Edm.Int32"
9.   sem:Mapping="northw:employee_id" />
10.</EntityType>
11.
12.<EntityType Name="Manager" BaseType="Employee"
13. sem:Type="?manager=northw:Manager">
14. <Property name="Level" Type="Edm.Int32"
15.  sem:Mapping="northw:level" />
16.</EntityType>
```

Listing 65: Semantische Erweiterung einer Typ-Hierarchie.

Die in den Basistypen eines Entitätstyps definierten Ressourcen-Variablen haben keinen Einfluss auf die Menge der relevanten Subjekt-Templates der Eigenschaften des Entitätstyps, falls im Rahmen der semantischen Erweiterung des Entitätstyps selbst Ressourcen-Variablen definiert wurden. Wenn der Entitätstyp selbst keine Ressourcen-Variablen definiert, kann es die semantische Beschreibung vereinfachen, wenn die im nächst höheren Entitätstypen definierten Ressourcen-Variablen übernommen werden. Angenommen, es existiert, mit Hinblick auf das Beispiel aus Listing 65, eine Entitätsmenge *Humans*, die den Entitätstyp *Human* referenziert. Neben den Entitäten dieses Typs kann die Entitätsmenge auch Entitäten aller abgeleiteten Typen enthalten. Wenn der Entitätstyp *Employee* selbst keine Ressourcen-Variablen definiert (das *sem:Type*-Attribut in Zeile 7 fehlt), dann sollten die *Employee*-Entitäten ebenfalls auf Ressourcen des Typs *northw:Human* abgebildet werden. Die andere Möglichkeit die Abbildung zu definieren, würde darin bestehen, die *Employee*-Entitäten von der Abbildung auszunehmen. Dies würde dazu führen, dass ausschließlich aus den Entitäten des Entitätstyps *Human* RDF-Ressourcen des Typs *northw:Human* erzeugt werden. Die

Employee-Entitäten wären davon ausgeschlossen, obwohl sie der Definition der Typ-Hierarchie entsprechend auch Instanzen des *Human*-Entitätstypen sind. Dies widerspricht dem erwarteten Verhalten. Daher wird für einen Entitätstyp, der keine Ressourcen-Variablen-Definition enthält, die Abbildung entsprechend des nächst höherem Entitätstypen mit Ressourcen-Variablen-Definition definiert. Abbildungen von Eigenschaften, wie sie z.B. in Zeile 9, Listing 65 angegeben sind, müssen dabei berücksichtigt werden. Daher wird in diesem Fall die im Basistyp definierte Ressourcen-Variable in die Menge der relevanten Subjekt-Templates aufgenommen.

Bei der Bestimmung der relevanten Ressourcen-Variablen für die Eigenschaft eines Entitätstyps müssen auch die Navigationseigenschaften berücksichtigt werden, die den Entitätstyp referenzieren. Wie in Abschnitt 4.8 beschrieben und in Listing 42 gezeigt, können mehrere Entitätstypen durch Markierung der entsprechenden Navigationseigenschaft auf ein einziges Ressourcen-Template abgebildet werden. Dabei ist die Abbildung der Eigenschaften des referenzierten Entitätstyps durch das Prädikat-Template ausreichend beschrieben. Das Subjekt-Template ergibt sich aus der als Teil der semantischen Annotation der Navigationseigenschaft angegebenen Ressourcen-Variablen. Wenn der referenzierte Entitätstyp Teil einer Typ-Hierarchie ist, dann muss die semantische Annotation der entsprechenden Navigationseigenschaften auch bei der Bestimmung der Subjekt-Templates der Eigenschaften der Basis- und Untertypen berücksichtigt werden. Im Entitätsdatenmodell des Northwind-Services wird der Entitätstyp *Employee*, ausgehend von dem *Territory*-Entitätstyp, über die Navigationseigenschaft *Employees* referenziert. Der *Territory*-Entitätstyp soll auf ein Ressourcen-Template des Typs *northw:Territory* abgebildet werden. Wenn die *LastName*-Eigenschaft des *Human*-Entitätstyp mit in diese Abbildung einbezogen werden soll, dann kann dies den Erläuterungen aus Abschnitt 4.8 entsprechend erfolgen. Wobei bei der Bestimmung der Subjekt-Templates der *LastName*-Eigenschaft die Navigationseigenschaften der Untertypen berücksichtigt werden müssen. Somit würde sich für die *LastName*-Eigenschaft, zusätzlich zu den durch die semantischen Erweiterungen der Entitätstypen gegebenen Ressourcen-Variablen *?human*, *?manager* und *?employee*, auch die Ressourcen-Variable *?territory* ergeben (unter der Annahme, dass die Navigationseigenschaft *Employees* mit dieser Ressourcen-Variable annotiert wurde). Ähnliches gilt bei der Bestimmung der Ressourcen-Variablen für die Eigenschaften des *Manager*-Entitätstyps. Dabei müssen auch die Navigationseigenschaften der Basistypen mit einbezogen werden.

Entitätstypen können durch Entitätsmengen referenziert werden. Die semantischen Annotationen der Entitätsmengen müssen bei der Bestimmung der Subjekt-Templates der Eigenschaften der referenzierten Entitätstypen berücksichtigt werden. Als Beispiel sei die Entitätsmenge *Trainee* gegeben, die den in Listing 65 dargestellten Entitätstyp *Employee* referenziert und alle Angestellten umfasst, die sich noch in der Ausbildung befinden. In der semantischen Erweiterung der Entitätsmenge *Trainee* ist eine weitere Ressourcen-Variable *?trainee* definiert. Wie in Abschnitt 5.3 erläutert wird, werden bei der Bestimmung des RDF-Graph-Templates einer Entitätsmenge nur die Tripel-Templates berücksichtigt, deren Subjekt-Templates den Ressourcen-Variablen entsprechen, die als Teil der semantischen Erweiterungen der Entitätsmengen definiert wurden. Dabei bleiben aber die Ressourcen-Variablen, die für die referenzierten Entitätstypen definiert wurden, weiterhin Teil der Menge der für die Eigenschaften der Entitätstypen relevanten Subjekt-Templates. Der Grund dafür

ist, dass ein Entitätstyp direkt oder indirekt (als abgeleiteter Typ) von weiteren Entitätsmengen referenziert sein kann, welche selbst keine eigenen Ressourcen-Variablen definieren. In diesem Fall erfolgt die Abbildung entsprechend den Annotationen des Entitätstyps. Daher werden die in der Entitätsmenge definierten Ressourcen-Variablen zusätzlich in die Menge der relevanten Subjekt-Templates aufgenommen und ersetzen diese nicht. Somit ergibt sich für die *EmployeeID*-Eigenschaft aus Listing 65 aufgrund der Referenzierung durch die Entitätsmene *Trainee* als zusätzliches relevantes Subjekt-Template die Ressourcen-Variable *?trainee*. Da die Entitäten der Entitätsmenge auch die Eigenschaften des Basistyps *Human* aufweisen, ist es, ausgehend von den obigen Erläuterungen, sinnvoll, die Ressourcen-Variable auch in die Mengen der relevanten Subjekt-Templates der Eigenschaften der Basistypen zu übernehmen. Obwohl es bei dem gegebenen Beispiel unwahrscheinlich ist (Auszubildende sind üblicherweise keine Manager), kann eine Entitätsmenge, die einen bestimmten Entitätstyp referenziert, auch Entitäten der entsprechenden Untertypen enthalten. Daher werden die in der Entitätsmenge definierten Ressourcen-Variablen auch für die Eigenschaften der Untertypen übernommen.

Aufbauend auf diesen Erläuterungen wird die Bestimmung der relevanten Subjekt-Templates für Eigenschaften im Folgenden formalisiert. Algorithmus 1 beschreibt die entsprechende Funktion im Pseudocode. Die dargestellte Funktion gibt für die als Eingabeparameter übergebene Eigenschaft $p \in P$ die Menge aller relevanten Subjekt-Templates ST_r zurück. Dabei ist P die Menge aller Eigenschaften, E ist die Menge aller Entitätstypen und C ist die Menge aller komplexen Typen. Für ein $p \in P$ ist $p.t$ der Typ der Eigenschaft und $p.t_d$ der Typ, der die Eigenschaft enthält. Es gilt $p.t_d \in E \cup C$. Die Funktion *rV* gibt für ein gegebenes EDM-Element w alle Ressourcen-Variablen zurück, die als Teil der semantischen Annotation des Elements definiert wurden: $rV(w) \coloneqq \{rv \mid rv \in RV \wedge \exists tt \in w.TT : rv = tt.s \wedge tt.p = rdf{:}type\}$. Es sei noch einmal darauf hingewiesen, dass die Menge $w.TT$, wie oben erläutert, auch die über das *sem:Type*-Attribut spezifizierten Tripel-Templates enthält. Bei Ressourcen-Variablen Definitionen über das *sem:Mapping*-Attribut muss per Definition immer das Subjekt-Template spezifiziert sein. Daher sind für die in der Menge $w.TT$ enthaltenen Typ-Definitionen immer die Ressourcen-Variablen bzw. Subjekt-Templates gegeben, unabhängig davon, wie weit die Tripel-Templates, entsprechend den Erläuterungen in diesem Abschnitt, bereits aufgelöst wurden.

Algorithmus 1: $ST_r = sT\_P(p)$

Eingabe: $p \in P$

Ausgabe: ST_r Menge der relevanten Subjekt-Templates

1. **begin**
2. $ST_r \longleftarrow rV(p)$
3. **if** $ST_r = \emptyset \wedge p.t_d \in E$ **then**
4. $ST_r \longleftarrow sT\_ET(p.t_d)$
5. **if** $ST_r = \emptyset \wedge p.t_d \in C$ **then**
6. $ST_r \longleftarrow sT\_CT(p.t_d)$
7. **return** ST_r
8. **end**

Zur Ermittlung der relevanten Subjekt-Templates wird zunächst geprüft, ob im Rahmen der semantischen Erweiterung Ressourcen-Variablen definiert wurden (Zeile 2). Wenn dies nicht der Fall ist, wird unterschieden, ob die Eigenschaft Teil eines Entitätstyps (Zeile 3) oder eines komplexen Typs (Zeile 5) ist. Wenn es sich um einen Entitätstyp handelt, wird Algorithmus 2 ausgeführt. Ist die Eigenschaft Teil eines komplexen Typs, übernimmt Algorithmus 4 die Ermittlung der Subjekt-Templates. Algorithmus 4 wird weiter unten erläutert.

Um die Ermittlung aller relevanten Subjekt-Templates für einen gegebenen Entitätstyp formalisieren zu können, sind zunächst weitere Definitionen erforderlich. Für einen Entitätstyp $e \in E$ ist $e.c$ der direkte Untertyp und $e.b$ der direkte Basistyp von e. Die Menge aller Untertypen $e.C$ ist definiert als $e.C \coloneqq \{ e_c \mid e_c \in E \wedge \exists e_1, e_2, \dots, e_k \in E : e.c = e_1 \wedge e_1.c = e_2 \wedge \dots \wedge e_k.c = e_c\}$. Analog dazu wird die Menge aller Basistypen $e.B$ definiert als $e.B \coloneqq \{e_b | e_b \in E \wedge \exists e_1, e_2, \dots, e_k \in E : e.b = e_1 \wedge e_1.b = e_2 \wedge \dots \wedge e_k.b = e_b\}$. Die Menge aller Eigenschaften des Entitätstyps ist gegeben durch $e.P$. Sei N die Menge aller Navigationseigenschaften. Für ein $n \in N$ ist $n.f$ der Startpunkt und $n.t$ der Zielpunkt der Navigationseigenschaft. Die Navigationseigenschaften eines Entitätstyps e sind in der Menge $e.N \coloneqq \{n \mid n \in N \wedge n.f = e\}$ zusammengefasst. Die Menge aller Navigationseigenschaft, die auf einen bestimmten Entitätstyp e verweisen, ist definiert durch $e.N_t \coloneqq \{n \mid n \in N \wedge n.t = e\}$. ES kennzeichnet die Menge aller Entitätsmengen. Für eine bestimmte Entitätsmenge $es \in ES$ ist $es.t$ der von der Entitätsmenge referenzierte Entitätstyp. Alle Entitätsmengen, die einen Entitätstyp e referenzieren, sind in der Menge $e.ES \coloneqq \{es \mid es \in ES \wedge es.t = e\}$ zusammengefasst. Basierend auf diesen Definitionen ermittelt Algorithmus 2 alle relevanten Subjekt-Templates für einen gegebenen Entitätstypen e.

Algorithmus 2: $ST_r = sT\_ET(e)$

Eingabe: $e \in E$

Ausgabe: ST_r Menge der relevanten Subjekt-Templates

1. **begin**
2. $ST_r \longleftarrow upRV(e)$
3. $ST_r \longleftarrow ST_r \cup \bigcup_{\forall e_c \in e.C} rV(e_c)$
4. **foreach** $e_h \in e.C \cup e.B \cup \{e\}$
5. $ST_r \longleftarrow ST_r \cup \bigcup_{\forall n \in e_h.N_t} mV(n)$
6. $ST_r \longleftarrow ST_r \cup \bigcup_{\forall es \in e_h.ES} rV(es)$
7. **return** T_{rs}
8. **end**

Zunächst werden in Zeile 2 alle Ressourcen-Variablen ermittelt, die als Teil der semantischen Erweiterung des Entitätstyps selbst oder einer seiner Basistypen relevant sind. Dazu wird Algorithmus 3 ausgeführt. Algorithmus 3 durchläuft, ausgehend von dem gegebenen Entitätstypen, dessen Basistypen, bis ein Typ gefunden wird, dessen semantische Annotation eine oder mehrere Ressourcen-Variablen definiert.

Algorithmus 3: $ST_r = upRV(t)$

Eingabe: $t \in E \cup C$

Ausgabe: ST_r Menge der relevanten Subjekt-Templates

1. **begin**
2. $ST_r \longleftarrow \emptyset$
3. $t_b \longleftarrow t$
4. **while** $(ST_r = \emptyset) \wedge (t_b \in E \cup C)$
5. $ST_r \longleftarrow rV(t_b)$
6. $t_b \longleftarrow t_b.b$
7. **return** ST_r
8. **end**

Nachdem Algorithmus 3 ausgeführt wurde, werden alle Ressourcen-Variablen ermittelt, die in den Untertypen des Entitätstyps definiert wurden (Zeile 3, Algorithmus 2). Anschließend werden alle Entitätstypen durchlaufen, die Teil der Typhierarchie des gegebenen Entitätstyps sind (Zeile 4). Für jeden Entitätstyp werden alle Navigationseigenschaften ermittelt, die auf diesen Entitätstyp verweisen. Wenn die semantischen Annotationen dieser Navigationseigenschaften Tripel-Templates enthalten, die nur aus einem Subjekt-Template bestehen, dann werden diese in die Ergebnismenge übernommen (Zeile 5). Die Funktion mV ist dabei folgendermaßen definiert: $mV(w) \coloneqq \{rv \mid rv \in RV \wedge \exists tt \in w.TT : rv = tt.s \wedge tt.p \notin PT \wedge tt.o \notin OT\}$. Des Weiteren werden für jeden Entitätstyp der Hierarchie die Ressourcen-Variablen-Definitionen aller Entitätsmengen (Zeile 6) bestimmt, die diesen Typ referenzieren.

Die Bestimmung der relevanten Subjekt-Templates für eine gegebene Eigenschaft als Teil eines komplexen Typs (Algorithmus 4) unterscheidet sich teilweise von der Bestimmung für eine Eigenschaft, die Teil eines Entitätstyps ist. Komplexe Typen können, wie Entitätstypen, Teil einer Typhierarchie sein. Ein komplexer Typ kann von einem anderen komplexen Typ abgeleitet sein und selbst Untertypen besitzen. Die Typhierarchie wird daher, wie bereits weiter oben für Entitätstypen erläutert, zur Ermittlung der relevanten Ressourcen-Variablen durchlaufen. Zunächst werden, wie in Algorithmus 3 definiert, ausgehend von dem komplexen Typ, der komplexe Typ selbst und die Basistypen bis zum ersten Typen durchlaufen, der eine Ressourcen-Variablen-Definition enthält (Zeile 2). Diese Ressourcen-Variablen werden in die Menge der relevanten Subjekt-Templates aufgenommen. Anschließend werden alle Ressourcen-Variablen, die in den Untertypen definiert wurden, übernommen (Zeile 3). Dabei markiert, analog zu den Definitionen mit Bezug auf die Entitätstypen, für einen komplexen Typ $c \in C$ der Ausdruck $c.c$ den Untertypen des komplexen Typs. Die Menge aller Untertypen des komplexen Typs ist definiert durch $c.C \coloneqq \{ c_c \mid c_c \in C \wedge \exists c_1, c_2, \dots, c_k \in C : c.c = c_1 \wedge c_1.c = c_2 \wedge \dots \wedge c_k.c = c_c\}$.

Algorithmus 4: $ST_r = sT\_CT(c)$

Eingabe: $c \in C$

Ausgabe: ST_r Menge der relevanten Subjekt-Templates

1. **begin**
2. $ST_r \longleftarrow upRV(c)$
3. $ST_r \longleftarrow ST_r \cup \bigcup_{\forall c_c \in c.C} rV(c_c)$
4. $ST_l \longleftarrow \emptyset$
5. **foreach** $c_h \in c.C \cup c.B \cup \{c\}$
6. **foreach** $p \in c_h.P_d$
7. $V_m \longleftarrow mV(p)$
8. **if** $V_m = \emptyset \wedge ST_r = \emptyset$ **then**
9. $ST_l \longleftarrow ST_l \cup sT\_P(p)$
10. **else**
11. $ST_l \longleftarrow ST_l \cup V_m$
12. $ST_r \longleftarrow ST_r \cup ST_l$
13. **return** ST_r
14. **end**

Ein komplexer Typ kann weder Quelle noch Ziel einer Navigationseigenschaft sein. Er kann aber Typ beliebig vieler Eigenschaften sein. Wie in Abschnitt 4.13 erläutert, werden Eigenschaften mit komplexen Typen hinsichtlich der semantischen Annotation ähnlich behandelt wie Navigationseigenschaften. Die Eigenschaften eines komplexen Typs können mit in die Abbildung eines Entitätstyps einbezogen werden, indem die entsprechende Eigenschaft des Entitätstyps mit der entsprechenden Ressourcen-Variablen annotiert wird. Listing 66 zeigt ein Beispiel.

```
1. <EntityType Name="Customer"
2.  sem:Type="?customer=northw:Customer">
3.  <Property Name="Address" Type="NorthwindModel.Address"
4.   sem:Mapping="?customer"/>
5. </EntityType>
6.
7. <ComplexType Name="Address">
8.  <Property Name="City" Type="Edm.String"
9.   sem:Mapping="northw:city"/>
10.</ComplexType>
```

Listing 66: Einbeziehung eines komplexen Typs in die Abbildung.

Der Entitätstyp *Customer* besitzt die Eigenschaft *Address* mit dem komplexen Typ *Address*. Entitäten des Entitätstyps *Customer* werden auf Ressourcen des Typs *northw:Customer* abgebildet. Die Eigenschaften des referenzierten komplexen Typs *Address* sollen dabei in die Abbildung mit einbezogen werden. Dazu wurde die *Address*-Eigenschaft entsprechend

erweitert (Zeile 4). Dies hat zur Folge, dass die Eigenschaft *City* des komplexen Typs *Address* auf ein Tripel-Template des *northw:Customer*-Ressourcen-Templates abgebildet wird.

Wie an diesem Beispiel zu erkennen ist, müssen bei der Bestimmung der relevanten Subjekt-Templates für die Eigenschaften eines komplexen Typs auch die Eigenschaften, welche den komplexen Typ referenzieren, berücksichtigt werden. Dabei werden, wie bei der Behandlung der Navigationseigenschaften für die Entitätstypen, auch die Eigenschaften mit einbezogen, die auf Ober- oder Untertypen verweisen. Wenn eine Eigenschaft keine Markierungen in Form von einzelnen Ressourcen-Variablen als Teil der semantischen Annotation besitzt, dann werden zur Vereinfachung der Annotation die relevanten Ressourcen-Variablen des Typs, der die Eigenschaft enthält, übernommen. Dies erfolgt allerdings nur dann, wenn für den komplexen Typ keine Ressourcen-Variablen definiert wurden. Algorithmus 4 formalisiert das eben Beschriebene in den Zeilen 5 bis 11. Dabei entspricht $c.B$ der Menge der Basistypen des komplexen Typs c: $c.B \coloneqq \{c_b \mid c_b \in C \wedge \exists c_1, c_2, \ldots, c_k \in C : c.b = c_1 \wedge c_1.b = c_2 \wedge \ldots \wedge c_k.b = c_b\}$. Die Menge $c.P$ enthält die Eigenschaften des komplexen Typs. Alle Eigenschaften, die auf den komplexen Typ c verweisen, sind in der Menge $c.P_d \coloneqq \{p \in P \mid p.t = c\}$ zusammengefasst.

Algorithmus 2 und Algorithmus 4 können auch zur Bestimmung der relevanten Subjekt-Templates bei der Auflösung der semantischen Annotationen von Entitätstypen und komplexen Typen verwendet werden. Somit muss die Ermittlung der relevanten Subjekt-Templates noch für die Elemente *EntitySet* und *NavigationProperty* definiert werden. Für semantische Annotationen einer Entitätsmenge sind ausschließlich die als Teil der semantischen Erweiterung der Entitätsmenge definierten Ressourcen-Variablen relevant. Sei ES die Menge aller Entitätsmengen, dann entspricht für eine Entitätsmenge $es \in ES$ die Menge $rV(es)$ der Menge der relevanten Subjekt-Templates: $ST_r = rV(es)$. Wenn im Rahmen der semantischen Annotation einer Navigationseigenschaft $n \in N$ Ressourcen-Variablen definiert wurden, dann gilt ebenfalls $ST_r = rV(n)$. Ansonsten entsprechen die relevanten Subjekt-Templates der Navigationseigenschaft den relevanten Subjekt-Templates des Entitätstypen, welcher die Quelle der Navigationseigenschaft bildet. Die Berechnung erfolgt mittels Algorithmus 2: $ST_r = sT\_ET(n.f)$. Listing 44 zeigt ein Beispiel. Der Entitätstyp *Order* enthält die Navigationseigenschaft *Shipper*. Die Menge der relevanten Subjekt-Templates dieser Navigationseigenschaft besteht aus der vom System für den Wert des *sem:Type*-Attributs erzeugten Ressourcen-Variablen *?Order*.

5.2.2 Bestimmung der Objekt-Templates

Die Auflösung des Objekt-Templates ist, wie weiter oben erläutert, erforderlich, wenn für das Tripel-Template nur das Prädikat-Template (Fall 2) oder das Subjekt- und Prädikat-Template (Fall 4) angegeben wurden. Beide Fälle können bei der semantischen Annotation von Eigenschaften (*Property*) und Navigationseigenschaften (*NavigationProperty*) auftreten. Bei der semantischen Annotation von Eigenschaften erfolgt die Auflösung in Abhängigkeit des Typs der Eigenschaften. Wenn die Eigenschaft einen einfachen Typ besitzt, entspricht das Objekt-Template, nach den Erläuterungen aus Abschnitt 4.4, einer Referenz auf die Eigenschaft. In Listing 30 ist ein Beispiel dargestellt. Die Eigenschaft *ShipName* wird auf ein Tripel-Template abgebildet, das durch das Prädikat-Template *northw:ship_name* beschrieben

ist. Da es sich bei der *ShipName*-Eigenschaft um eine Eigenschaft mit einfachem Typ handelt, ist das Objekt-Template durch die Eigenschaft-Referenz *$ShipName* gegeben. Nach den Erläuterungen des vorherigen Abschnitts, besteht die Menge der relevanten Subjekt-Templates aus der Ressourcen-Variable *?order* (unter der Annahme, dass vom System eine Variable dieses Namens erzeugt wurde). Es ergibt sich somit nach der Auflösung folgendes Tripel-Template: *?employee northw:last_name $LastName*.

Wenn die Eigenschaft einen komplexen Typ besitzt, ergibt sich das Objekt-Template aus der semantischen Annotation des komplexen Typs. Listing 52 aus Abschnitt 4.13 zeigt ein Beispiel. Die *ShipAddress*-Eigenschaft des Entitätstyps *Order* hat den komplexen Typ *Address*. Dieser wird auf ein Ressourcen-Template des Typs *northw:Address* abgebildet. Für das Ressourcen-Template ermittelt das System entsprechend, den Erläuterungen des vorherigen Abschnitts, eine Ressourcen-Variable, z.B. *?address*. Diese Ressourcen-Variable bildet das Objekt-Template für das Tripel-Template, auf welches die *ShipAddress*-Eigenschaft abgebildet wird. Generell müssen neben der Ressourcen-Variable, auf welche der komplexe Typ abgebildet wird, auch die Ressourcen-Variablen der Untertypen mit in die Menge der relevanten Objekt-Templates aufgenommen werden. Denn werden die Untertypen auf andere Ressourcen-Variablen abgebildet, hängen die für eine konkrete Entität erzeugten Tripel von dem Typ der Instanz ab, die durch die Eigenschaft referenziert wird. Ausgehend von diesen Erläuterungen, ergibt sich der in Algorithmus 5 dargestellte Algorithmus zur Bestimmung der relevanten Objekt-Templates für eine gegebene Eigenschaft.

Algorithmus 5: $OT_r = oT\_P(p)$

Eingabe: $p \in P$
Ausgabe: OT_r Menge der relevanten Objekt-Templates

1. **begin**
2. $t \leftarrow p.t$
3. **if** $t \in EST$ **then**
4. $OT_r \leftarrow \{getRef(p)\}$
5. **if** $t \in C$ **then**
6. $OT_r \leftarrow upRV(t)$
7. $OT_r \leftarrow OT_r \cup \bigcup_{\forall t_c \in t.C} rV(t_c)$
8. **return** OT_r
9. **end**

In Zeile 3 wird zunächst geprüft, ob die Eigenschaft einen einfachen Typ hat. Dabei beinhaltet die Menge EST (**E**DM**S**imple**T**ype) alle durch CSDL definierten einfachen Datentypen. Wenn der Typ in der Menge der einfachen Datentypen enthalten ist, wird eine Referenz auf die Eigenschaft in die Menge der relevanten Objekt-Templates aufgenommen. Die Funktion *getRef* liefert eine Referenz auf die übergebene Eigenschaft zurück. Wenn die Eigenschaft einen komplexen Typ aufweist (Zeile 5), wird die Ressourcen-Variable des Typs oder des in der Hierarchie nächst höheren Typs mit Variablen-Definition der Menge hinzugefügt (Zeile 6). Das gleiche gilt für die Ressourcen-Variablen der Untertypen (Zeile 7).

Eine semantisch erweiterte Navigationseigenschaft, für welche die Menge der relevanten Objekt-Templates bestimmt werden soll, kann diesbezüglich mit einer Eigenschaft, die einen komplexen Typ besitzt, verglichen werden. Listing 44 (Abschnitt 4.9) zeigt ein Beispiel. Der Entitätstyp *Order* ist über die Navigationseigenschaft *Shipper* mit dem Entitätstyp *Shipper* verbunden. Beide Entitätstypen werden auf unterschiedliche Ressourcen-Templates abgebildet. Die Navigationseigenschaft wird auf ein Tripel-Template abgebildet, welches durch das Prädikat-Template *northw:shipper* beschrieben ist. Den Erläuterungen aus Abschnitt 4.9 entsprechend, ergibt sich als Objekt-Template die vom System für das Ressourcen-Template des Typs *northw:Shipper* erzeugte Ressourcen-Variable. Wenn der, durch die Navigationseigenschaft referenzierte, Entitätstyp Teil einer Typhierarchie ist, gilt bzgl. der Einbeziehung der Ressourcen-Variablen der Ober- und Untertypen das gleiche wie bei der Bestimmung der Objekt-Templates für Eigenschaften mit komplexen Typ (Algorithmus 5). Daraus folgend, ergibt sich Algorithmus 6 zur Bestimmung der relevanten Objekt-Templates für eine gegebene Navigationseigenschaft.

Algorithmus 6: $OT_r = oT\_N(n)$

Eingabe: $n \in N$
Ausgabe: OT_r Menge der relevanten Subjekt-Templates
1. **begin**
2. $t \longleftarrow n.t$
3. $OT_r \longleftarrow upRV(t)$
4. $OT_r \longleftarrow OT_r \cup \bigcup_{\forall t_c \in t.C} rV(t_c)$
5. **return** OT_r
6. **end**

5.2.3 Auflösung der Tripel-Templates

Wenn als Teil einer semantischen Annotation nur ein Prädikat-Template gegeben ist (Fall 2), wird dieses auf Basis der ermittelten Subjekt- und Objekt-Templates auf eine Menge von Tripel-Templates abgebildet. Die Menge der Tripel-Templates ergibt sich aus der Menge aller möglichen Kombinationen der Subjekt- und Objekt-Templates. Algorithmus 7 beschreibt dieses Verfahren. Dabei erzeugt die Funktion $createTT$ aus dem übergebenen Subjekt-, Prädikat- und Objekt-Template ein Tripel-Template. Dabei werden insbesondere auch die Datentypen und die Sprachangaben der Literal-Templates entsprechend den Erläuterungen aus Abschnitt 4.4.2 ermittelt.

Wenn sowohl Subjekt- als auch Prädikat-Template gegeben sind (Fall 4), enthält die Menge der an den Algorithmus übergebenen Subjekt-Templates nur das gegebene Subjekt-Template. Analog dazu enthält die Menge der Objekt-Templates nur das gegebene Objekt-Template, wenn dieses zusammen mit dem Prädikat-Template als Teil der semantischen Erweiterung spezifiziert wurde (Fall 6).

Algorithmus 7: $TT = generateTTs(ST, pt, OT)$

Eingabe: ST Menge der Subjekt-Templates, $pt \in PT$, OT Menge der Objekt-Templates
Ausgabe: TT Menge der erzeugten Tripel-Templates

1. **begin**
2. $TT \longleftarrow \emptyset$
3. **foreach** $st \in ST$
4. **foreach** $ot \in OT$
5. $tt \longleftarrow createTT(st, pt, ot)$
6. $TT \longleftarrow TT \cup \{tt\}$
7. **return** TT
8. **end**

Als Spezialfall bedarf noch die in Abschnitt 4.18 beschriebene Abbildung auf geschlossene Listen einer genaueren Erläuterung. Anhand des Beispiels aus Abbildung 29 wurde gezeigt, wie mittels der Turtle-Syntax für geschlossene Listen die Abbildung der Eigenschaften auf Elemente einer geschlossenen Liste realisiert werden kann. Die Abbildung der Adressen-Angaben auf eine geschlossene Liste wurde dabei über folgenden Ausdruck spezifiziert: *?customer northw:address ($City $PostalCode $Country)*. Dieser Ausdruck ist, den Erläuterungen aus Abschnitt 2.1 entsprechend, eine Abkürzung für die in Listing 67 dargestellten Tripel-Templates. Wobei die Namen der leeren Knoten willkürlich gewählt wurden.

```
1. ?customer northw:address _:l1 .
2. _:l1 rdf:first $City .
3. _:l1 rdf:rest _:l2 .
4. _:l2 rdf:first $PostalCode .
5. _:l2 rdf:rest _:l3 .
6. _:l3 rdf:first $Country .
7. _:l3 rdf:rest rdf:nill .
```

Listing 67: Auflösung einer geschlossenen Liste.

Jede mittels der Turtle-Syntax spezifizierte Abbildung auf geschlossene Listen kann somit auf eine Menge von Tripel-Templates abgebildet werden. Die leeren Knoten für die Teillisten werden dabei automatisch generiert. Da die sich ergebenden Tripel-Templates vollständig in dem Sinne sind, dass sie über ein Subjekt, Prädikat und Objekt verfügen, ist eine weitere Auflösung nicht erforderlich.

5.3 Ermittlung der RDF-Graph-Templates

Nachdem alle Abkürzungen, wie sie nach Kapitel 4 bei der semantischen Annotation erlaubt sind, entsprechend des im vorherigen Abschnitt vorgestellten Algorithmus aufgelöst wurden, besteht der nächste Schritt in der Ermittlung der Graph-Templates der Services. Dabei werden die im Rahmen der semantischen Erweiterung des Service-Metadaten-Dokuments eines Services definierten Tripel-Templates zu einem Graph-Template zusammengefasst. Wobei die sich aus dem Aufbau und der Position des Tripel-Templates innerhalb des Entitätsdatenmodells ergebenden Informationen darüber, welche Daten vom Service abgefragt werden müssen, um die durch das Tripel-Template beschriebenen Tripel erzeugen zu können, explizit als Teil des Tripel-Templates formal gefasst werden.

Tripel-Templates beschreiben immer die Erzeugung bestimmter Tripel auf Basis oder ausgehend von bestimmten Entitäten. Jede Entität ist Bestandteil genau einer Entitätsmenge. Daher erfolgt die Bestimmung der Information, welche Daten zur Erzeugung der beschriebenen Tripel von einem Service abgerufen werden müssen, ausgehend von den Entitätsmengen. Jeder am System registrierte OData-Service stellt eine Menge von Entitätsmengen zur Verfügung. Diese werden den semantischen Annotationen der Entitätsmenge und der von diesen referenzierten EDM-Konstrukte, wie z.B. Entitätstypen, komplexe Typen usw., entsprechend auf RDF-Graphen abgebildet. Die Tripel-Templates, welche den RDF-Graphen beschreiben, der potentiell aus den Entitäten einer Entitätsmenge erzeugt werden kann, bilden das Graph-Template der Entitätsmenge. Das Graph-Template eines Services setzt sich aus den Graph-Templates der Entitätsmengen des Services zusammen. Die Ermittlung des Graph-Templates des Services erfolgt daher ausgehend von der Ermittlung der Graph-Templates der Entitätsmengen. Diese wird im Folgenden erläutert.

Das Graph-Template einer Entitätsmenge wird in zwei Schritten bestimmt. Zunächst müssen alle Tripel-Templates ermittelt werden, die Teil des Graph-Templates sind. Anschließend wird für jedes dieser Tripel-Templates bestimmt, welche Daten abgerufen werden müssen, um die beschriebenen Tripel erzeugen zu können.

Die Herleitung soll, ausgehend von dem in Listing 68 dargestellten Beispiel, erfolgen. Um zu verdeutlichen, dass die Auflösung der Abkürzungen eine Voraussetzung für die folgenden Schritte ist, sind die semantischen Annotationen in Listing 68 und den folgenden Beispielen vollständig ausgeschrieben. Somit ist für jede semantische Annotation, die ein Tripel-Template beschreibt, das Subjekt-, Prädikat- und Objekt-Template gegeben. Die in Listing 68 dargestellten semantischen Annotationen beschreiben unter anderem die Abbildung der Entitäten der *Employees*-Entitätsmenge auf Ressourcen des Typs *northw:Employee*. Entsprechend den Erläuterungen aus Abschnitt 4.4 werden die Eigenschaften des Entitätstyps, den semantischen Annotationen folgend, auf Aussagen über die Ressource dieses Typs abgebildet. Dabei werden nur die Tripel-Templates in die Abbildung einbezogen, welche die *?employee*-Ressourcen-Variable als Subjekt-Template enthalten. Das in Zeile 10 dargestellte Tripel-Template ist daher nicht Teil des RDF-Graph-Templates der Entitätsmenge. Der Grund dafür ist, dass die Ressourcen, die durch die *?order*-Variable repräsentiert werden, aus den Entitäten der *Order*-Entitätsmenge erzeugt werden und nicht aus den Entitäten der *Employee*-Entitätsmenge, da die Definition der Ressourcen-Variable in Form der Typ-Zuweisung

(*rdf:type*) als Teil der semantischen Beschreibung des *Order*-Entitätstyps (Zeile 18) gegeben ist.

```
1. <EntitySet Name="Employees"
2.  EntityType="NorthwindModel.Employee"
3.  sem:Mapping="?employee rdf:type northw:Employee .
4.               ?employee northw:order ?order"/>
5. <EntityType Name="Employee">
6.  <Property Name="EmployeeID" Type="Edm.Int32"
7.   sem:Mapping="?employee northw:employee_id $EmployeeID" />
8.  <Property Name="LastName" Type="Edm.String"
9.   sem:Mapping="?employee northw:last_name $LastName .
10.              ?order northw:employee_name $LastName"/>
11. <NavigationProperty Name="Orders"
12.  Relationship="NorthwindModel.FK_Orders_Employees"
13.  ToRole="Orders" FromRole="Employees" />
14.</EntityType>
15.<EntitySet Name="Orders"
16. EntityType="NorthwindModel.Employee" />
17.<EntityType Name="Order"
18. sem:Mapping="?order rdf:type northw:Order">
19. <Property Name="OrderID" Type="Edm.Int32"
20.  sem:Mapping="?order northw:order_id $OrderID .
21.               ?employee northw:order_id $OrderID"
22.</EntityType>
```

Listing 68: Ausschnitt aus einem semantisch erweiterten CSDL-Dokument.

Allgemeiner formuliert, setzt sich das RDF-Graph-Template einer Entitätsmenge aus den Tripel-Templates zusammen, deren Subjekt-Templates Ressourcen-Variablen entsprechen, die als Teil der semantischen Annotation der Entitätsmenge, des von der Entitätsmenge referenzierten Entitätstyps oder als Teil der semantischen Annotation der entsprechenden komplexen Typen definiert wurden. Zur Bestimmung des Graph-Templates einer Entitätsmenge müssen daher zunächst die für die Entitätsmenge *relevanten* Ressourcen-Variablen ermittelt werden. Das sind die Ressourcen-Variablen, welche Ressourcen repräsentieren, die potentiell aus den Entitäten der Entitätsmenge erzeugt werden können.

Für den Fall, dass als Teil der semantischen Annotation der Entitätsmenge Ressourcen-Variablen definiert wurden, sind diese Bestandteil der Menge relevanter Ressourcen-Variablen. Wobei die Ressourcen-Variablen des von der Entitätsmenge referenzierten Entitätstyps und dessen Untertypen nicht in die Menge einbezogen werden. Diese Festlegung folgt aus den Erläuterungen aus Abschnitt 4.15. Wenn eine Entitätsmenge Ressourcen-Variablen definiert, werden alle Entitäten dieser Entitätsmenge unabhängig von ihren Typen und deren semantischen Beschreibungen auf die durch die Ressourcen-Variablen repräsentierten Ressourcen abgebildet. Die einzige Ausnahme bilden Ressourcen, welche auf

Basis von Instanzen komplexer Typen erstellt werden. Die zugehörigen Ressourcen-Variablen werden, wie weiter unten noch genauer erläutert wird, bei der Ermittlung des Graph-Templates immer berücksichtigt, unabhängig davon, ob die Entitätsmenge semantisch annotiert wurde. Sollten die Entitätstypen, wie in Abschnitt 4.15 beschrieben, im Rahmen der semantischen Annotation, auf die Ressourcen-Variablen mehrerer Entitätsmengen verweisen, hat dies ebenfalls keinen Einfluss auf die Menge der relevanten Ressourcen-Variablen einer spezifischen Entitätsmenge. Diese Angaben spezifizieren ausschließlich auf welche der für die Entitätsmenge definierten Ressourcen-Variablen die Entitäten der jeweiligen Typen abgebildet werden. Sie erweitern diese jedoch nicht.

Für die Entitätsmenge *Employees* aus Listing 68 besteht somit die Menge der relevanten Ressourcen-Variablen aus der als Teil der semantischen Annotation der Entitätsmenge definierten Variablen *?employee*. Auf Basis dieser Ressourcen-Variablen wird das Graph-Template der Entitätsmenge ermittelt. Dazu werden zunächst die für Entitätsmenge definierten Tripel-Templates betrachtet. Wobei für jedes Tripel-Template geprüft wird, ob das Subjekt-Template in der Menge der relevanten Ressourcen-Variablen enthalten ist. Ist dies der Fall, wird das Tripel-Template in das Graph-Template der Entitätsmenge aufgenommen. Die Entitätsmenge *Employees* in Listing 68 wurde mit den in den Zeilen 3 und 4 dargestellten Tripel-Templates annotiert. Da die Tripel-Templates als Subjekt-Template jeweils die Ressourcen-Variable *?employee* aufweisen, werden sie Teil des Graph-Templates.

Neben den Tripel-Templates der Entitätsmenge beinhaltet das Graph-Template einer Entitätsmenge das Graph-Template des von der Entitätsmenge referenzierten Entitätstyps. Wenn der Entitätstyp Teil einer Typ-Hierarchie ist, dann müssen die Graph-Templates der Untertypen ebenfalls berücksichtigt werden. Sei die Entitätsmenge *Employees* aus Listing 68 gegeben, welche den Entitätstypen *Employee* aus Listing 65 referenziert. In diesem Fall hat der Entitätstyp *Employee* den Obertypen *Human* und den Untertypen *Manager*. Neben den Entitäten des Entitätstyps *Employee* können somit auch die Entitäten des Entitätstyps *Manager* Teil der Entitätsmenge sein. Sofern, wie es bei der *Employees*-Entitätsmenge der Fall ist, für die Entitätsmenge selbst Ressourcen-Variablen definiert wurden, werden die Graph-Templates der Entitätstypen auf Basis dieser Ressourcen-Variablen ermittelt. D.h. alle Entitäten werden unabhängig von ihrem Typ und der für diesen Typen definierten Ressourcen-Variablen auf die im Rahmen der semantischen Annotation der Entitätsmenge spezifizierten Ressourcen-Variablen abgebildet. Für die semantisch annotierte *Employees*-Entitätsmenge aus Listing 68 würden somit, sowohl die Entitäten des Typs *Employee*, als auch die Entitäten des Typs *Manager*, auf die durch die Variable *?employee* repräsentierten Ressourcen abgebildet werden.

Wenn jedoch, wie z.B. bei der *Orders*-Entitätsmenge, die Entitätsmenge nicht semantisch erweitert wurde oder wenn im Rahmen der semantischen Annotation keine Ressourcen-Variablen definiert wurden, hängt die Menge der relevanten Ressourcen-Variablen von den entsprechenden Entitätstypen ab. Die Ressourcen, die aus den Entitäten eines Entitätstyps erzeugt werden, werden durch die Ressourcen-Variablen beschrieben, die als Teil der semantischen Annotation des Entitätstyps spezifiziert wurden. Wenn im Rahmen der semantischen Erweiterung des Entitätstyps keine Ressourcen-Variablen definiert wurden, dann werden den Erläuterungen aus den Abschnitten 4.14 und 5.2.1 folgend, die Entitäten dieses Typs entsprechend den Ressourcen-Variablen des nächst höheren Entitätstyps mit

Variablen-Definition abgebildet. Für eine nicht semantisch annotierte *Employees*-Entitätsmenge würde folglich die Abbildung entsprechend der Variablen-Definitionen der *Employee*- und *Manager*-Entitätstypen erfolgen. *Employee*-Entitäten würden auf Ressourcen des Typs *northw:Employee* abgebildet. Die *Manager*-Entitäten würden hingegen auf Ressourcen des Typs *northw:Manager* abgebildet. Somit würde die Menge der für die Berechnung des Graph-Templates relevanten Ressourcen-Variablen für den Entitätstypen *Employee* aus der Variable *?employee* bestehen. Die Berechnung des Graph-Templates des *Manager*-Entitätstyps würde hingegen auf Basis der *?manager*-Variable erfolgen. Des Weiteren müssen auch die Ressourcen-Variablen berücksichtigt werden, die für die Eigenschaften und Navigationseigenschaften definiert wurden.

Aufbauend auf diesen Überlegungen, formalisiert Algorithmus 8 die Bestimmung der relevanten Ressourcen-Variablen für Entitätstypen. Wobei der Algorithmus mit Hinblick auf die Ermittlung der Graph-Templates von komplexen Typen auch für diese definiert ist. Zunächst werden in Zeile 2 alle Ressourcen-Variablen hinzugefügt, welche für den Typ spezifiziert sind. Die Ressourcen-Variablen der Eigenschaften und, sofern es sich um einen Entitätstypen handelt, auch der Navigationseigenschaften werden in den Zeilen 3 und 5 in die Menge aufgenommen.

Algorithmus 8: $R = gtRV(t)$

Eingabe: $t \in E \cup C$
Ausgabe: $R \subset UT \cup BT$ Menge der relevanten Ressourcen-Variablen

1. **begin**
2. $R \longleftarrow upRV(t)$
3. $R \longleftarrow R \cup \bigcup_{\forall p \in t.P} rV(p)$
4. **if** $t \in E$ **then**
5. $R \longleftarrow R \cup \bigcup_{\forall n \in t.N} rV(n)$
6. **return** R
7. **end**

Auf Basis der relevanten Ressourcen-Variablen kann das Graph-Template des Entitätstyps ermittelt werden. Dazu werden, wie bei der Entitätsmenge, zunächst alle Tripel-Templates betrachtet, die als Teil der semantischen Annotation des Entitätstyps selbst spezifiziert wurden. Ein Tripel-Template wird Teil des Graph-Templates des Entitätstyps, wenn das Subjekt-Template Bestandteil der Menge der relevanten Ressourcen-Variablen ist. Anderenfalls wird es ignoriert. Das gleiche gilt für alle Tripel-Templates der Eigenschaften und Navigationseigenschaften des Entitätstyps. Dabei werden auch die von den Obertypen geerbten Eigenschaften und Navigationseigenschaften berücksichtigt. Mit Hinblick auf das Beispiel aus Listing 68 werden in das Graph-Template des *Employee*-Entitätstyps die in den Zeilen 7 und 9 dargestellten Tripel-Templates, jedoch nicht, wie weiter oben bereits erläutert, das in Zeile 10 dargestellte Tripel-Template, aufgenommen. Neben den Tripel-Templates, die als Teil der semantischen Annotation des Entitätstypen und dessen Eigenschaften und Navigationseigenschaften in das Graph-Template eingehen, müssen auch Tripel-Templates berücksichtigt werden, die für komplexe Typen spezifiziert wurden, welche von dem

Entitätstypen direkt oder indirekt über eine oder mehrere Eigenschaften referenziert werden. Wobei nur die Tripel-Templates in das Graph-Template eingehen, welche eine Ressourcen-Variable als Subjekt-Template aufweisen, die sich in der Menge der relevanten Ressourcen-Variablen befindet. In Listing 66 wurde ein Beispiel gezeigt. Die *City*-Eigenschaft des komplexen Typs *Address* wird für Entitäten des *Customer*-Entitätstyps auf Aussagen mit dem Prädikat *northw:city* abgebildet, da der komplexe Typ über die *Address*-Eigenschaft mit dem Entitätstypen verbunden ist. Allgemein müssen die Tripel-Templates aller komplexen Typen bei der Ermittlung des Graph-Templates eines Entitätstypen berücksichtigt werden, die von diesem Entitätstypen aus erreicht werden können. Das umfasst die komplexen Typen, welche von dem Entitätstypen über Eigenschaften referenziert werden, inklusive deren Untertypen. Wobei, wie bei den Entitätstypen, auch die Tripel-Templates der von den Obertypen geerbten Eigenschaften berücksichtigt werden. Des Weiteren müssen auch alle komplexen Typen und deren Untertypen berücksichtigt werden, die indirekt über eine oder mehrere andere komplexe Typen referenziert werden.

Entsprechend den Erläuterungen aus den Abschnitten 4.8 bis 4.11, können bei der Abbildung auf ein Ressourcen-Template auch Entitätstypen einbezogen werden, die mit dem Entitätstypen, der die Definition der Ressourcen-Variablen enthält, über eine oder mehrere Navigationseigenschaften miteinander verbunden sind. Zum Beispiel beschreibt Listing 68 eine Abbildung des Entitätstyps *Employee* auf ein Ressourcen-Template, das durch die Ressourcen-Variable *?employee* repräsentiert wird. Für jede *Order*-Entität, die von einer bestimmten *Employee*-Entität referenziert wird, erfolgt eine Abbildung der *OrderID*-Eigenschaft auf eine Aussage der *?employee*-Ressource. Daher sind auch die referenzierten Entitätstypen inklusive ihrer Untertypen, sowie die entsprechenden komplexen Typen, die Teil dieser Entitätstypen sind, bei der Bestimmung des Graph-Templates zu berücksichtigen. Folglich müssen die Tripel-Templates aller Entitätstypen und komplexer Typen, die von dem gegebenen Entitätstyp über Navigationseigenschaften, Kind-Vererbungsbeziehungen und Eigenschaften erreicht werden können, mit einbezogen werden. Wobei die Ermittlung der Tripel-Templates auf Basis der für die Entitätsmenge bestimmten Menge an relevanten Ressourcen-Variablen erfolgt.

Wenn der Entitätstyp der Entitätsmenge oder einer seiner Untertypen eine Eigenschaft mit einem komplexen Typen enthält und dieser komplexe Typ, wie in Abschnitt 4.13 beschrieben, auf ein Ressourcen-Template abgebildet wird, dann ist das Graph-Template des komplexen Typs ebenfalls Bestandteil des Graph-Templates der Entitätsmenge, da die entsprechenden Aussagen aus den Entitäten der Entitätsmenge erzeugt werden. Dabei müssen alle komplexen Typen, die von den Eigenschaften eines Entitätstyps referenziert werden, berücksichtigt werden. Das gilt ebenfalls für komplexe Typen, deren Referenzierung indirekt über die Eigenschaften eines anderen komplexen Typs gegeben ist. Wenn der referenzierte komplexe Typ Teil einer Typ-Hierarchie ist, werden, aus den gleichen Überlegungen heraus wie bei Hierarchien von Entitätstypen, die Graph-Templates der Untertypen ebenfalls Bestandteil des Graph-Templates der Entitätsmenge. Die Bestimmung des Graph-Templates eines komplexen Typs kann auf die gleiche Art und Weise erfolgen, wie für einen Entitätstypen. Wobei, wie bereits weiter oben erläutert, unabhängig davon, ob als Teil der semantischen Erweiterung der Entitätsmenge Ressourcen-Variablen definiert wurden, die

Menge der relevanten Ressourcen-Variablen aus den Ressourcen-Variablen des komplexen Typs besteht.

Abbildung 31 zeigt, anhand eines abstrakten Beispiels, die für die Bestimmung des Graph-Templates einer Entitätsmenge relevanten EDM-Elemente und deren Berücksichtigung bei der Ermittlung der Graph-Templates der entsprechenden Entitätstypen und komplexen Typen.

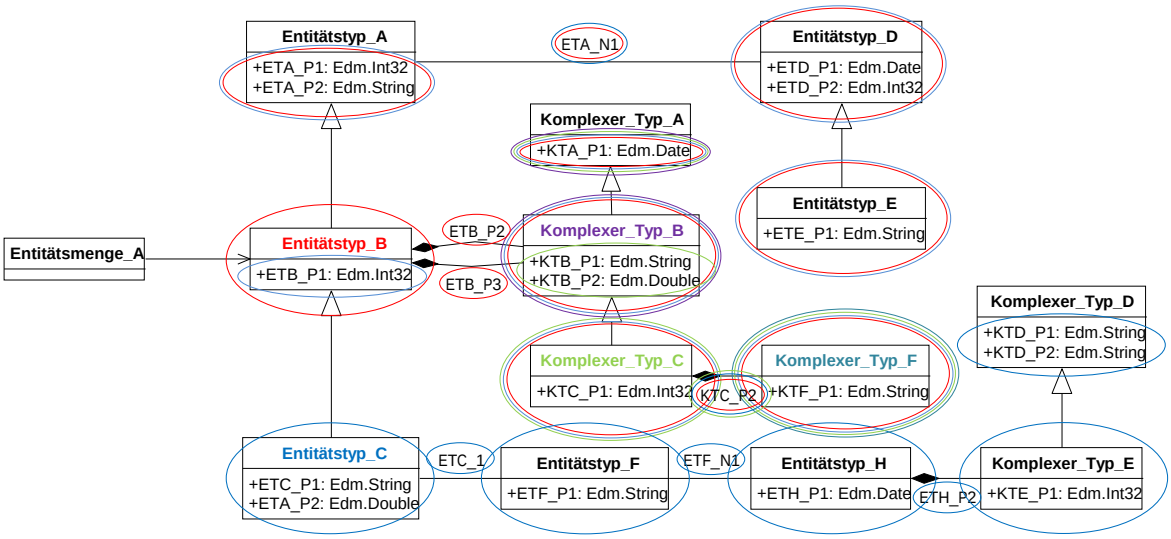


Abbildung 31: Ermittlung der Graph-Templates in Abhängigkeit der relevanten Ressourcen-Variablen.

Es existiert eine Entitätsmenge A. Diese Entitätsmenge wurde selbst nicht semantisch erweitert. Stattdessen referenziert sie den semantisch annotierten Entitätstypen B. Als Teil der semantischen Erweiterung des Entitätstyps B wurden eine oder mehrere Ressourcen-Variablen definiert. Der Entitätstyp B besitzt einen Untertypen C, der ebenfalls ein oder mehrere Ressourcen-Variablen spezifiziert. Da die Entitätsmenge A sowohl Entitäten des Entitätstyps B, als auch des Entitätstyps C enthalten kann, enthält das Graph-Template der Entitätsmenge die Graph-Templates dieser beiden Entitätstypen. Die EDM-Elemente, die bei der Bestimmung des Graph-Templates des Entitätstyps B berücksichtigt werden müssen, sind durch rote Ellipsen eingeschlossen. Das Graph-Templates des Entitätstyps B enthält alle Tripel-Templates, die im Rahmen der semantischen Erweiterung des Typs und seiner Eigenschaften spezifiziert wurden und deren Subjekt-Templates in der Menge der für diesen Entitätstypen definierten Ressourcen-Variablen enthalten ist. Da der Entitätstyp B von dem Entitätstyp A abgeleitet ist, sind auch die Tripel-Templates der geerbten Eigenschaften und Navigationseigenschaften mit entsprechenden Subjekt-Templates Teil des Graph-Templates. Das gleiche gilt, wie weiter oben erläutert, auch für die über die Navigationseigenschaften erreichbaren Entitätstypen. Daher sind auch die Entitätstypen D und E rot eingekreist. Da der komplexe Typ B von dem Entitätstyp B über zwei Eigenschaften referenziert wird, sind auch die Tripel-Templates dieses Typs und seiner Eigenschaften, inklusive der geerbten Eigenschaften, in dem Graph-Template des Entitätstyps enthalten, sofern sie eine der relevanten Ressourcen-Variablen als Subjekt-Template aufweisen. Da der komplexe Typ *C* von dem komplexen Typ *B* abgeleitet ist, kann der Entitätstyp *B* über die beiden Eigenschaften *ETB_P2* und *ETB_P3* auch Instanzen des komplexen Typs *C* referenzieren. Die

Tripel-Templates des komplexen Typs C, dessen Eigenschaften und der von diesem Typ referenzierte komplexen Typ F werden daher ebenfalls berücksichtigt.

Die EDM-Elemente, deren Tripel-Templates bei der Kalkulation des Graph-Templates von Entitätstyp C herangezogen werden müssen, sind durch blaue Ellipsen eingeschlossen.

Den Erläuterungen weiter oben entsprechend, enthält das Graph-Template einer Entitätsmenge auch die Graph-Templates aller komplexer Typen, die von der Entitätsmenge direkt oder indirekt erreichbar sind. In dem Beispiel aus Abbildung 31 referenziert der Entitätstyp *B* über die beiden Eigenschaften *ETB_P2* und *ETB_P3* den komplexen Typen *B*. Daher ist sowohl das Graph-Template dieses Typs, als auch die Graph-Templates seines Untertyps (komplexer Typ *C*), Bestandteil des Graph-Templates der Entitätsmenge. Der komplexe Typ *F* ist indirekt von dem Entitätstypen *B* über die Eigenschaft *KTC_P2* des komplexen Typs *C* erreichbar. Das Graph-Templates dieses Typs ist somit ebenfalls Bestandteil des Graph-Templates der Entitätsmenge. Wie bereits erläutert, erfolgt die Bestimmung des Graph-Templates eines komplexen Typs grundsätzlich wie die Bestimmung des Graph-Templates eines Entitätstyps. Wobei die Menge der relevanten Ressourcen-Variablen aus den Ressourcen-Variablen des entsprechenden komplexen Typs besteht. Unabhängig davon, ob Ressourcen-Variablen für die Entitätsmenge oder den entsprechenden Entitätstypen definiert wurde.

5.3.1 Der Pfad eines Tripel-Templates

Nachdem für eine Entitätsmenge alle Tripel-Templates ermittelt wurden, die Teil des Graph-Templates der Entitätsmenge sind, muss, wie weiter oben bereits beschrieben, für jedes dieser Tripel-Templates bestimmt werden, welche Daten vom Service abgefragt werden müssen, um die beschriebenen Tripel erzeugen zu können. Dazu wird für jedes der ermittelten Tripel-Templates eines Graph-Templates die Menge an EDM-Elementen ermittelt, die vorhanden sein müssen, um ein Tripel auf Basis eines Tripel-Templates zu generieren. Wobei es im Rahmen dieses Prozesses, wie weiter unten noch erläutert wird, zu einer Replizierung von Tripel-Templates innerhalb des Graph-Templates kommen kann.

Sei als Beispiel das Tripel-Template *?employee northw:order ?order* aus Zeile 4, Listing 68 als Teil der semantischen Erweiterung der Entitätsmenge *Employees* gegeben. Den Erläuterungen aus Abschnitt 4.9 folgend, wird diesem Tripel-Template entsprechend jede *Employees*-Entität, die mit einer *Orders*-Entität über eine Navigationseigenschaft verbunden ist, auf eine abstrakte Aussage abgebildet. Die Tripel, die durch dieses Tripel-Template beschrieben werden, können für die Entitäten der *Employees*-Entitätsmenge nur dann erzeugt werden, wenn für jede Entität über die entsprechende Navigationseigenschaft die dazugehörigen *Orders*-Entitäten abgerufen werden. Die Daten, welche als Teil einer Entitätsmenge abgerufen werden müssen, um die durch ein bestimmtes Tripel-Template des Graph-Templates beschriebenen Tripel erzeugen zu können, ist abhängig von dem Aufbau und der Position des Tripel-Templates. Für das Tripel-Template aus Zeile 4, Listing 68 ist die Abfrage der referenzierten *Orders*-Entitäten notwendig, da das Objekt-URI-Template für den Entitätstyp *Order* definiert wurde. Das in Zeile 3, Listing 68 dargestellte Tripel-Template hingegen, basiert ausschließlich auf den Entitäten der Entitätsmenge *Employees*. D.h. außer

den *Employee*-Entitäten sind keine zusätzlichen Daten zur Erstellung der durch das Tripel-Template beschriebenen Tripel erforderlich.

Allgemein wird ein Verfahren benötigt, das für das Tripel-Template eines Graph-Templates die EDM-Elemente ermittelt, die abgerufen werden müssen bzw. Teil der Rückgabemenge sein müssen, um die durch das Tripel-Template beschriebenen Tripel erzeugen zu können. Die Menge an EDM-Elementen nimmt dabei die Form eines Pfades an, da sie die Elemente, ausgehend von der Entitätsmenge des Graph-Templates, spezifiziert. Somit handelt es sich bei einem Pfad um eine geordnete Liste von EDM-Elementen, die ein bestimmtes EDM-Element adressieren. Ein Pfad für ein Entitätsdatenmodell kann foglich mit einem XPath-Ausdruck [50] für ein XML-Dokument verglichen werden. Pfade werden im Folgenden als durch Schrägstriche getrennte *<EDM-Element>:<EDM-Element-Name>* Kombinationen geschrieben. Der Pfad des Tripel-Templates aus Zeile 4, Listing 68 kann damit folgendermaßen angegeben werden: *EntitySet:Employees / EntityType:Employee / NavigationProperty:Orders / EntitySet:Orders/EntityType:Order*.

Im Folgenden werden die verschiedenen Fälle, die bei der Bestimmung des Pfades im Rahmen der semantischen Beschreibung auftreten können, behandelt. Es können die beiden Fälle unterschieden werden, dass ein Tripel-Template als Objekt-Template ein URI-Template bzw. ein Leerer-Knoten-Template aufweist oder nicht. Zunächst wird letzteres betrachtet.

Der einfachste Fall besteht darin, dass, wie in Zeile 3, Listing 68, die Entitätsmenge selbst als Teil der semantischen Erweiterung ein Tripel-Template spezifiziert. Ein solches Tripel-Template wird für jede Entität dieser Entitätsmenge erzeugt. Daher besteht der Pfad dieses Tripel-Templates aus dieser Entitätsmenge. Wenn der von der Entitätsmenge referenzierte Entitätstyp, wie in Zeile 18, Listing 68 dargestellt, ein Tripel-Template ohne Ressourcen-Variable als Objekt-Template enthält, dann wird neben der Entitätsmenge auch der Entitätstyp selbst Teil des Pfades. Dabei ist für die Tripel-Templates der Untertypen der jeweilige Typ Bestandteil des Pfades. Für ein Tripel-Template, das als Teil der semantischen Annotation der Eigenschaft eines Entitätstyps spezifiziert wurde, wird, folgend auf den Entitätstypen, die Eigenschaft an den Pfad angehängt. Das gleiche gilt, wenn das Tripel-Template für eine Navigationseigenschaft definiert wurde.

Der Pfad eines Tripel-Templates, welches Bestandteil des Graph-Templates eines komplexen Typs ist, hängt von der Eigenschaft ab, über den der komplexe Typ von dem Entitätstypen aus erreicht werden kann. Sei als Beispiel die semantische Erweiterung aus Listing 52 gegeben. Der komplexe Typ *Address* wird über die *ShipAddress*-Eigenschaft des Entitätstyps *Order* referenziert. In diesem Fall setzt sich der Pfad aus der Entitätsmenge (*Orders*), dem Entitätstyp (*Order*), der Eigenschaft (*ShipAddress*) und dem komplexen Typ (*Address*) zusammen. Wenn ein komplexer Typ nur indirekt über einen weiteren komplexen Typ erreicht wird, dann werden der entsprechende komplexe Typ und die zugehörige Eigenschaft ebenfalls Teil des Pfades. Des Weiteren besteht die Möglichkeit, dass mehrere Eigenschaften auf den gleichen komplexen Typen verweisen (siehe z.B. *ETB_P2*, *ETB_P3* in dem abstrakten Beispiel aus Abbildung 31). Wie weiter unten noch genauer erläutert wird, werden Tripel-Templates, welche Subjekt-Templates aufweisen, die auf solchen komplexen Typen basieren, mit Bezug auf die Anzahl möglicher Pfade repliziert. Wobei sich die ergebenden Ressourcen-URIs hinsichtlich des Aufbaus unterscheiden. Somit ergibt sich für jeden möglichen Pfad ein Tripel-Template. Die Tripel-Templates des komplexen Typs *B* aus

Abbildung 31 würden bei der Ermittlung des Graph-Templates des Entitätstyps *B* dupliziert, da sie von dem Entitätstyp *B* über zwei Eigenschaften aus referenziert werden. Die Pfade der beiden Tripel-Templates beginnen jeweils mit der Entitätsmenge *A* und dem Entitätstypen *B*. Auf den Entitätstypen folgt die entsprechende Eigenschaft *ETB_P2* oder *ETB_P3*. Der Pfad endet mit dem komplexen Typ *B*. Für Tripel-Templates, die als Teil semantischer Annotationen von Eigenschaften komplexer Typen spezifiziert wurden, ergibt sich aus den gleichen Überlegungen heraus eine Menge replizierter Tripel-Templates. Diese unterscheiden sich hinsichtlich des Pfades zu dem komplexen Typ, enden aber alle auf der gleichen Eigenschaft.

Ein anderer Fall ergibt sich, wenn auf Basis eines komplexen Typs Aussagen über eine Ressource erzeugt werden, die auf einem Entitätstypen oder einem anderen komplexen Typen basieren. Sei als Beispiel der in Listing 69 dargestellte Entitätstyp *Order* gegeben. Dieser Entitätstyp referenziert über die Eigenschaft *ShipAddress* den komplexen Typ *Address*. Für den Entitätstyp *Order* wurde die Ressourcen-Variable *?order* definiert. Die in Zeile 8 dargestellte semantische Annotation der *Street*-Eigenschaft des komplexen Typs *Address* führt dazu, dass das beschriebene Tripel für jede *Order*-Entität aus der über die *ShipAddress*-Eigenschaft referenzierten Instanz des komplexen Typs erzeugt wird. Der Pfad des Tripel-Templates aus Zeile 8 ist in diesem Fall eindeutig definiert, da nur eine Verbindung zwischen dem Entitätstyp und dem komplexen Typ existiert. Er besteht aus der nicht in Listing 69 dargestellten Entitätsmenge *Orders*, dem Entitätstyp *Order*, der Eigenschaft *ShipAddress*, dem komplexen Typ *Address* und der Eigenschaft *Street*.

```
1. <EntityType Name="Order"
2.  sem:Mapping="?order rdf:type northw:Order">
3.  <Property Name="ShipAddress" Type="Address" />
4. </EntityType>
5.
6. <ComplexType Name="Address">
7.  <Property Name="Street" Type="Edm.String"
8.   sem:Mapping="?order northw:street $Street" />
9. </ComplexType>
```

Listing 69: Abbildung eines Entitätstyps und eines komplexen Typs auf ein Ressourcen-Template.

Wie in Abschnitt 4.13 beschrieben, kann es vorkommen, dass mehrere Verbindungen zwischen einem Entitätstypen und einem komplexen Typen existieren. In diesem Fall existieren zwei Möglichkeiten. Wenn eine Verbindung zwischen dem Entitätstypen und dem komplexen Typen explizit markiert wurde, wird die markierte Verbindung in die Abbildung mit einbezogen. Dann folgt der Pfad des Tripel-Templates dem markierten Pfad. Falls keine Verbindung explizit markiert wurde, wird den Erläuterungen aus Abschnitt 4.13 entsprechend, die kürzeste Verbindung zwischen dem Entitätstypen und dem komplexen Typen angenommen. Der Pfad des Tripel-Templates folgt dann dieser kürzesten Verbindung. Für jedes dieser Tripel-Templates muss somit, in Abhängigkeit des Subjekt-Templates, zunächst geprüft werden, ob ein markierter Pfad existiert. Ist dies nicht der Fall, muss die kürzeste

Verbindung von dem komplexen Typ zu dem Entitätstypen bzw. dem komplexem Typen berechnet werden, für den die Ressourcen-Variable definiert wurde. Das gleiche gilt für Tripel-Templates, die, wie in Abschnitt 4.10 beschrieben, als Teil der semantischen Erweiterung eines Entitätstypen spezifiziert wurden, welcher nicht selbst die Definition der Ressourcen-Variable enthält. Zunächst wird geprüft, ob eine markierte Verbindung in Form einer oder mehrerer semantisch annotierter Navigationseigenschaften existiert. Wenn keine Verbindung explizit markiert wurde, wird die kürzeste Verbindung ermittelt. Der sich ergebende Pfad wird Teil des Pfades des Tripel-Templates.

In dem vorherigen Abschnitt wurde die Ermittlung der Pfade für Tripel-Templates hergeleitet, die keine Ressourcen-Variable als Objekt-Template aufweisen. Tripel-Templates mit URI-Template oder Leerer-Knoten-Template als Objekt-Template sollen im Folgenden behandelt werden. Sei das in Zeile 4, Listing 68 dargestellte Tripel-Template gegeben: *?employee northw:order ?order*. Den Erläuterungen aus Abschnitt 4.9 folgend und wie bereits zu Beginn von Abschnitt 5.3.1 beschrieben, wird für jede *Order*-Entität, die mit einer *Employee*-Entität über eine Navigationseigenschaft verbunden ist, ein diesem Tripel-Template entsprechendes Tripel erzeugt. D.h. eine Voraussetzung damit dieses Tripel erzeugt werden kann, ist, dass bei der Abfrage der *Employees*-Entitätsmenge zur jeder *Employee*-Entität alle referenzierten *Order*-Entitäten abgefragt werden. Somit setzt sich der Pfad dieses Tripel-Templates aus der *Employees*-Entitätsmenge, dem Entitätstyp *Employee*, der Navigationseigenschaft *Orders*, der Entitätsmenge *Orders* und dem Entitätstyp *Order* zusammen.

Generell können mehrere Fälle unterschieden werden, wenn eine Entitätsmenge mit einem Tripel-Template semantisch annotiert wurde, welches eine Ressourcen-Variable als Objekt-Template aufweist. Der erste Fall besteht darin, dass die Ressourcen-Variable selbst als Teil der Entitätsmenge definiert wurde. Da zur Erzeugung des Tripels nur die Entität selbst benötigt wird, setzt sich der Pfad nur aus der Entitätsmenge zusammen. Der zweite Fall besteht darin, dass die Ressourcen-Variable für einen komplexen Typ definiert wurde, der über den Entitätstypen der Entitätsmenge direkt oder indirekt referenziert wird. Den Erläuterungen aus Abschnitt 4.13 entsprechend, wird das Tripel auf Basis einer Entität der Entitätsmenge erzeugt, wenn die Entität direkt oder indirekt über eine oder mehrere Eigenschaften eine Instanz des komplexen Typs enthält. Der Pfad setzt sich somit aus der Entitätsmenge, dem Entitätstypen und der entsprechenden Abfolge von Eigenschaften und komplexen Typen zusammen. Falls ein referenzierter komplexer Typ Teil einer Typhierarchie ist und mehr als ein Typ auf die referenzierte Ressourcen-Variable abgebildet wird, dann bildet jeder Typ einen eigenen Pfad und das Tripel-Template wird, der Anzahl möglicher Pfade entsprechend, repliziert. Das gleiche gilt, wenn der komplexe Typ, auf dem die Ressourcen-Variable basiert, über mehrere Typen einer Typhierarchie erreicht werden kann. Wenn mehrere Verbindungen zwischen dem Entitätstypen und dem komplexen Typen existieren, wird bei der Abbildung nur die kürzeste Verbindung berücksichtigt. Wobei, wie bereits erläutert, die Länge einer Verbindung durch die Anzahl der Eigenschaften von dem Entitätstypen zu dem komplexen Typen definiert ist.

Neben den Fällen, dass die dem Objekt-Template entsprechende Ressourcen-Variable für die Entitätsmenge selbst oder für einen komplexen Typ der Entitätsmenge definiert wurde, kann die Ressourcen-Variable, wie in den Abschnitten 4.9 und folgenden beschrieben, für

einen Entitätstypen bzw. eine Entitätsmenge definiert sein, die über eine oder mehrere Navigationseigenschaften erreicht werden kann. Ein Beispiel wurde bereits zu Beginn von Abschnitt 5.3 anhand des Tripel-Templates aus Zeile 4, Listing 68 gegeben. Das dargestellte Tripel-Template enthält als Objekt-Template die Ressourcen-Variable *?order*. Diese basiert auf dem Entitätstyp *Order*, welche von dem Entitätstyp *Employee* über die Navigationseigenschaft *Orders* referenziert wird. Da das Service-Metadaten-Dokument des Northwind-Services nur eine Entitätsmenge beschreibt, die auf den *Order*-Entitätstyp verweist, können die von den *Employee*-Entitäten referenzierten *Order*-Entitäten nur Teil der *Orders*-Entitätsmenge sein. Generell können der Entitätstyp und die Entitätsmengen der über eine Navigationseigenschaft referenzierten Entitäten, der CSDL-Spezifikation entsprechend, über die der Navigationseigenschaft zugehörigen Assoziation bzw. Assoziationsmengen ermittelt werden. In dem dargestellten Beispiel ist die Navigationseigenschaft, welche in die Abbildung einbezogen wird, eindeutig definiert, da nur eine Verbindung zwischen den beiden Entitätstypen existiert. Wenn mehrere Verbindungen zwischen den entsprechenden Entitätstypen existieren, sind, wie in Abschnitt 4.11 beschrieben, die relevanten Navigationseigenschaften durch die kürzeste Verbindung definiert. Wenn das Objekt-Template für einen komplexen Typen definiert wurde, der von einer Entitätsmenge bzw. einem Entitätstypen referenziert wird, welcher sich von der Entitätsmenge bzw. dem Entitätstypen des Subjekt-Templates unterscheidet, setzt sich der Pfad des Tripel-Templates aus dem Pfad zu dem entsprechenden Entitätstypen und dem Pfad von dem Entitätstypen zu dem entsprechenden komplexen Typen zusammen. Die Länge einer Verbindung ist dabei definiert als die Summe aus der Verbindungslänge zwischen den Entitätstypen und der Verbindungslänge von dem Entitätstyp zu dem komplexen Typ.

Wenn abstrakte Aussagen beschreibende Tripel-Templates für Eigenschaften definiert wurden, basieren die Objekt-Templates, den Erläuterungen aus Abschnitt 4.13 folgend, immer auf komplexen Typen. Das Tripel-Template markiert dabei die Eigenschaft, die in die Abbildung mit einbezogen werden soll. Dabei können mehrere Eigenschaften mit dem gleichen Tripel-Template annotiert worden sein. Wobei das Tripel-Template nur einmal in das Graph-Template eingeht. Der Pfad dieses Tripel-Templates setzt sich aus der Gesamtheit der mit diesem Tripel-Template annotierten Eigenschaften und der entsprechenden komplexen Typen zusammen. Bei der Referenzierung von komplexen Typen, welche Teil einer Typ-Hierarchie sind, wird, wie bereits erläutert, für jeden Typ ein extra Pfad erzeugt und die Tripel-Templates werden entsprechend der Anzahl der Pfade repliziert. Das gleiche gilt, wenn Tripel-Templates mit Ressourcen-Variablen als Objekt-Templates für Navigationseigenschaften spezifiziert wurden. Das Tripel-Template beschreibt in diesem Fall die Verbindung zu einer Menge von Kombinationen der referenzierten Entitätsmengen und Entitätstypen. Wobei ebenfalls mehrere Navigationseigenschaften mit dem gleichen Tripel-Template annotiert sein können. Dabei werden diese Tripel-Templates nur in Form eines einzelnen Tripel-Templates Bestandteil des Graph-Templates. Der Pfad dieses Tripel-Templates enthält die Gesamtheit der markierten Navigationseigenschaften. Eine Kombination beider Ansätze ist ebenfalls möglich, wenn das Tripel-Template zunächst als semantische Annotation einer oder mehrerer Navigationseigenschaften die Verbindung zu einem Entitätstypen beschreibt und, ausgehend von dem Entitätstypen, über eine oder mehrere Eigenschaften die Beziehung zu einem komplexen Typen herstellt. Der Pfad des Tripel-

Templates, das in das Graph-Template eingeht, setzt sich aus den markierten Navigationseigenschaften und aus den markierten Eigenschaften zusammen.

Bezüglich der semantischen Annotation von komplexen Typen mittels Tripel-Templates, welche abstrakte Aussagen beschreiben, gilt im Wesentlichen das gleiche, wie bei der Annotation von Entitätstypen. Das Objekt-Template kann entweder einer Ressourcen-Variablen entsprechen, welche auf dem komplexen Typ selbst oder auf einem komplexen Typen bzw. einem Entitätstypen basiert, welche den komplexen Typen referenziert oder von diesem referenziert wird. Wenn die Ressourcen-Variable auf dem komplexen Typ selbst basiert, entspricht der Pfad des Tripel-Templates dem Pfad zu diesem komplexen Typen. Wobei das Tripel-Template, entsprechend der Anzahl möglicher Pfade zu dem komplexen Typen, repliziert wird. Wenn die Ressource-Variable auf einem anderem komplexen Typ basiert, auf den der annotierte komplexe Typ verweist, dann besteht der Pfad des Tripel-Templates aus dem Pfad zu dem annotierten komplexen Typ und der kürzesten Verbindung zwischen den beiden komplexen Typen. Auch in diesem Fall müssen, falls vorhanden, mehrere mögliche Pfade zu dem annotierten komplexen Typ durch Replikation des Tripel-Templates im Graph-Template berücksichtigt werden. Für den Fall, dass das Objekt-Template der Ressourcen-Variable eines komplexen Typs oder eines Entitätstyps entspricht, welches auf den mit dem Tripel-Template annotierten komplexen Typ verweist, dann muss der Pfad des Tripel-Templates dem Pfad des komplexen Typs entsprechen. Der Grund dafür liegt darin, dass die Instanz des komplexen Typs benötigt wird, um die Subjekt-Ressource erzeugen zu können. Der Pfad des Typs, auf dem das Objekt-Template basiert, muss hingegen immer Teil des Pfades des komplexen Typs sein, für den das Tripel-Template spezifiziert wurde.

5.3.2 Formalisierung

Aufbauend auf den Erläuterungen der vorherigen Abschnitte wird im Folgenden die Ermittlung des Graph-Templates einer Entitätsmenge formalisiert. Die Erläuterung erfolgt ausgehend von Algorithmus 9. In Zeile 2 werden die Ressourcen-Variablen der Entitätsmenge bestimmt. Anschließend erfolgt die Aufnahme der Tripel-Templates, welche für die Entitätsmenge selbst spezifiziert wurden, in das Graph-Template. Dabei wird, den obigen Erläuterungen entsprechend (formalisiert in Algorithmus 12), für jedes Tripel-Template die Menge an Pfaden berechnet (Zeile 4). Für jeden möglichen Pfad wird in Zeile 5 ein neues Tripel-Template erzeugt (*clone*-Funktion). Wobei die weiter unten erläuterte *cTT*-Funktion auf Basis des Pfades die Eigenschaften des Tripel-Templates ermittelt. In den Zeilen 6 bis 9 werden die Graph-Templates der relevanten Entitätstypen berechnet. Wie in den vorherigen Abschnitten beschrieben, setzt sich die Menge der relevanten Entitätstypen aus dem von der Entitätsmenge referenzierten Entitätstyp und seiner Untertypen zusammen. Die Ressourcen-Variablen, die bei der Berechnung des Graph-Templates des Entitätstyps berücksichtigt werden, entsprechen den für die Entitätsmenge definierten Ressourcen-Variablen. Wenn keine Ressourcen-Variablen für die Entitätsmenge spezifiziert wurden (Zeile 7), werden die Ressourcen-Variablen des Entitätstyps in die Kalkulation einbezogen (Zeile 8). Die Ermittlung der Graph-Templates der komplexen Typen erfolgt in den Zeilen 10 – 12. Dabei werden, unabhängig von den für die Entitätsmenge oder die Entitätstypen definierten

Ressourcen-Variablen, ausschließlich die Ressourcen-Variablen des jeweiligen komplexen Typs berücksichtigt (Zeile 11).

Algorithmus 9: $GT = gT(es)$

Eingabe: $es \in ES$
Ausgabe: GT Das Graph-Template.

1. **begin**
2. $R \longleftarrow rV(es)$
3. **foreach** $tt \in es.TT$
4. $PA \longleftarrow cP(tt, es)$
5. $GT \longleftarrow GT \cup \bigcup_{\forall pa \in PA} cTT(clone(tt), pa)$
6. **foreach** $e \in es.t.C \cup \{es.t\}$
7. **if** $R = \emptyset$ **then**
8. $R \longleftarrow gtRV(e)$
9. $GT \longleftarrow GT \cup gT(e, R)$
10. **foreach** $c \in getCTs(es)$
11. $R \longleftarrow gtRV(c)$
12. $GT \longleftarrow GT \cup gT(c, R)$
13. **return** GT
14. **end**

Die Ermittlung des Graph-Templates eines Entitätstyps oder eines komplexen Typs ist durch Algorithmus 10 formalisiert. Zunächst müssen alle Tripel-Templates bestimmt werden, die Teil des Graph-Templates sind. Das sind, entsprechend den Erläuterungen aus Kapitel 4, alle Tripel-Templates, welche von dem gegebenen Typen aus erreicht werden können und deren Subjekt-Template in der Menge der relevanten Ressourcen-Variablen enthalten ist. Ein Tripel-Template kann von dem gegebenen Typen aus erreicht werden, wenn es Teil der semantischen Annotation des Typs selbst oder seiner Eigenschaften ist. Des Weiteren ist die Erreichbarkeit auch dann gegeben, wenn das Tripel-Template im Rahmen der semantischen Erweiterung für einen anderen Typen definiert wurde und dieser Typ von dem gegebenen Typ über Eigenschaften und Navigationseigenschaften erreicht werden kann. Die Ermittlung aller EDM-Elemente, die von dem gegebenen Typen erreicht werden können, erfolgt in Zeile 2. Das Entitätsdatenmodell eines OData-Services kann als gerichteter Graph angesehen werden. Wobei die Typen (Entitätstypen, komplexe Typen und primitive Typen) die Knoten bilden und die Navigationseigenschaften und Eigenschaften durch Pfeile repräsentiert werden. Die von einem gegebenen Typen erreichbare Menge an EDM-Elementen kann dann, mittels bekannter Graph-Algorithmen, wie z.B. der Breiten- oder der Tiefensuche [146], bestimmt werden. Die *rI*-Funktion realisiert einen solchen Algorithmus.

Algorithmus 10: $GT = gt(t, R)$

Eingabe: $t \in E \cup C$, $R \subset UT \cup BT$ Menge der relevanten Ressourcen-Variablen.
Ausgabe: GT Das Graph-Template

1. **begin**
2. $M \longleftarrow rI(t)$
3. $TT \longleftarrow rTTs(M, R)$
4. **foreach** $tt \in TT$
5. $PA \longleftarrow cP(tt)$
6. $GT \longleftarrow GT \cup \bigcup_{\forall pa \in PA}\{cTT(clone(tt), pa)\}$
7. **return** GT
8. **end**

Nachdem alle von einem gegebenen Entitätstypen oder komplexen Typen aus erreichbaren EDM-Elemente bestimmt wurden, können alle Tripel-Templates dieser Elemente, welche einen der relevanten Ressourcen-Variablen als Subjekt-Template aufweisen, ermittelt werden (Zeile 3). Algorithmus 11 zeigt die Realisierung der *rTTs*-Funktion. Für jedes EDM-Element (Zeile 2) werden alle Tripel-Templates (Zeile 3) durchlaufen. Wobei für jedes Tripel-Template geprüft wird, ob das Subjekt-Template in der Menge der relevanten Ressourcen-Variablen enthalten ist (Zeile 4). Wenn dies der Fall ist, wird das Tripel-Template in die Rückgabemenge aufgenommen (Zeile 5).

Algorithmus 11: $GT = rTTs(M, R)$

Eingabe: M Menge von EDM-Elementen, R Menge der relevanten Ressourcen-Variablen
Ausgabe: TT Relevante Tripel-Templates

1. **begin**
2. **foreach** $i \in M$
3. **foreach** $tt \in i.TT$
4. **if** $tt.s \in R$ **then**
5. $TT \longleftarrow TT \cup \{tt\}$
6. **return** TT
7. **end**

Nachdem alle Tripel-Templates bestimmt wurden, wird in Zeile 5 (Algorithmus 10) für jedes Tripel-Template, mit der weiter unten erläuterten *cP*-Funktion, die Menge aller Pfade errechnet. Entsprechend der Anzahl der Pfade wird das jeweilige Tripel-Template mittels der *clone*-Funktion repliziert (Zeile 6). Wie noch gezeigt wird, bestimmt die *cTT*-Funktion auf Basis des Pfades die Eigenschaften des Tripel-Templates und weist diese dem replizierten Tripel-Template zu. Anschließend wird dieses Teil des Graph-Templates.

Algorithmus 12 formalisiert die Berechnung der Pfade für ein gegebenes Tripel-Template. In Zeile 2 werden über die *aP*-Funktion alle Pfade, ausgehend von der Entitätsmenge, zu dem Element berechnet, welches die Ressourcen-Variable des Subjekt-Templates definiert (*tt.s.i*). Wenn die Ressourcen-Variable als Teil der semantischen Erweiterung eines Entitätstyps definiert wurde, enthält diese Menge nur einen Pfad. Der sich,

wie weiter oben erläutert, aus der Entitätsmenge und dem Entitätstypen zusammensetzt. Wenn die Ressourcen-Variablen im Rahmen der semantischen Annotation eines komplexen Typs definiert wurden, werden, wie bereits beschrieben, alle Pfade von der Entitätsmenge zu dem entsprechenden komplexen Typ ermittelt. Dazu können zur Realisierung der *aP*-Funktion existierende Graph-Algorithmen verwendet werden. Falls das Tripel-Template für eine Entitätsmenge spezifiziert wurde, enthält die Pfadangabe auch den entsprechenden Entitätstypen. Wenn die Entitätsmenge einen Entitätstyp referenziert, der Teil einer Typ-Hierarchie ist, dann existiert für jeden Entitätstypen ein extra Pfad. Die Angabe des Entitätstypen als Teil der Pfadangabe ist notwendig, um, wie weiter unten noch gezeigt wird, den Schlüsselpfad bestimmen zu können.

Algorithmus 12: $PA = cP(tt)$

Eingabe: $tt \in TT$
Ausgabe: PA Menge der Pfade
1. **begin**
2. $PA_{si} \longleftarrow aP(tt.s.i)$
3. $PA_i \longleftarrow sP(tt.s.i, tt.i)$
4. **if** $tt.o \in UT \cup BT$ **then**
5. $PA_{oi} \longleftarrow sP(tt.i, tt.o.i)$
6. $PA \longleftarrow \forall_{pa_{si} \in PA_{si}} \forall_{pa_i \in PA_i} \forall_{pa_{oi} \in PA_{oi}} pa_{si} \oplus pa_i \oplus pa_{oi}$
7. **return** PA
8. **end**

Im nächsten Schritt werden mittels der sP-Funktion die kürzesten Pfade von dem Element, welches die Ressourcen-Variable definiert, zu dem Element, das über seine semantische Erweiterung das Tripel-Template spezifiziert, berechnet (Zeile 3). Wie bereits erläutert, kann es mehrere kürzeste Pfade geben, wenn die Verbindung Typen enthält, die Teil einer Typ-Hierarchie sind. Auf eine detaillierte Beschreibung der *sP*-Funktion wird hier verzichtet, da diese ebenfalls mittels bekannter Graph-Algorithmen realisiert werden kann. Das Entitätsdatenmodell bildet dabei einen gewichteten Graphen. Wobei durch Ressourcen-Variablen markierte Verbindungen, wie sie in den Abschnitten 4.10 und 4.13 beschrieben sind, über unterschiedliche Gewichtungen der Kanten abgebildet werden können. Wenn das Tripel-Template eine Ressourcen-Variable als Objekt-Template aufweist (Zeile 4), dann werden in Zeile 5 alle kürzesten Pfade von dem Element, welches das Tripel-Template spezifiziert, bis zu dem Element, das die Ressourcen-Variablen definiert (*tt.o.i*), berechnet. Ein Pfad eines Tripel-Templates setzt sich aus den in den vorherigen Schritten ermittelten, aneinandergereihten Teilpfaden zusammen. Die Menge aller Pfade eines Tripel-Templates besteht somit aus allen möglichen Pfad-Kombinationen (Zeile 6).

Entsprechend der Anzahl der Pfade, wird das Tripel-Template repliziert (Zeile 6, Algorithmus 10). Wobei die *cTT*-Funktion anhand des Pfades die Informationen ermittelt, die von der Query Engine und dem RDF Adapter benötigt werden, um die relevanten Daten abzurufen bzw. um die Tripel aus den abgerufenen Daten erzeugen zu können. Algorithmus 13 zeigt diese Funktion.

Algorithmus 13: $tt = cTT(tt, pa)$

Eingabe: $tt \in TT$, pa Pfad des Tripel-Templates
Ausgabe: tt

```
1.  begin
2.    if tt.s ∈ UT then
3.      tt.s.eli ⟵ eli_s(pa)
4.    tt.s.cp ⟵ cp_s(pa, tt.s)
5.    tt.s.KP ⟵ kp_s(pa)
6.    if tt.o ∈ UT then
7.      tt.o.es ⟵ eli_o(pa)
8.    if tt.o ∈ UT ∪ BT then
9.      tt.o.cp ⟵ cp_o(pa)
10.   tt.o.KP ⟵ kp_o(pa)
11.   if tt.o ∈ LT then
12.   tt.o.pp ⟵ cPP(pa, tt.o.p)
13.   if tt.o ∈ FT then
14.     foreach p ∈ tt.o.PR
15.       p.pp ⟵ cPP(pa, p)
16.   tt.pa ⟵ pa
17.   return tt
18. end
```

Für URI-Templates, die das Subjekt-Template eines Tripel-Templates bilden (Zeile 2), müssen die Entity-Location-ID (Zeile 3) und die Pfade zu den Schlüsseln des Entitätstyps (Zeile 5), auf welchem das URI-Template basiert, ermittelt werden. Wenn das URI-Template auf einem komplexen Typ basiert, ist die Bestimmung des Pfades von dem Entitätstypen zu dem entsprechenden komplexen Typen erforderlich (Zeile 5). Dieser Pfad muss ebenfalls für Leere-Knoten-Templates berechnet werden, die auf komplexen Typen basieren. Das gleiche gilt für die Pfade zu den Schlüsseln des entsprechenden Entitätstyps. Beide Angaben werden als Hinweis an die Query Engine benötigt, die zur Erzeugung der leeren Knoten benötigten Daten abzurufen. Des Weiteren sind sie für den RDF Adapter zur Identifikation der entsprechenden Elemente erforderlich. Die Entity-Location-ID hingegen, wird für URI-Templates, den Erläuterungen aus Abschnitt 4.3 folgend, zur Erzeugung der Ressourcen-URIs benötigt. Zur Generierung von IDs für leere Knoten ist sie jedoch nicht notwendig. Sowohl die Entity-Location-ID, als auch die Pfade zu den Schlüsseln und dem komplexen Typ können aus dem Pfad des Tripel-Templates ermittelt werden. Die Entity-Location-ID wird aus der Service-Basis-URI, dem Entitätscontainer und der entsprechenden Entitätsmenge berechnet (siehe Abschnitt 4.3). Die Service-Basis-URI entspricht der Basis-URI des Services, für dessen Entitätsmenge das Graph-Template aktuell berechnet wird. Der Entitätscontainer ist ebenfalls durch die Entitätsmenge eindeutig geben. Die Entitätsmenge entspricht dem ersten Element des Pfades des Tripel-Templates. Die *eli_s*-Funktion ermittelt somit, anhand des ersten Pfad-Elements, die Entitätsmenge, den zugehörigen Entitätscontainer und die Service-

Basis-URI. Mittels einer Hashfunktion wird die Entity-Location-ID berechnet. Der Pfad von der Entitätsmenge zu dem komplexen Typ kann ebenfalls aus dem Pfad des Tripel-Templates berechnet werden. Dazu läuft die *cp_s*-Funktion, ausgehend von der Entitätsmenge und dem zugehörigen Entitätstyp, alle Pfad-Elemente ab, bis der komplexe Typ erreicht wird, welcher die übergebene Ressourcen-Variable definiert. Die Pfade der Schlüssel werden in der *kp_s*-Funktion auf Basis des Entitätstyps berechnet, welcher den Erläuterungen aus Abschnitt 5.3.1 folgend, immer das zweite Element des Tripel-Template-Pfades bildet.

Bezüglich der Objekt-Templates gilt für URI-Templates und Leere-Knoten-Templates das gleiche wie für Subjekt-Templates. Die Kalkulation der Entity-Location-ID (Zeile 7) erfolgt auf Basis der Entitätsmenge, für welche die Ressourcen-Variable definiert ist. Wie bereits erläutert, bildet die Entitätsmenge, für die das Graph-Template berechnet wird, immer das erste Element eines Tripel-Template-Pfades. Neben dieser Entitätsmenge kann der Pfad, in Abhängigkeit der Anzahl der in die Abbildung einbezogenen Navigationseigenschaften, beliebig viele weitere Entitätsmengen beinhalten. Für ein Objekt-Template, das einer Ressourcen-Variable entspricht, folgt jedoch aus den Überlegungen aus Abschnitt 5.3.1, dass die Entitätsmenge, auf der diese Ressourcen-Variable basiert, immer die letzte Entitätsmenge innerhalb des Pfades sein muss. Somit unterscheidet sich die *eli_o*-Funktion zu der *eli_s*-Funktion nur in der Art, wie die Entitätsmenge bestimmt wird. Ähnliches gilt für die *kp_o*-Funktion. Anstatt des ersten Entitätstyps wird bei der Kalkulation der Schlüsselpfade der letzte Entitätstyp in dem Pfad berücksichtigt. Bei der Bestimmung des Pfades mittels der *cp_o*-Funktion zu dem komplexen Typen der Ressourcen-Variable ist der relevante komplexe Typ durch den letzten komplexen Typen innerhalb des Pfades gegeben.

Wenn es sich bei dem Objekt-Template um ein Literal- (Zeile 11) oder ein Funktions-Template (Zeile 13) handelt, müssen die Pfade zu den Eigenschaften dieser Templates bestimmt werden. Den Erläuterungen aus Kapitel 4 entsprechend, kann ein Tripel-Template, das ein Literal-Template als Objekt-Template enthält, nur im Rahmen der semantischen Erweiterung einer Eigenschaft oder des Typs, der die Eigenschaft enthält, spezifiziert sein. Im ersten Fall entspricht der durch die *cPP*-Funktion berechnete Pfad des Literal-Templates dem Pfad des Tripel-Templates. Im zweiten Fall wird der Pfad des Tripel-Templates um die entsprechende Eigenschaft erweitert. Für ein Funktions-Template müssen die Pfade aller Argumente ermittelt werden, die eine Eigenschaft referenzieren. Wenn, wie in dem Beispiel aus Listing 39 aus Abschnitt 4.6 gezeigt, das Funktions-Template mehr als eine Eigenschaft referenziert, kann das Funktions-Template nur als Teil der semantischen Annotation des entsprechenden Typs spezifiziert sein. Der Pfad eines Arguments setzt sich daher wieder aus dem Pfad des Tripel-Templates plus der Eigenschaft zusammen. Für ein Funktions-Template, das für eine Eigenschaft spezifiziert wurde, entspricht der Pfad des entsprechenden Arguments dem Pfad des Tripel-Templates.

5.4 Bestimmung der relevanten Tripel-Templates

Nachdem die Graph-Templates der Services ermittelt wurden, müssen, wie zu Beginn von Kapitel 5 erläutert, im nächsten Schritt für eine gegebene SPARQL-Anfrage, die Tripel-Templates der Services ermittelt werden, die für die Beantwortung der SPARQL-Anfrage

relevant sind. Der Graph, der aus den von den Services bereitgestellten Daten erzeugt wird, muss alle Tripel enthalten, die benötigt werden, um alle möglichen Lösungen bestimmen zu können. Die einfachste Möglichkeit, die Vollständigkeit sicher zu stellen, würde darin bestehen, alle Daten der verfügbaren Services abzurufen. Für die meisten Anwendungsfälle ist dieses Vorgehen jedoch nicht akzeptabel, da es, aufgrund der Größe der zu übertragenden Datenmenge und der Anzahl der zum Abruf aller Daten benötigten Service-Aufrufe, sehr ineffizent wäre. Um eine effizientere Verarbeitung zur ermöglichen, sollten nur die Daten abgerufen werden, aus denen Tripel erzeugt werden, die zur Bestimmung der Lösungen benötigt werden. Das Ziel besteht darin, die minimale Menge an Daten zu bestimmen, die abgerufen werden müssen, um eine vollständige Ergebnismenge berechnen zu können.

Die Menge der relevanten Tripel-Templates wird, ausgehend von den durch die Tripel-Templates beschriebenen Tripel, ermittelt, da die zur Beantwortung einer Anfrage relevanten Tripel auf Basis der durch die SPARQL-Spezifikation definierten Evaluierungssemantik ermittelt werden können. Entsprechend den Erläuterungen aus Abschnitt 2.2.3, wird ein Ausdruck der SPARQL-Algebra, ausgehend von den einfachen Graph-Mustern (*BGP*), aufgelöst. Wobei, wie bereits erläutert, der *BGP*-Operator der einzige Operator der SPARQL-Algebra ist, zu dessen Auswertung der RDF-Datensatz benötigt wird. Eine Lösung als Teil der Rückgabemenge des BGP-Operators für den Graphen G und das Graph-Muster GP ist ein Lösungs-Mapping μ, für das es ein RDF-Instanz-Mapping σ gibt, so dass $\mu(\sigma(GP))$ ein Teilgraph von G ist [155]. Für ein gegebenes, einfaches Graph-Muster muss ermittelt werden, welche Tripel Teil des Graphen sein müssen, um das vollständige Ergebnis berechnen zu können. Das Ergebnis der Funktion $\mu(\sigma(GP))$ ist genau dann ein Teilgraph von G_r, wenn für jedes Tripel-Pattern $tp \in GP$ gilt $\mu(\sigma(tp)) \in G_r$. Um ein vollständiges Ergebnis zu erhalten, muss somit jedes Tripel t Element von G_r sein, für das ein Tripel-Pattern $tp_t \in GP$ existiert, für das es ein Instanz-Mapping σ und eine Lösung μ gibt, so dass gilt $\mu(\sigma(tp)) = t$. Sei als Beispiel das in Listing 70 dargestellte einfache Graph-Muster und der in Listing 71 gezeigte RDF-Graph gegeben.

```
1. ?order rdf:type northw:Order .
2. ?order northw:ship_address _:addr .
3. _:addr northw:city ?city .
```

Listing 70: Ein einfaches Graph-Muster mit leeren Knoten.

Die einzige Lösung μ_L des einfachen Graph-Musters für den gegebenen RDF-Graphen ist definiert durch $\mu_L(?order) = i:order1$ und $\mu_L(?city) = "Dresden"$ für das Instanz-Mapping $\sigma_L(\_:addr) = i:addr1$. Um diese Lösung ermitteln zu können, werden die in den Zeilen 1, 3 und 4 (Listing 71) dargestellten Tripel benötigt. Das in Zeile 2 dargestellte Tripel t_2 ist hingegen nicht erforderlich. Da kein Tripel-Pattern aus dem einfachen Graph-Muster existiert, das nach der Ersetzung der leeren Knoten und der Variablen diesem Tripel entsprechen würde. Wenn GP_e das einfache Graph-Muster ist, gilt somit für alle $tp \in GP_e$ und für alle möglichen μ und σ immer $\mu(\sigma(tp)) \neq t_2$.

```
1. i:order1 rdf:type northw:Order .
2. i:order1 northw:order_date "25-07-2013" .
3. i:order1 northw:ship_address i:addr1 .
4. i:addr1 northw:city "Dresden" .
```

Listing 71: Ein RDF-Graph.

Ein Tripel-Template beschreibt eine Menge von Tripeln $T_{tt} = triples(tt)$. Somit ist für ein gegebenes Tripel-Pattern tp zu bestimmen, ob es ein Instanz-Mapping σ und eine Lösung μ geben kann, so dass gilt: $\mu(\sigma(tp)) \in T_{tt}$. Folglich muss ermittelt werden, ob die leeren Knoten und die Variablen eines Tripel-Patterns so aufgelöst werden können, dass sich das ergebende Tripel in der Menge der durch das Tripel-Template beschriebenen Tripel befindet. Dieses Problem wird im folgenden Abschnitt behandelt.

5.4.1 Vergleich eines Tripel-Patterns mit einem Tripel-Template

Ein Tripel-Pattern pt ist der SPARQL-Spezifikation folgend ein Element der Menge $(I \cup L \cup B \cup V) \times (I \cup V) \times (I \cup L \cup B \cup V)$. Wobei I die Menge aller IRIs [74] und V die Menge aller Variablen ist. Entsprechend den Erläuterungen des vorherigen Kapitels und der Formalisierung in Abschnitt 4.19 ist ein Tripel-Template tt ein Element der Menge $(UT \cup BT) \times U \times (UT \cup BT \cup FT \cup LT \cup U \cup L)$. Um zu ermitteln, ob ein bestimmtes Tripel-Pattern pt auf ein Element der Menge T_{tt} abgebildet werden kann, ist ein Vergleich des Subjekts, des Prädikats und des Objekts des Tripel-Patterns mit dem Subjekt-, dem Prädikat- und dem Objekt-Template des Tripel-Templates erforderlich. Das Subjekt des Tripel-Patterns kann der eben erwähnten Definition entsprechend, eine IRI (I), ein Literal (L), ein leerer Knoten (B) oder eine Variable (V) sein. Im Folgenden wird für jedes dieser Fälle geprüft, ob die Menge der durch ein gegebenes Tripel-Template beschriebenen Tripel, ein Tripel mit einem Subjekt enthalten kann, auf welches das Subjekt des Tripel-Patterns abgebildet werden kann. Das gleiche erfolgt für alle möglichen Kombinationen von Pattern- und Template-Prädikaten und Pattern- und Template-Objekten.

Subjekt-IRIs

Sei das Subjekt des Tripel-Patterns eine IRI $pt.s \in I$. Es muss nun ermittelt werden, ob die Menge T_{tt} ein Tripel mit der gegebenen IRI enthalten kann. Das Subjekt-Templates eines Tripel-Templates kann den obigen Erläuterungen folgend ein URI-Template oder ein Leerer-Knoten-Template sein. Wenn es sich bei dem Subjekt-Template um ein Leerer-Knoten-Template handelt, enthalten alle Tripel, die von dem Tripel-Template beschrieben werden, als Subjekt einen leeren Knoten. Daher kann kein Tripel mit der IRI des Tripel-Patterns als Subjekt existieren. Falls das Subjekt-Template des Tripel-Templates einem URI-Template entspricht, enthalten alle Tripel der Menge T_{tt} URIs als Subjekt. Ob es ein Tripel mit der gegebenen IRI als Subjekt geben kann, hängt vom Aufbau der URIs ab. Wie in Abschnitt 4.3 beschrieben, haben die URIs der Ressourcen, die aus den Entitäten des Services erstellt werden, folgenden Aufbau: *Ressource-URI* = "http://", *Host*, [*Path-Prefix*], "/", *Entity-Location-ID*, "/", *Path-Suffix* ;. Wobei sich für Ressourcen, die auf Basis von Instanzen

komplexer Typen erzeugt wurden, die URI noch um die Namen der Eigenschaften, über die der komplexe Typ, ausgehend von dem entsprechenden Entitätstypen, referenziert wird, erweitert (siehe Abschnitt 4.13): ("/", *Property-Name*)+. Die *Host-* und *Path-Prefix*-Angabe wird über das *sem:URI*-Attribut im Rahmen der Erweiterung des CSDL-Dokuments spezifiziert. Die *Entity-Location-ID* wird aus der Basis-URI, dem Namen des Entitätscontainers und dem Namen der Entitätsmenge gebildet. Das *Path-Suffix* enthält den Namen der Ressourcen-Variable und den Entitätsschlüssel. Wobei das Format des *Path-Suffix* von der über das *sem:URI_struc*-Attribut gemachten Angabe abhängt. Der Name der Ressourcen-Variable wird entweder über die semantische Annotation spezifiziert oder es wird automatisch vom System auf Basis des Namens des entsprechenden EDM-Elements generiert. Die Namen der Eigenschaften sind durch die Verbindung zwischen dem Entitätstypen und dem komplexen Typen definiert. Mit Ausnahme des Entitätsschlüssels sind somit alle Daten, die zur Erzeugung der Ressourcen-URIs benötigt werden, durch das semantisch erweiterte CSDL-Dokument gegeben. Ohne dass die Abfrage von Daten notwendig ist, können daher die *Host-*/*Path-Prefix*-Angabe, die *Entity-Location-ID*, der Name der Ressourcen-Variable, das Format der Subjekt-IRI und falls gegeben, die Namen der Eigenschaften des Tripel-Patterns mit den entsprechenden Werten und dem Format des URI-Templates verglichen werden. Wenn diese Angaben oder das Format nicht übereinstimmen, kann die Menge der durch das Tripel-Template beschriebenen Tripel niemals ein Tripel mit der IRI des Tripel-Patterns enthalten.

Sei als Beispiel folgendes Tripel-Pattern gegeben: *http://northwind.beispiel.org/237421/order#10249 northw:order_date ?orderDate*. Des Weiteren sei als Teil des Graph-Templates einer Entitätsmenge folgendes Tripel-Template gegeben: *?order northw:order_date ?orderDate*. Es muss nun ermittelt werden, ob die Menge der durch dieses Tripel-Template beschriebenen Tripel ein Tripel mit dem Subjekt *http://northwind.beispiel.org/237421/order#10249* enthalten kann. Dazu wird zunächst das Format geprüft. Die IRI des Tripel-Patterns wurde nach dem Hash-Verfahren erzeugt (siehe Abschnitt 4.3). Das bedeutet, dass es nur dann eine Übereinstimmung geben kann, wenn die URIs der Ressourcen, die aus den Entitäten für die *?order*-Variable erzeugt werden, ebenfalls nach diesem Format erzeugt werden. Dies kann anhand des zugehörigen *sem:URI_struc*-Attributs ermittelt werden. Dass *sem:URI_struc*-Attribut ist für die Element *EntityType*, *EntitySet* und *DataServices* definiert. Wenn kein *sem:URI_struc* angegeben wurde, werden den Erläuterungen aus Abschnitt 4.3 entsprechend, die URIs nach dem Umleitungsverfahren erzeugt. Wenn für die *?order*-Ressourcen das Format mit *sem:URI_struc="hash"* spezifiziert wurde, werden im nächsten Schritt die *Host-* und *Path-Prefix*-Angabe überprüft. Die Host-Angabe der gegebenen IRI ist *northwind.beispiel.org*. Ein *Path-Prefix* ist nicht gegeben. Folglich muss der Wert des zur *?order*-Ressourcen gehörenden *sem:URI*-Attributs mit der Host-Angabe der IRI verglichen werden. Das *sem:URI*-Attribut kann entweder in der entsprechenden Entitätsmenge oder dem von diesem referenzierten Entitätstyp oder für alle Ressourcen als Erweiterung des *DataServices*-Elements definiert sein. Wenn der Wert mit der Host-Angabe übereinstimmt, muss im nächsten Schritt die *Entity-Location-ID* der Ressource berechnet werden. Dazu wird der Name der Entitätsmenge, zu dessen Graph-Template das Tripel-Template gehört, zusammen mit dem Namen des entsprechenden Entitätscontainers und der Service-Basis-URI an die Hash-Funktion übergeben. Der berechnete Wert muss der

Entity-Location-ID der gegebene IRI (*237421*) entsprechen. Als letztes erfolgt der Vergleich der Variablennamen, welche für das gegebene Beispiel übereinstimmen. Wenn alle erwähnten Vergleiche positiv sind, kann die Menge der Tripel, die durch das Tripel-Template beschrieben und aus den Entitäten der entsprechenden Entitätsmenge erzeugt werden, ein Tripel mit der gegebenen IRI als Subjekt enthalten. Wobei es sich in diesem Fall bei der IRI immer um eine URI handeln muss.

In Vorbereitung auf die Formalisierung des Algorithmus weiter unten, wird eine Funktion $matchI\_UT(i, ut)$ eingeführt, die für eine gegebene IRI $i \in I$ und ein gegebenes URI-Template $ut \in UT$ genau dann $true$ zurückgibt, wenn es sich bei der IRI um eine URI handelt und die URI und das URI-Template dasselbe Format, dieselbe Host-/Path-Prefix-Angabe, dieselbe Entity-Location-ID, denselben Variablen-Name und falls vorhanden, die gleichen Namen der Eigenschaften aufweisen. Anderenfalls gibt die Funktion $false$ zurück.

Subjekt-Literale

Die SPARQL-Spezifikation erlaubt Literale als Subjekte von Tripel-Patterns. Der RDF-Spezifikation entsprechend, sind als Subjekte ausschließlich URI-Referenzen und leere Knoten zugelassen. Aktuell kann es somit für ein Tripel-Pattern mit einem Literal als Subjekt keine Lösung geben. Wie in der SPARQL-Spezifikation angemerkt wird, weist die RDF Core Working Group darauf hin, dass kein Grund erkennbar ist, warum es nicht möglich sein sollte, Aussagen über Literale zu treffen, und das die Spezifikation in Zukunft möglicherweise dementsprechend angepasst wird [155]. SPARQL erlaubt die Angabe von Subjekt-Literalen, um solche zukünftigen Erweiterungen abzudecken [85]. Der in dieser Arbeit vorgestellte Ansatz zur Beschreibung der Abbildung gestattet, in Übereinstimmung mit der aktuellen RDF-Spezifikation, keine Literal-Templates als Subjekte. Für ein Tripel-Pattern mit einem Subjekt-Literal kann es daher kein Tripel-Template geben, welches ein Tripel beschreibt, für das eine Lösung existieren könnte.

Subjekt-Leere-Knoten und -Variablen

Leere Knoten verhalten sich der SPARQL-Spezifikation entsprechend wie Platzhalter, die über ein Instanz-Mapping σ auf Literale, leere Knoten oder URIs abgebildet werden [96]: $\sigma: B \longrightarrow L \cup B \cup U$. Listing 70 zeigt ein Beispiel. Das in Zeile 3 dargestellte Tripel-Pattern enthält als Subjekt den leeren Knoten *_:addr*. Dieser wird für den RDF-Graphen aus Listing 71 auf die URI *i:addr1* abgebildet: $\sigma_L(\_: addr) = i: addr1$. Ein Tripel-Template kann als Subjekt ein URI-Template oder ein Leerer-Knoten-Template enthalten. Für beide Fälle ist es möglich, eine Abbildung anzugeben. Wenn das Subjekt-Template einem URI-Template entspricht und für eine konkrete Ressource die URI $u \in U$ erzeugt wird, dann gilt, wie soeben an dem Beispiel gezeigt, $\sigma(tp.s) = u$. Analog dazu gilt, wenn es sich um ein Leerer-Knoten-Template handelt und für eine konkrete Ressource der leere Knoten $b \in B$ erzeugt wird, ist die Abbildung gegeben durch $\sigma(tp.s) = b$.

Das Gleiche gilt für Variablen. Wenn das Subjekt-Pattern einer Variablen entspricht, kann für das Lösungs-Mapping die Abbildung mit $\mu(tp.s) = u$ bzw. mit $\mu(tp.s) = b$ angegeben werden.

Prädikat-IRIs und -Variablen

Ein Tripel-Pattern kann als Prädikat eine IRI oder eine Variable aufweisen. Für ein Tripel-Template ist, den Erläuterungen zu Beginn dieses Abschnitts entsprechend, die Angabe einer URI erlaubt. Es sind daher zwei Fälle zu berücksichtigen. Wenn das Prädikat des Tripel-Patterns einer IRI entspricht, muss diese mit der URI des Tripel-Templates übereinstimmen, anderenfalls kann es niemals ein Tripel in der Menge der durch das Tripel-Template beschriebenen Tripel geben, auf welches das Tripel-Pattern abgebildet werden kann. Für eine Prädikats-Variable $tp.p \in V$ ist die Abbildung gegeben durch $\mu(tp.p) = tt.p$.

Objekt-IRIs

Wie für das Subjekt und das Prädikat des Tripel-Patterns ist auch für das Objekt die Angabe einer IRI erlaubt. Das Objekt eines Tripel-Templates kann unter anderem einem URI-Template und einem Leerer-Knoten-Template entsprechen. Diese beiden Fälle wurden bereits bei dem Vergleich der Subjekte abgehandelt. Wenn das Objekt-Template ein Leerer-Knoten-Template ist, kann es kein Tripel in der Menge der durch das Tripel-Template beschriebenen Tripel geben, welches die IRI als Objekt aufweist. Für ein URI-Template ist, wie weiter oben beschrieben, ein Vergleich des Formats und der Teilelemente (*Host-*/*Path-Prefix*-Angabe, *Entity-Location-ID* usw.) der Objekt-IRI mit dem Format und den entsprechenden Werten des URI-Templates erforderlich. Neben einem URI-Template und einem Leerer-Knoten-Template kann als Objekt des Tripel-Templates auch eine URI angegeben werden (siehe z.B. Listing 58 in Abschnitt 4.17). In diesem Fall muss die IRI des Tripel-Patterns mit der URI des Tripel-Templates verglichen werden. Nur bei Übereinstimmung kann es eine Lösung geben. Für Tripel-Templates, die ein Literal-Template, ein Funktions-Template oder ein Literal als Prädikat aufweisen, kann es niemals eine Lösung geben, da alle Tripel, die durch solch ein Tripel-Template beschrieben werden, nur Literale als Objekte aufweisen.

Objekt-Literale

Wenn das Objekt eines Tripel-Patterns einem Literal entspricht, können alle Tripel-Templates, die ein URI-Template, ein Leerer-Knoten-Template oder eine URI als Objekt-Template aufweisen, ausgeschlossen werden. Es bleiben daher folgende drei Fälle übrig, die einer genaueren Betrachtung bedürfen: Literal-Templates, Funktions-Templates und Literale. Es wird zunächst der Fall betrachtet, dass das Objekt des Tripel-Templates einem Literal entspricht. Einem Literal kann neben dem Literal-Wert ein Datentyp oder eine Sprachangabe zugewiesen sein. Wobei ein Literal niemals gleichzeitig einen Datentyp und eine Spracheangabe haben kann [155]. Der SPARQL-Spezifikation entsprechend, stimmen zwei Literale nur dann überein, wenn neben dem Wert auch der Datentyp bzw. die Sprachangabe übereinstimmen. Für ein Tripel-Pattern und ein Tripel-Template mit jeweils einem Literal als Objekt, kann es daher nur dann eine Lösung geben, wenn die Literale sowohl in ihren Werten, als auch in den Datentypen bzw. Sprachangaben übereinstimmen.

Wenn das Objekt eines Tripel-Templates einem Literal-Template entspricht, dann können, sofern keine Constraint-Angaben für die referenzierte Eigenschaft spezifiziert wurden, keine Aussagen über die Werte der Literale getroffen werden, die von dem Literal-Template repräsentiert werden. Es muss daher davon ausgegangen werden, dass in der Menge der vom Tripel-Template beschriebenen Tripel ein Tripel existiert, das als Objekt ein Literal

mit dem gleichen Wert aufweist, wie das als Teil des Tripel-Patterns gegebene Literal. Der Datentyp und die Sprachangabe des Literal-Templates können hingegen auf Basis der im semantisch erweiterten CSDL-Dokument gemachten Angaben ermittelt werden. Der Datentyp des Literal-Templates kann, wie in Abschnitt 4.4.2 beschrieben, in Turtle-Schreibweise für einzelne Literal-Templates spezifiziert werden. Des Weiteren besteht die Möglichkeit, über das *sem:Datatype*-Attribut für bestimmte Gruppen von Literal-Templates, die Abbildung auf untypisierte Literale festzulegen. Wenn weder ein Datentyp explizit angegeben wurde, noch die Abbildung auf untypisierte Literale spezifiziert wurde, dann wird der Datentyp auf Basis des Datentyps der Eigenschaft nach Tabelle 5 ermittelt. Der so ermittelte Datentyp des Literal-Templates wird mit dem Datentyp des Literals verglichen. Das gleiche gilt für die Sprachangabe, die ebenfalls in Anlehnung an die Turtle-Syntax oder über das *sem:Lang*-Attribut angegeben werden kann (siehe Abschnitt 4.4.2). Wenn sowohl die Datentypen als auch die Sprachenangaben übereinstimmen, kann in der Menge der vom Tripel-Template beschriebenen Tripel ein Tripel mit dem gegebenen Literal enthalten sein.

Wenn für die vom Literal-Template referenzierte Eigenschaft eine Constraint-Angabe spezifiziert wurde (siehe Abschnitt 4.16), dann muss diese bei dem Vergleich des Literals mit dem Literal-Template berücksichtigt werden. Sei, wie in Listing 56 dargestellt, für das Literal-Template *$UnitPrice* mit Bezug auf die Entitätsmenge *ExpensiveProducts* die Einschränkung *$UnitPrice > 50* definiert. Das Literal-Template referenziert dabei als Teil des Tripel-Templates *?product northw:price $UnitPrice* die Eigenschaft *UnitPrice*. Sei folgendes Tripel-Pattern gegeben: *?prod northw:price 7*. Der Constraint-Angabe entsprechend, kann die Menge der vom Tripel-Template beschriebenen Tripel niemals ein Tripel mit dem Literal-Wert *7* als Objekt enthalten. Somit kann es auch keine Lösung geben, die das Tripel-Pattern auf ein Tripel dieser Menge abbildet. Allgemein wird zur Auflösung einer Constraint-Angabe gegen ein konkretes Tripel-Pattern in Bezug auf ein bestimmtes Tripel-Template das Literal-Template des Tripel-Templates auf das Literal des Tripel-Patterns abgebildet. Anschließend wird dieser Abbildung entsprechend, das Literal-Template innerhalb des Constraints-Ausdruck, unabhängig von der Häufigkeit des Auftretens, durch das Literal ersetzt. Die Auswertung des Constraints-Ausdrucks erfolgt auf Basis einer dreiwertigen Logik (siehe dazu Abschnitt 5.5.5). Wenn der Constraint-Ausdruck zu *false* ausgewertet wird, kann es kein passendes Tripel geben. Anderenfalls (Auswertung zu *true* oder *unkown*) muss das Tripel-Template als Teil der Menge der relevanten Tripel-Templates berücksichtigt werden.

Wenn es sich bei dem Objekt des Tripel-Templates um ein Funktions-Template handelt, ergibt sich ein ähnliches Vorgehen wie bei dem soeben erläuterten Vergleich mit einem Literal-Template. Sofern nicht mittels des *sem:Datatype*-Attributs die Abbildung auf untypisierte Literale spezifiziert wurde, weist das Funktions-Template einen Datentyp auf. Dieser wurde entweder explizit als Teil der semantischen Annotation angegeben, oder er ergibt sich aus dem Rückgabetyp der verwendeten Funktion. Des Weiteren kann einem Funktions-Template wie bei einem Literal-Template auch eine Sprachangabe zugewiesen worden sein. Damit es für das Tripel-Pattern eine Lösung auf eines der durch das Tripel-Template beschriebenen Tripel geben kann, muss der Datentyp bzw. die Sprachangabe des Funktions-Template mit dem Datentyp bzw. der Sprachangabe des Literals des Tripel-Patterns übereinstimmen.

Objekt-Leere Knoten und -Variablen

Der Vergleich eines leeren Knotens oder einer Variable als Objekt eines Tripel-Patterns mit dem Objekt eines Tripel-Templates erfolgt analog zu dem weiter oben erläuterten Fall, bei dem das Subjekt-Pattern einem leeren Knoten oder einer Variable entspricht. Dabei müssen, neben dem URI-Template und dem Leerer-Knoten-Template, auch die Fälle berücksichtigt werden, in denen das Objekt-Template einer URI, einem Literal, einem Funktions-Template oder einem Literal-Template entspricht. Wenn es sich bei dem Objekt-Template um eine URI $u \in U$ handelt, dann kann für einen leeren Knoten $tp.o \in B$ das Instanz-Mapping mit $\sigma(tp.o) = u$ angegeben werden. Für eine Variable $tp.o \in V$ gilt analog dazu $\mu(tp.o) = u$. Ebenso gilt für ein Literal $l \in L$, das entweder als Teil der semantischen Annotation spezifziert wurde oder sich in der durch ein Funktion- oder Literal-Template beschriebenen Menge an Tripeln befindet: $\sigma(tp.o) = l$ bzw. $\mu(tp.o) = l$.

Aufbauend auf den vorhergehenden Erläuterungen, formalisiert Algorithmus 14 den Vergleich des Elements eines Tripel-Patterns mit dem Element eines Tripel-Templates. Als Eingabeparameter erhält die dargestellte Funktion das Subjekt, Prädikat oder Objekt eines Tripel-Patterns und das entsprechende Subjekt-, Prädikat- oder Objekt-Template. Die Funktion gibt $true$ zurück, wenn das Pattern auf ein Element in der durch das Template beschriebenen Menge an RDF-Termen abgebildet werden kann. Anderenfalls wird der Wert $false$ zurückgegeben. Dabei ist die durch ein Template t beschriebene Menge an RDF-Termen gegeben durch die Funktion $terms(t)$. Ω kennzeichnet die Menge aller Lösungen und Σ die Menge aller Instanz-Mappings.

Algorithmus 14: $m = matchPT(p, t)$

Eingabe: $p \in I \cup L \cup B \cup V$, $t \in UT \cup BT \cup U \cup FT \cup LT \cup L$
Ausgabe: $true$ wenn $\exists \mu \in \Omega, \exists \sigma \in \Sigma$: $\mu(\sigma(tp)) \in terms(t)$, sonst $false$

1. **begin**
2. **if** $p \in I \wedge \big((t \in UT \wedge matchI\_UT(p,t)\big) \vee (t \in U \wedge p = t))$ **then**
3. **return** $true$
4. **if** $p \in L \wedge ((t \in L \wedge p = t) \vee \big(\big(t \in FT \cup LT \wedge dt(p) = dt(t) \wedge l(p) = l(t)\big) \wedge$
5. $(t \in LT \longrightarrow eval\_const\_expr(t \mapsto p) \in \{true, unkown\})$
6. **then return** $true$
7. **if** $p \in B \cup V$ **then**
8. **return** $true$
9. **return** $false$
10. **end**

Wenn es sich bei dem übergebenen Pattern um eine IRI handelt (Zeile 2), dann muss das Template einem URI-Template (Subjekt- oder Objekt-Template) oder einer URI (Prädikat- oder Objekt-Template) entsprechen. Wenn es sich um ein URI-Template handelt, kann der Vergleich nur dann positiv ausfallen, wenn das Template einem URI-Template entspricht und die IRI und das URI-Template sowohl hinsichtlich des Formats, der Host-/Path-Prefix-Angabe, der Entity-Location-ID, des Namens der Ressourcen-Variable und falls vorhanden,

bzgl. der Namen der Eigenschaften übereinstimmen (*matchI_UT*-Funktion). Für eine URI muss die IRI des Patterns mit dieser übereinstimmen. Wenn als Pattern ein Literal gegeben ist (Zeile 4), dann muss den obigen Erläuterungen folgend, das Template einem Literal, einem Funktions-Template oder einem Literal-Template entsprechen. Wobei diese in Bezug auf den Datentypen bzw. der Sprachangabe übereinstimmen müssen. Für ein Literal müssen ebenfalls die Literal-Werte übereinstimmen. Die in Algorithmus 14 dargestellten Funktionen *dt* und *l* geben den Datentypen bzw. die Sprachangabe des übergebenen Templates oder Patterns zurück. Falls ein Constraint-Ausdruck existiert, wird dieser entsprechend den obigen Erläuterungen ausgewertet (*eval_const_expr*-Funktion in Zeile 5). Wobei die *eval_const_expr* für einen nicht vorhandenen Constraint-Ausdruck immer den Wert *true* zurück liefert. Wie gezeigt, kann für eine Variable oder einen leeren Knoten (Zeile 6) immer eine Abbildung angegeben werden.

Auf Basis von Algorithmus 14 kann ein Algorithmus zum Vergleich eines Tripel-Patterns mit einem Tripel-Template hergeleitet werden. Dabei ist es nicht ausreichend, das Subjekt-, Prädikat-, und Objekt eines Tripel-Patterns mit den entsprechenden Elementen des Tripel-Templates zu vergleichen. Es müssen auch die Fälle berücksichtigt werden, in denen eine Variable oder ein leerer Knoten mehrfach in einem Tripel-Pattern vorkommt. Sei als Beispiel folgendes Tripel-Pattern gegeben, welches alle Mitarbeiter abfragt, die an sich selbst berichten (Mitglieder des Vorstandes): *?employee northw:reports_to ?employee*. Da dieses Tripel-Pattern für das Subjekt und das Objekt die gleiche Variable aufweist, können sich für alle möglichen Lösungen nach Durchführung der Abbildung nur Tripel ergeben, die hinsichtlich des Subjekts und des Objekts übereinstimmen. Ein Tripel-Template muss daher bei der Verarbeitung der Anfrage nur dann berücksichtigt werden, wenn in der Menge der durch das Tripel-Template beschriebenen Tripel potentiell Tripel enthalten sein können, die übereinstimmende Subjekte und Objekte haben. Das Tripel-Template *?emp northw:reports_to $ReportsTo* weist als Subjekt ein URI-Template und als Objekt ein Literal-Template auf. Es kann daher mit Bezug auf das soeben gezeigte Tripel-Pattern ausgeschlossen werden, da eine Ressourcen-URI niemals einem Literal entsprechen kann.

Wenn ein Tripel-Pattern eine Variable oder einen leeren Knoten an mehreren Stellen aufweist, müssen die Template-Elemente (URI-Template, Literal-Template usw.) an den entsprechenden Stellen Mengen beschreiben, die potentiell gemeinsame RDF-Terme enthalten können. Anderenfalls kann es keine Lösung geben, die das Tripel-Pattern auf ein Tripel des Tripel-Templates abbildet. Es ist daher für zwei gegebene Template-Elemente zu prüfen, ob sich die Mengen der Terme, die durch die Template-Elemente beschrieben werden, überschneiden. Da ein Tripel-Pattern für das Subjekt, das Prädikat und das Objekt die gleiche Variable aufweisen kann, müssen alle möglichen Kombinationen von Subjekt-, Prädikat- und Objekt-Templates betrachtet werden. Mit Hinblick auf die weiter unten erläuterte Prüfung von Kombinationen von Tripel-Templates werden im Folgenden alle möglichen Kombinationen von Template-Elementen behandelt. Ein Template-Element kann einem URI-Template, einem Leerer-Knoten-Template, einer URI, einem Funktions-Template, einem Literal-Template oder einem Literal entsprechen.

Wenn ein URI-Template gegeben ist, muss das andere Template-Element ebenfalls einem URI-Template oder einer URI entsprechen. Alle anderen Template-Elemente beschreiben entweder Mengen von Literalen oder Mengen von leeren Knoten, für die es

niemals eine Überschneidung mit der durch das URI-Template beschriebenen Menge an URIs geben kann. Wenn als zweites Template-Element eine URI gegeben ist, kann diese URI nur dann in der URI-Menge des Templates enthalten sein, wenn wie weiter oben bereits erläutert, die URI und das URI-Template sowohl hinsichtlich des Formats, der Host-/Path-Prefix-Angabe, der Entity-Location-ID, des Variablen-Namens und falls vorhanden, der Eigenschaften-Namen übereinstimmen. Das gleiche gilt, wenn das zweite Template-Element einem URI-Template entspricht. Den Erläuterungen aus Abschnitt 4.3 entsprechend, können diese Eigenschaften bei zwei URI-Templates nur dann übereinstimmen, wenn es sich um das gleiche URI-Template handelt.

Sofern ein Leerer-Knoten-Template gegeben ist, muss es sich bei dem zweiten Leerer-Knoten-Template ebenfalls um ein Leerer-Knoten-Template handeln, da es das einzige Template-Element ist, das die Abbildung auf leere Knoten beschreibt. Zwei Leere-Knoten-Templates mit verschiedenen Namen werden immer auf unterschiedliche leere Knoten abgebildet. Daher müssen die Leerer-Knoten-Templates übereinstimmen.

Für eine URI müssen zwei Fälle unterschieden werden. Wenn es sich bei dem anderen Template-Element um ein URI-Template handelt, kann es, wie soeben bereits erläutert, nur dann eine Übereinstimmung geben, wenn die entsprechenden Eigenschaften übereinstimmen. Der zweite Fall besteht darin, dass das zweite Template-Element ebenfalls einer URI entspricht. Dabei muss es sich um dieselbe URI handeln.

Alle weiteren Template-Elemente (Funktions-Template, Literal-Template und Literal) beschreiben Mengen von Literalen. Wie weiter oben bereits erläutert, kann es Überschneidungen zwischen diesen Mengen nur dann geben, wenn diese hinsichtlich der Datentypen und Sprachangaben übereinstimmen. Wenn es sich um zwei Literale handelt, ist ebenfalls eine Übereinstimmung der Literal-Werte erforderlich. Da die Menge, die durch ein Funktions-Template, ein Literal-Template oder ein Literal beschrieben wird, niemals eine URI oder einen leeren Knoten enthalten kann, fällt der Vergleich mit einem URI-Template, einem Leerer-Knoten-Template oder einer URI immer negativ aus.

Aufbauend auf diesen Erläuterungen, formalisiert Algorithmus 15 den Vergleich zweier Template-Elemente.

Algorithmus 15: $m = matchTT(t_1, t_2)$

Eingabe: $t_1, t_2 \in UT \cup BT \cup U \cup FT \cup LT \cup L$
Ausgabe: $true$ wenn $terms(t_1) \cap terms(t_2) \neq \emptyset$, sonst $false$

1. **begin**
2. **if** $t_1 = t_2$ **then**
3. **return** $true$
4. **if** $t_1, t_2 \in L \wedge t_1 \neq t_2$ **then**
5. **return** $false$
6. **if** $t_1 \in U \wedge t_2 \in UT$ **then**
7. **return** $matchI\_UT(t_1, t_2)$
8. **if** $t_1 \in UT \wedge t_2 \in U$ **then**
9. **return** $matchI\_UT(t_2, t_1)$
10. **if** $t_1, t_2 \in FT \cup LT \cup L \wedge dt(t_1) = dt(t_2) \wedge l(t_1) = l(t_2)$ **then**
11. **return** $true$
12. **return** $false$
13. **end**

Auf Basis von Algorithmus 14 und Algorithmus 15 kann nun der Vergleich eines Tripel-Patterns mit einem Tripel-Template formalisiert werden. Algorithmus 16 liefert den Wert $true$, wenn es eine Lösung geben könnte, die das Tripel-Pattern auf ein Tripel der durch das Tripel-Template beschriebenen Menge an Tripeln abbildet.

Algorithmus 16: $m = matchTP\_TT(tp, tt)$

Eingabe: $tp \in TP$, $tt \in TT$
Ausgabe: $true$ wenn $\exists \mu \in \Omega \wedge \exists \sigma \in \Sigma$: $\mu(\sigma(tp)) \in triples(tt)$, ansonsten $false$

1. **begin**
2. **if** $!matchPT(tp.s, tt.s)$ **then**
3. **return** $false$
4. **if** $!matchPT(tp.p, tt.p) \vee (tp.p \in V \wedge tp.p = tp.s \wedge !matchTT(tt.s, tt.p))$ **then**
5. **return** $false$
6. **if** $!matchPT(tp.o, tt.o) \vee (tp.o \in V \cup B \wedge tp.o = tp.s \wedge !matchTT(tt.s, tt.o)) \vee$
7. $(tp.o \in V \wedge tp.o = tp.p \wedge !matchTT(tt.p, tt.o))$ **then**
8. **return** $false$
9. **return** $true$
10. **end**

Zunächst wird in Zeile 2 das Subjekt des Tripel-Patterns mit dem Subjekt des Tripel-Templates verglichen. Wenn keine Abbildung möglich ist, bricht der Algorithmus ab. Die gleiche Prüfung wird in Zeile 4 für die Prädikate durchgeführt. Zusätzlich wird geprüft, ob das Prädikat und das Subjekt des Tripel-Patterns die gleiche Variable aufweisen. Wenn dies der Fall ist, wird mittels Algorithmus 15 geprüft, ob es in der Menge der durch das Tripel-Template beschriebenen Tripel ebenfalls ein Tripel mit gleichem Subjekt und Prädikat geben

kann. Als letztes erfolgt in den Zeilen 6 und 7 der Vergleich der Objekte. Wobei auch hier eine Prüfung hinsichtlich des mehrfachen Auftretens von Variablen bzw. leeren Knoten und ein Vergleich der entsprechenden Elemente des Tripel-Templates erfolgt.

5.5 Eine Evaluierungssemantik für Graph-Templates

Aufbauend auf Algorithmus 16 ist es bereits möglich, einen Algorithmus zu realisieren, der für eine gegebene SPARQL-Anfrage alle Daten abruft, die zur Ermittlung aller Lösungen benötigt würden. Dazu müsste jedes Tripel-Pattern der SPARQL-Anfrage mittels Algorithmus 16 mit allen Tripel-Templates aller am System registrierten Services verglichen werden. Die Menge der erhaltenen Tripel-Templates würde alle Tripel-Templates umfassen, die Tripel beschreiben, auf die es potentiell eine Abbildung von einem Tripel-Pattern der Anfrage geben könnte. Dieser Algorithmus wäre daher vollständig. Es lässt sich aber leicht zeigen, dass er nicht sehr effizient ist, da mehr Daten abgerufen werden, als tatsächlich benötigt werden.

Sei folgendes einfaches Graph-Muster als Teil einer SPARQL-Anfrage gegeben: *?order rdf:type northw:Order . ?order northw:employee_id ?employeeID*. Dieses einfache Graph-Muster fragt nach allen Bestellungen und den IDs der Angestellten, die für diese Bestellungen verantwortlich sind. Sei für die *Orders*-Entitätsmenge folgendes Graph-Template gegeben: *?o rdf:type northw:Order . ?o northw:employee_id $EmployeeID*. Entsprechend des obigen Algorithmus müssen korrekterweise beide Tripel-Templates bei der Verarbeitung der Anfrage berücksichtigt werden. Sei des Weiteren die *Employees*-Entitätsmenge gegeben, die als Teil ihres Graph-Templates folgendes Tripel-Template enthält: *?employee northw:employee_id $EmployeeID*. Dieses Tripel-Template würde ebenfalls in die Menge der relevanten Tripel-Templates aufgenommen werden, da es, entsprechend Algorithmus 16, für das Tripel-Pattern *?order northw:employee_id ?employeeID* eine Abbildung geben könnte. Da aber das Graph-Template der *Employees*-Entitätsmenge kein Tripel-Template enthält, auf welches das Tripel-Pattern *?order rdf:type northw:Order* abgebildet werden könnte, kann es eine Lösung für das einfache Graph-Muster unter Einbeziehung des Tripel-Templates der *Employees*-Entitätsmenge nicht geben. Der Grund dafür ist, dass eine Lösung die *?order*-Variable für beide Tripel-Patterns immer auf die gleiche URI abbildet. Die Mengen der Ressourcen-URIs, die durch die Ressourcen-Variablen *?o* und *?employee* beschrieben werden, können aber keine gemeinsamen URIs enthalten, da sie aus den Entitäten unterschiedlicher Entitätsmengen erzeugt werden und sich in den Variablen-Namen unterscheiden. Somit ist es nicht notwendig, das Tripel-Template *?employee northw:employee_id $EmployeeID* bei der Verarbeitung der Anfrage zu berücksichtigen.

Allgemein ist es nicht erforderlich, jedes Tripel-Pattern mit jedem Tripel-Template der registrierten Services zu vergleichen, um alle relevanten Tripel-Templates zu bestimmen. Da die Subjekte der Tripel-Patterns, die ein gemeinsames Subjekt aufweisen, von einer Lösung immer auf die gleiche Ressourcen-URI oder den gleichen leeren Knoten abgebildet werden, ist es ausreichend, diese Tripel-Patterns in ihrer Gesamtheit mit Tripel-Templates gleichen Subjekts zu vergleichen. Mit dieser Vorgehensweise würde die Anzahl der notwendigen Vergleiche und die Menge der zu übertragenden Daten bereits erheblich reduziert. Um jedoch ein Verfahren zu erhalten, das hinsichtlich der zu übertragenden Datenmenge optimal ist, ist

es nicht ausreichend, die Tripel-Patterns mit gleichen Subjekt-Variablen gemeinsam zu verarbeiten. Vielmehr ist generell ein Verfahren erforderlich, mit dem Variablen, die in mehr als einem Tripel-Pattern auftreten, abhängig voneinander mit den entsprechenden Tripel-Templates zu vergleichen.

Sei als weiteres Beispiel folgendes einfaches Graph-Muster gegeben: *?order rdf:type northw:Order . ?order northw:order_date ?x . ?order northw:ship_country ?x*. Die Variable *?x* tritt dabei in zwei der dargestellten Tripel-Patterns auf. Es sei nun das Graph-Template der *Orders*-Entitätsmenge gegeben: *?or rdf:type northw:Order . ?or northw:order_date $OrderDate^^xsd:dateTime . ?or northw:ship_country $ShipCountry^^xsd:string*. Bei einem Vergleich des einfachen Graph-Musters mit dem Graph-Template auf Basis von Algorithmus 16 würde sich ergeben, dass für jedes Tripel-Pattern ein passendes Tripel-Template existiert. Trotzdem kann es keine Lösung geben, welche das einfache Graph-Muster auf eine Untermenge der durch das Graph-Template beschriebenen Menge an Tripeln abbildet. Der Grund dafür liegt darin, dass eine Lösung die Variable *?x* für beide Tripel-Pattern, in denen diese Variable auftritt, auf das gleiche Literal abbilden müsste. Die entsprechenden Literal-Mengen sind durch die beiden Literal-Templates *$OrderDate^^xsd:dateTime* und *$ShipCountry^^xsd:string* beschrieben. Da sich diese Literal-Templates in den Datentypen unterscheiden, können beide keine gemeinsamen Literale enthalten. Folglich kann es auch keine Lösung geben. Generell kann es für mehrere Tripel-Patterns, die gemeinsame Variablen enthalten, nur dann eine Lösung geben, wenn die entsprechenden Templates Mengen beschreiben, die gemäß Algorithmus 15 gemeinsame Elemente enthalten können.

Um nur die Tripel-Templates zu berücksichtigen, die zur Bestimmung der Antwort einer Anfrage tatsächlich benötigt werden, wird im Folgenden eine Evaluierungssemantik für Graph-Templates hergeleitet. Wie in Abschnitt 2.2.2 anhand von Tabelle 4 erläutert, unterteilt die SPARQL-Spezifikation die SPARQL-Algebra-Operatoren in zwei Gruppen: Graph-Pattern-Operatoren und Ergebnismengen-Modifikatoren. Dem Namen entsprechend beschreiben die Graph-Pattern-Operatoren die für die SPARQL-Anfrage relevanten Graph-Muster. Die Ergebnismengen-Modifikatoren hingegen verändern die ermittelte Menge von Ergebnissen (Lösungen). Obwohl für bestimmte Arten von SPARQL-Anfragen und bestimmte Ergebnismengen-Modifikatoren durch eine Abbildung der Modifikatoren auf OData-Abfrageoptionen eine Verbesserung der Performance erreicht werden kann, ist dies für den allgemeinen Fall nicht möglich. Daher werden im Folgenden ausschließlich die Graph-Pattern-Operatoren betrachtet.

5.5.1 Pattern-Instanz-Template-Mappings

Zunächst wird, angelehnt an die entsprechenden Definitionen der SPARQL-Spezifikation, die Definition der Pattern-Instanz-Mappings auf Graph-Templates angepasst. Sei $TE = UT \cup BT \cup FT \cup LT \cup U \cup L$ die Menge aller Templates. Dann wird die partielle Funktion $\varrho: B \longrightarrow TE$ als *RDF-Instanz-Template-Mapping* (*RITM*) und die partielle Funktion $\vartheta: V \longrightarrow TE$ als *Lösungs-Template-Mapping* (*LTM*) bezeichnet. Die Definition der Funktion ϱ erfolgt mit Hinblick auf die in der SPARQL-Spezifikation [155] gemachte Definition des RDF-Instanz-Mappings $\sigma: B \longrightarrow L \cup B \cup U$. Wie in Abschnitt 2.2.3 beschrieben, bildet das RDF-Instanz-Mapping eines Pattern-Instanz-Mappings alle leeren Knoten eines einfachen Graph-Musters

auf Literale, leere Knoten oder URIs ab. Analog dazu, bildet ein RDF-Instanz-Template-Mapping die leeren Knoten eines Patterns auf Templates ab. Ein Lösungs-Mapping (*solution mapping*) μ bildet der SPARQL-Spezifikation entsprechend Variablen auf RDF-Terme ab: $\mu: V \longrightarrow T$. Angelehnt an diese Definition, erfolgt durch ein *Lösungs-Template-Mapping* ϑ die Abbildung von Variablen auf Templates. Zur Vereinfachung der Schreibweise wird im Folgenden, im Widerspruch zu den Funktionsdefinitionen, aber wie in der Literatur üblich [84, 148] und auch in der SPARQL-Spezifikation selbst angewendet, für eine Menge von Tripel-Patterns $tps \subset TP$ mit $\vartheta(tps)$ und $\varrho(tps)$ die Menge an Tripel-Templates gekennzeichnet, die nach der Abbildung aller Variablen der Tripel-Patterns entsprechend ϑ bzw. aller leerer Knoten entsprechend ϱ entstehen.

Aufbauend auf der Definition des RDF-Instanz-Mappings und des Lösungs-Mappings definiert die SPARQL-Spezifikation das Pattern-Instanz-Mapping: $P = \mu(\sigma(x))$. Mit Hinblick darauf, wird für ein RDF-Instanz-Template-Mapping ϱ und ein Lösungs-Template-Mapping ϑ die Funktion $\varphi(x) = \vartheta(\varrho(x))$ als *Pattern-Instanz-Template-Mapping* (*PITM*) bezeichnet.

Aufbauend auf diesen Definitionen, kann nun hergeleitet werden, wie für ein nach Algorithmus 16 passendes Tripel-Template ein Pattern-Instanz-Template-Mapping (Zeile 7) erzeugt wird. Ein PITM ist immer eine Komposition aus einem Lösungs-Template-Mapping und einem RDF-Instanz-Template-Mapping. Zunächst wird die Erstellung des RDF-Instanz-Template-Mappings für ein gegebenes Tripel-Pattern in Bezug auf ein bestimmtes Tripel-Template betrachtet. Den Erläuterungen zu Beginn dieses Abschnitts entsprechend, bildet ein RITM die leeren Knoten eines Tripel-Patterns auf Templates ab. Die SPARQL-Spezifikation erlaubt die Verwendung von leeren Knoten als Subjekt und Objekt eines Tripel-Patterns. Es ist zu definieren, wie diese auf die Templates eines Tripel-Templates abgebildet werden. Sei als Beispiel folgendes Tripel-Pattern gegeben: *_:n rdf:type northw:Employee*. Für das Tripel-Template *?employee rdf:type northw:Employee* kann ein RITM ϱ über die folgende Abbildung definiert werden: $\varrho(\_:n) = ?employee$. Wenn das gegebene Tripel-Pattern nur einen leeren Knoten enthält oder sich die im Tripel-Pattern vorkommenden leeren Knoten unterscheiden, erfolgt die Abbildung entsprechend der Position der leeren Knoten im Tripel-Pattern. Das bedeutet, dass ein leerer Knoten an der Subjekt-Position eines Tripel-Patterns auf das Subjekt-Template des Tripel-Templates abgebildet wird. Bei einem leeren Knoten an Objekt-Position erfolgt die Abbildung hingegen auf das Objekt-Template.

Bei der Erstellung eines RITM für ein Tripel-Pattern, das sowohl als Subjekt als auch als Objekt den gleichen leeren Knoten aufweist, müssen zwei Fälle unterschieden werden. Wenn das Subjekt- und das Objekt-Template des gegebenen Tripel-Templates ebenfalls übereinstimmen, erfolgt die Abbildung des leeren Knoten auf dieses Template. Der Fall, dass Subjekt- und Objekt-Template nicht übereinstimmen, kann für ein Tripel-Template mit demselben leeren Knoten als Subjekt und Objekt, nur dann auftreten, wenn es sich bei dem Subjekt-Template um ein URI-Template und bei dem Objekt-Template um eine URI handelt. Als Beispiel sei das Tripel-Pattern *_:n northw:reports_to _:n* und das Tripel-Template *?employee northw:reports_to <http://northwind.beispiel.org/237421/employee#10249>* gegeben. In diesem Fall bestehen zwei Möglichkeiten, die Abbildung des leeren Knotens zu definieren: $\varrho(\_:n) = ?employee$ oder $\varrho(\_:n) = http://northwind.beispiel.org/237421/$

$employee\#10249$. Da alle der durch das Tripel-Template beschriebenen Tripel als Objekt die dargestellte URI aufweisen, kann es für eine Lösung μ nur ein RDF-Instanz-Mapping σ geben, welches den leeren Knoten auf diese URI abbildet. Somit müssen, im Rahmen der Auflösung eines einfachen Graph-Musters, ebenfalls nur die RDF-Instanz-Templates-Mappings berücksichtigt werden, die den leeren Knoten auf die URI abbilden. Daher wird für den oben genannten Fall der leere Knoten immer auf die URI (Objekt-Template) abgebildet. Damit ist die Erzeugung des RDF-Instanz-Template-Mappings ϱ_{tp-tt} für ein gegebenes Tripel-Pattern tp in Bezug auf ein nach Algorithmus 16 *passendes* Tripel-Template tt vollständig beschrieben:

$$\varrho_{tp-tt}(n) \coloneqq \begin{cases} tt.o, & tp.o = n \\ tt.s, & tp.s = n \wedge tp.o \neq n \end{cases}$$

Die Erzeugung des Lösungs-Template-Mappings für ein Tripel-Pattern mit Hinblick auf ein Tripel-Template erfolgt analog zu der Erzeugung des RDF-Instanz-Template-Mappings. Wobei die SPARQL-Spezifikation im Gegensatz zu leeren Knoten die Verwendung von Variablen auch an Prädikat-Position zulässt. Wenn das Tripel-Pattern nur eine Variable enthält oder sich die im Tripel-Pattern vorkommenden Variablen unterscheiden, bildet das Lösungs-Template-Mapping die Variablen, entsprechend ihrer Position im Tripel-Pattern, auf die Templates des Tripel-Templates ab. Wenn dieselbe Variable mehrfach in einem Tripel-Pattern vorkommt, müssen zwei Fälle unterschieden werden. Entweder es stimmen die Templates an den entsprechenden Stellen des Tripel-Templates überein (dieselben URIs, URI-Templates oder Leerer-Knoten-Templates), oder es handelt sich um mindestens ein URI-Template und eine URI. Im ersten Fall erfolgt die Abbildung der Variablen auf das gemeinsame Template. Im zweiten Fall wird aus den gleichen Gründen, wie bei der Erstellung des RDF-Instanz-Template-Mappings, die Variable immer auf die URI abgebildet. Es ergibt sich somit für ein Tripel-Pattern tp und ein nach Algorithmus 16 *passendes* Tripel-Template tt das Lösungs-Template-Mapping ϑ_{tp-tt} nach folgendem Schema:

$$\vartheta_{tp-tt}(v) \coloneqq \begin{cases} tt.o, & tp.o = v \wedge tp.p \neq v \\ tt.p, & tp.p = v \\ tt.s, & tp.s = v \wedge tp.o \neq v \wedge tp.p \neq v \end{cases}$$

Um die Formalisierung kompakter darstellen zu können, werden weitere Schreibweisen eingeführt. Für ein LTM ϑ bezeichnet $\vartheta.TT$ die Menge der Tripel-Templates, auf denen das LTM basiert. Das Tripel-Template tt, für welches das Lösungs-Template-Mapping ϑ definiert wurde, wird Element der Menge $\vartheta.TT$. Die Menge $\vartheta.TT$ enthält alle Tripel-Templates, die zur Bestimmung aller Lösungen, welche durch das Lösungs-Template-Mapping beschrieben sind, benötigt werden. Dabei werden ebenfalls die Tripel-Templates in die Menge $\vartheta.TT$ aufgenommen, auf deren Templates es keine Abbildung einer Variablen gibt, deren Tripel aber bei der Bestimmung des Pattern-Instanz-Mappings relevant sind. Für jedes Tripel-Template, das zu einem bestimmten Tripel-Pattern *passt*, wird somit ein Lösungs-Template-Mapping erstellt. Wobei dieses Lösungs-Template-Mapping ϑ die Abbildung keiner oder maximal dreier Variablen definiert und in der Menge $\vartheta.TT$ genau ein Tripel-Template enthält. Analog

zu dieser Schreibweise ist die Menge der Tripel-Templates eines PITM φ gegeben durch $\varphi.TT$. Wenn ein PITM φ auf einem LTM ϑ basiert gilt: $\varphi.TT = \vartheta.TT$.

Für jedes Tripel-Template wird, wie in Abschnitt 5.6 beschrieben, eine OData-URI erzeugt, die alle zur Erzeugung der durch das Tripel-Template beschriebenen Tripel benötigten Daten adressiert. Die Menge an OData-URIs, die ausgeführt werden müssen, um die von den Tripel-Templates der Menge $\varphi.TT$ beschriebene Menge an Tripeln erzeugen zu können, ist durch $\varphi.OU$ gegeben. Das Lösungs-Template-Mapping und das RDF-Instanz-Template-Mapping eines PITM werden durch $\varphi.\vartheta$ bzw. $\varphi.\varrho$ repräsentiert.

5.5.2 Auflösung von Tripel-Patterns

Basierend auf der Definition von Pattern-Instanz-Template-Mappings, Lösungs-Template-Mappings und RDF-Instanz-Template-Mappings in dem vorherigen Abschnitt, erfolgt nun die Definition des Lösungsbegriffs für Tripel-Patterns. Darauf aufbauend, wird in den folgenden Abschnitten die Evaluierungssemantik der SPARQL-Algebra-Operatoren mit Bezug auf die Templates angepasst.

Für ein Tripel-Pattern tp ist ein Pattern-Instanz-Template-Mapping φ mit Hinblick auf ein zu tp *passendes* Tripel-Template tt eine *Lösung*, wenn ein Lösungs-Template-Mapping ϑ_{tp-tt} und ein Instanz-Template-Mapping ϱ_{tp-tt} existiert, so dass gilt: $\varphi = \vartheta_{tp-tt} \circ \varrho_{tp-tt}$.

Mit Hinblick auf die Auflösung der Graph-Pattern-Operatoren werden in Anlehnung an die Definition von Datensätzen und Graphen durch die SPARQL-Spezifikation im Folgenden *Datensatz-Templates* und *Graph-Templates* definiert. Dabei wird die Formulierung aus der SPARQL-Spezifikation [155] übernommen (siehe dazu auch Abschnitt 2.2.3). Ein Graph-Template GT ist eine Menge von Tripel-Templates: $GT \subset (UT \cup BT) \times U \times (UT \cup BT \cup FT \cup LT \cup U \cup L)$. Das Graph-Template eines Services setzt sich aus den Tripel-Templates des Services zusammen. Für eine Menge von Graph-Templates $GT, GT_1, GT_2, \dots, GT_n$ und eine Menge von URIs $u_1, u_2, \dots, u_n$ ist ein Datensatz-Template DT definiert als $DT \coloneqq \{GT, (<u_1>, GT_1), (<u_2>, GT_2), \dots, (<u_n>, GT_n)\}$. Dabei ist jede $<u_i>$ für $i \in \{1, 2, \dots, n\}$ die Basis-URI eines Services zu dem das Graph-Template GT_i gehört. Das Graph-Template GT wird als das *Standard-Graph-Template* bezeichnet. Die Tupel $(<u_i>, GT_i)$ sind die *benannten Graph-Templates*.

Des Weiteren bezeichnen im Folgenden, wie auch in der SPARQL-Spezifikation üblich, P, P_1 und P_2 Graph-Mustern. F ist ein Filter-Ausdruck. Die in den folgenden Abschnitten verwendeten Formalismen entsprechen den in der Literatur üblichen (siehe z.B. [32, 84, 85, 101, 148]). Insbesondere wird für ein Graph-Muster P die Auswertung über das Datensatz-Template DT mit dem aktiven Graph-Template GT mit $[\![P]\!]_{DT,GT}$ angegeben. Dabei entspricht das Ergebnis der Auswertung eines SPARQL-Algebra-Ausdrucks im Gegensatz zur Evaluierungssemantik, wie sie durch die SPARQL-Spezifikation definiert ist, nicht einer Multimenge sondern einer Menge von PITMs bzw. LTMs. Der Grund dafür liegt darin, dass es für die Erzeugung der OData-URIs nicht von Bedeutung ist, wie oft eine mögliche Lösung existiert.

Darauf aufbauend, kann nun die Auswertung eines Tripel-Patterns tp gegen ein Datensatz-Template DT mit dem aktiven Graph-Template GT definiert werden:

$$[\![tp]\!]_{DT,GT} = \{\varphi \mid Es\ existiert\ ein\ tt \in GT, so\ dass\ \varphi\ eine\ Lösung\ für\ tp\ mit\ Bezug\ auf\ tt\ ist\}$$

5.5.3 BGP

Aufbauend auf der im letzten Abschnitt beschriebenen Auflösung von Tripel-Patterns, kann nun die Auswertung des *BGP*-Operators für ein Datensatz-Template DT mit dem aktiven Graph-Template GT folgendermaßen formuliert werden:

$$[\![BGP]\!]_{DT,GT} = \{\vartheta \mid \vartheta \in stmsOf([\![tp_1]\!]_{DT,GT} \bowtie [\![tp_2]\!]_{DT,GT} \bowtie \cdots \bowtie [\![tp_n]\!]_{DT,GT})\}$$

Wobei $tp_1, tp_2, \ldots, tp_n$ die Tripel-Patterns von *BGP* sind. Die Funktion *stmsOf* gibt für jedes Pattern-Instanz-Template-Mapping das Lösungs-Template-Mapping zurück, dessen Domäne die Variablen des Pattern-Instanz-Template-Mappings enthält und diese entsprechend des Pattern-Instanz-Template-Mappings abbildet.

Die Definition des Joins zweier Mengen von PITMs erfolgt im Folgenden, in Anlehnung an die in der SPARQL-Spezifikation gemachte Definition der Join-Operation für zwei Multimengen von Lösungen. Dazu ist zunächst die Definition der Kompatibilität zweier Pattern-Instanz-Template-Mappings erforderlich. Die SPARQL-Spezifikation definiert die Kompatibilität zweier Lösung-Mappings. Zwei Lösung-Mappings sind zueinander kompatibel, wenn sie alle Variablen, die in den Definitionsmengen beider Lösung-Mappings enthalten sind, auf dieselben RDF-Terme abbilden [155]. Analog dazu, erfolgt die Definition der Kompatibilität zweier PITMs. Wobei im Gegensatz zu den Lösung-Mappings nicht die Gleichheit der Elemente der Zielmenge für in beiden Definitionsmengen enthaltene Elemente relevant ist. Vielmehr müssen die durch die entsprechenden Templates beschriebenen Mengen der Zielmenge gemeinsame Elemente enthalten.

Als Beispiel seien die folgenden beiden Tripel-Templates gegeben: *?order northw:order_date $OrderDate* und *?order northw:shipped_date $ShippedDate*. Des Weiteren seien als Teil eines einfachen Graph-Musters folgende Tripel-Pattern gegeben, welche alle Bestellungen abfragen, die am Bestelltag ausgeliefert wurden: *?or northw:order_date ?date* und *?or northw:shipped_date ?date*. Für die beiden Tripel-Patterns können, mit Hinblick auf die beiden Tripel-Templates, zwei PITMs generiert werden. Dabei bildet das erste PITM die *?date*-Variable auf das Literal-Template *$OrderDate* ab. Das zweite PITM hingegen, bildet die *?date*-Variable auf das Literal-Template *$ShippedDate* ab. In der Menge aller Literale, die durch das *$OrderDate*-Literal-Template beschrieben werden, kann es aber ein Datum geben, das auch in der Menge der Daten enthalten ist, die durch das *$ShippedDate*-Literal-Template beschrieben werden. Daher besteht die Möglichkeit, dass als Teil eines Pattern-Instanz-Mappings eine Lösung für die Tripel-Patterns mit Hinblick auf die durch die Tripel-Templates beschriebenen Tripel existiert.

Auf Basis dieser Erläuterungen ergibt sich die folgende Definition für die Kompatibilität zweier PITMs:

$$comp(\varphi_1, \varphi_2) \coloneqq \begin{cases} true, & \forall p \in dom(\varphi_1) \cap dom(\varphi_2) \longrightarrow terms(\varphi_1(p)) \cap terms(\varphi_2(p)) \neq \emptyset \\ false, & in\ allen\ anderen\ Fällen \end{cases}$$

Wobei die Bedingung $terms(\varphi_1(p)) \cap terms(\varphi_2(p)) \neq \emptyset$ mittels Algorithmus 15 geprüft werden kann. Ausgehend von der Definition der Kompatibilität kann der Join zweier Mengen von PITMs Φ_1 und Φ_2.in Anlehnung an die Join-Definition zweier Multimengen von Lösung-Mappings durch die SPARQL-Spezifikation folgendermaßen definiert werden:

$$\Phi_1 \bowtie \Phi_2 \coloneqq \{\varphi_1 \cup \varphi_2 |\ \varphi_1 \in \Phi_1 \wedge \varphi_2 \in \Phi_2 \wedge comp(\varphi_1, \varphi_2)\}$$

Die SPARQL-Spezifikation definiert die Vereinigung zweier Lösung-Mappings über die *merge*-Operation. Wenn eine Variable in der Schnittmenge beider Definitionsbereiche enthalten ist, wird sie in dem vereinigten Lösungs-Mapping auf das gemeinsame Zielelement abgebildet. Alle weiteren Elemente in den Definitionsbereichen werden, entsprechend ihrer Abbildung, durch das jeweilige Lösungs-Mapping, in das vereinigte Lösungs-Mapping übernommen [155]. Mit Hinblick auf die Vereinigung zweier PITMs ist eine Erweiterung dieser Definition erforderlich. Der Grund dafür folgt aus der oben dargestellten Definition der Kompatibilität. Da es zwei zueinander kompatible PITMs geben kann, welche dieselbe Variable oder denselben leeren Knoten auf unterschiedliche Templates abbilden, muss eindeutig spezifiziert werden, auf welches der beiden Templates die Variable oder der leere Knoten in dem vereinigten PITM abgebildet wird. Entsprechend der Definition des PITMs in Abschnitt 5.5.1 gilt für ein PITM φ bzgl. eines Elements aus dem Definitionsbereich $p \in dom(\varphi)$ immer $\varphi(p) \in UT \cup BT \cup FT \cup LT \cup U \cup L$. Zwei nach Algorithmus 15 *passende* Templates können sich nur in zwei Fällen unterscheiden. Entweder entspricht eines der beiden Templates einem URI-Template und das andere Template einer URI in dem gleichen Format, oder beide Templates entsprechen Funktions-Templates, Literal-Templates oder Literalen, die hinsichtlich der Datentypen und der Sprachangaben übereinstimmen. Zunächst wird der erste Fall betrachtet. Wenn eines der beiden PITMs für ein Element aus dem Definitionsbereich auf eine URI abgebildet wird und das andere PITM für dasselbe Element auf ein URI-Template, muss das sich aus der Vereinigung der PITMs ergebende PITM das Element auf die URI abbilden. Der Grund dafür besteht darin, dass jedes mögliche Pattern-Instanz-Mapping das Element auf diese URI abbilden muss, unabhängig davon, welche weiteren URIs in der Menge der durch das URI-Template beschriebenen URIs enthalten sind. Die gleiche Argumentation gilt, wenn ein gemeinsames Element des Definitionsbereichs von einem PITM auf ein Literal abgebildet wird. In diesem Fall muss das sich aus der Vereinigung ergebende PITM das Element ebenfalls auf dieses Literal abbilden. Wobei es nicht von Bedeutung ist, ob das zweite PITM dieses Element auf ein Literal-Template oder ein Funktions-Template abbildet. Wenn beide PITMs das gemeinsame Element auf ein Literal-Template oder ein Funktions-Template abbilden, müssen diese mit Bezug auf die Datentypen und Sprachangaben übereinstimmen. Da für die weitere Bestimmung der Lösungs-Template-

Mappings nur der Datentyp und die Sprachangabe relevant sind, macht es keinen Unterschied, ob das Element von dem sich ergebenden PITM auf das erste oder das zweite Template abgebildet wird. Ausgehend von diesen Erläuterungen, wird die Abbildung der sich aus zwei PITMs φ_1 und φ_2 ergebenden Vereinigung $\varphi = \varphi_1 \cup \varphi_2$ folgendermaßen definiert:

$$\varphi(p) = \varphi_1 \cup \varphi_2(p)$$
$$:= \begin{cases} \varphi_1(p), & p \in dom(\varphi_1) \wedge (\varphi_1(p) \in L \cup U \vee p \notin dom(\varphi_2) \vee \varphi_2(p) \notin L \cup U) \\ \varphi_2(p), & p \in dom(\varphi_2) \wedge (\varphi_2(p) \in L \cup U \vee p \notin dom(\varphi_1) \vee \varphi_1(p) \notin L \cup U) \end{cases}$$

Neben der Definition der Abbildung müssen auch die Vereinigungen der OData-URIs und der Tripel-Templates der PITMs spezifiziert werden. Für die Mengen der relevanten Tripel-Templates ergibt sich die Zielmenge aus der Vereinigung der beiden Mengen: $\varphi.TT = \varphi_1.TT \cup \varphi_2.TT$. Die Mengen der OData-URIs werden aus Gründen der Performance nicht durch bloße Vereinigung zusammengefasst. Stattdessen werden diese, soweit möglich, miteinander verschmolzen (siehe dazu Abschnitt 5.6.3).

5.5.4 Union und Join

Die Auswertung der *Union* und *Join* Operatoren erfolgt ebenfalls in Einklang mit der SPARQL-Spezifikation:

$$[\![Union(P_1, P_2)]\!]_{DT,GT} = [\![P_1]\!]_{DT,GT} \cup [\![P_2]\!]_{DT,GT}$$
$$[\![Join(P_1, P_2)]\!]_{DT,GT} = [\![P_1]\!]_{DT,GT} \bowtie [\![P_2]\!]_{DT,GT}$$

Dabei ist die Vereinigung zweier Mengen von Lösungs-Template-Mappings folgendermaßen definiert:

$$\Theta_1 \cup \Theta_2 = \{\vartheta \mid \vartheta \in \Theta_1 \vee \vartheta \in \Theta_2\}$$

Der Join zweier Mengen von Lösungs-Template-Mappings wird in Anlehnung an die Definition des Joins zweier Multimengen von Lösungs-Mappings in der SPARQL-Spezifikation folgendermaßen angegeben:

$$\Theta_1 \bowtie \Theta_2 = \{\vartheta_1 \cup \vartheta_2 \mid \vartheta_1 \in \Theta_1 \wedge \vartheta_2 \in \Theta_2 \wedge comp(\vartheta_1, \vartheta_2)\}$$

Wobei die Definition der Kompatibilität aus den gleichen Gründen wie bei dem Join von PITMs (Abschnitt 5.5.3) dahingehend abgeändert wird, dass nicht die Gleichheit der Elemente der Zielmenge für in beiden Definitionsmengen enthaltene Variablen von Bedeutung sind. Stattdessen müssen die durch die entsprechenden Templates der Zielmenge beschriebenen Mengen gemeinsame Elemente enthalten. Somit ergibt sich folgende Definition:

$$comp(\vartheta_1, \vartheta_2) :=$$
$$\begin{cases} true, & \forall v \in dom(\vartheta_1) \cap dom(\vartheta_2) \longrightarrow terms(\vartheta_1(v)) \cap terms(\vartheta_2(v)) \neq \emptyset \\ false, & in\ allen\ anderen\ Fällen \end{cases}$$

Bei dem Verschmelzen zweier Lösungs-Template-Mappings werden, aus der gleichen Argumentation heraus wie bei der im vorherigen Abschnitt erläuterten Verschmelzung zweier PITMs, Variablen bevorzugt auf URIs und Literale abgebildet:

$$\vartheta(v) = \vartheta_1 \cup \vartheta_2(v) =$$
$$\begin{cases} \vartheta_1(v), & v \in dom(\vartheta_1) \wedge (\vartheta_1(v) \in L \cup U \vee v \notin dom(\vartheta_2) \vee \vartheta_2(v) \notin L \cup U) \\ \vartheta_2(p), & v \in dom(\vartheta_2) \wedge (\vartheta_2(v) \in L \cup U \vee v \notin dom(\vartheta_1) \vee \vartheta_1(v) \notin L \cup U) \end{cases}$$

Die Menge der Tripel-Templates des sich aus der Verschmelzung der beiden LTMs ergebenden LTM besteht aus der Vereinigung der beiden Mengen von Tripel-Templates:

$$\vartheta.TT = \vartheta_1.TT \cup \vartheta_2.TT$$

Die Mengen der OData-URIs werden, entsprechend den Erläuterungen aus Abschnitt 5.6.3, miteinander verschmolzen:

$$\vartheta.OU = merge(\vartheta_1.OU, \vartheta_2.OU)$$

5.5.5 Filter

Die Auflösung des *Filter*-Operators wird entsprechend der SPARQL-Spezifikation folgendermaßen festgelegt:

$$[\![Filter(F, P)]\!]_{DT,GT} = Filter(F, [\![P]\!]_{DT,GT})$$

Bei der Filterung einer Menge von Lösungs-Template-Mappings muss der Fall berücksichtigt werden, dass eine oder mehrere Variablen des Filter-Ausdrucks nicht auf ein Literal oder einen RDF-Term, sondern auf ein Template abgebildet werden. Sei als Beispiel folgender Ausdruck der SPARQL-Algebra gegeben: *Filter(?price > 20, BGP(?prod rdf:type northw:Product . ?prod northw:unit_price ?price))*. Diese ermittelt alle Produkte, die mehr als 20 Dollar kosten. Des Weiteren seien folgende beiden Tripel-Templates gegeben: *?product rdf:type northw:Product . ?product northw:unit_price 30.0*. Diese Tripel-Templates könnten z.B. Teil eines Graph-Templates einer Entitätsmenge sein, welche alle Produkte enthält, die 30 Dollar kosten. Nach Auflösung des einfachen Graph-Musters der SPARQL-Anfrage, entsprechend den Erläuterungen aus Abschnitt 5.5.3, ergibt sich ein Lösungs-Template-Mapping, das die *?price*-Variable auf das Literal 30.0 abbildet. In diesem Fall wird der *Filter*-Operator entsprechend der SPARQL-Spezifikation zu *true* ausgewertet. Dies hat zur Folge, dass das Lösungs-Template-Mapping Teil der Ergebnismenge wird, da es eine Lösung für die Anfrage mit Bezug auf die Tripel-Templates geben könnte. Wird hingegen eine Variable eines Filter-Ausdrucks auf ein Template abgebildet, kann der Filter-Ausdruck ggf. nicht mehr

ausgewertet werden. Es seien folgende Tripel-Templates gegeben: *?product rdf:type northw:Product . ?product northw:unit_price $UnitPrice*. Nach der Auflösung des BGP-Operators, des obigen SPARQL-Algebra-Ausdrucks, mit Hinblick auf diese Tripel-Templates, wird die *?price*-Variable auf das Literal-Template *$UnitPrice* abgebildet. Eine Auswertung des *Filter*-Operators für dieses Lösungs-Template-Mapping ist nicht möglich, da die durch das Literal-Template beschriebenen Literale nicht bekannt sind. Der Filter-Ausdruck wird daher im Sinne einer dreiwertigen Logik zu *unkown* ausgewertet. Unter der Berücksichtigung dass, wie in der SPARQL-Spezifikation angegeben, Filter-Ausdrücke in bestimmten Fällen auf den Wert *error* abgebildet werden, muss die Auswertung des logischen Und- und logischen Oder-Operators erweitert werden. Wenn der logische Oder-Operator für ein Argument den Wert *true* aufweist, wird dieser immer zu *true* ausgewertet, unabhängig davon, ob das zweite Argument einem *unkown* entspricht oder nicht. Wenn es sich bei einem der beiden Argumente um ein *unkown* handelt, und das zweite Argument keinem *true* entspricht, wird der Ausdruck zu *unkown* ausgewertet. Der logische Und-Operator wird für ein *false* Argument, unabhängig von dem zweiten Argument, immer auf *false* abgebildet. Entspricht eines der Argumente einem *unkown* und das andere Argument einem *error*, wird der Gesamtausdruck zu *error* ausgewertet. Wenn hingegen das zweite Argument einem *true* entspricht, erfolgt eine Abbildung auf *unkown*. Damit kann die Auswertung des Filter-Operators für eine Menge von Lösungs-Template-Mappings Θ und eines Filter-Ausdrucks F folgendermaßen definiert werden:

$$Filter(F, \Theta) = \{\vartheta \mid \vartheta \in \Theta \wedge [\![\vartheta(F)]\!] \in \{true, unkown\}\}$$

5.5.6 LeftJoin

Die Auswertung des *LeftJoin*-Operators wird mit Hinblick auf die SPARQL-Spezifikation folgendermaßen festgelegt:

$$[\![LeftJoin(P_1, P_2, F)]\!]_{DT,GT} = [\![P_1]\!]_{DT,GT} \underset{F}{\bowtie} [\![P_2]\!]_{DT,GT}$$

Der linke Join zweier Mengen von Lösungs-Template-Mappings muss, mit Hinblick auf die erweiterte Auflösung von Filter-Ausdrücken auf den Wert *unkown*, im Vergleich zu dem linken Join zweier Multimengen von Lösungs-Mappings angepasst werden:

$$\Theta_1 \underset{F}{\bowtie} \Theta_2 = \{\vartheta_1 \cup \vartheta_2 \mid \vartheta_1 \in \Theta_1 \wedge \vartheta_2 \in \Theta_2 \wedge$$
$$comp(\vartheta_1, \vartheta_2) \wedge [\![\vartheta_1 \cup \vartheta_2(F)]\!] \in \{true, unknown\}\} \cup$$
$$\{\vartheta_1 \mid \vartheta_1 \in \Theta_1, \forall \vartheta_2 \in \Theta_2 : ! comp(\vartheta_1, \vartheta_2) \vee [\![\vartheta_1 \cup \vartheta_2(F)]\!] \in \{false, unknown\}\}$$

5.5.7 Graph

Die Auflösung des *Graph*-Operators für Datensatz-Templates und Graph-Templates stimmt mit der in der SPARQL-Spezifikation definierten Auflösung des *Graph*-Operators für

Datensätze und Graphen überein. Wobei die IRIs den Basis-URIs der registrierten Services entsprechen müssen. Wenn die als Argument an den *Graph*-Operator übergebene IRI ein Graph-Template in dem Datensatz-Template DT identifiziert, dann gilt:

$$[\![Graph(IRI, P)]\!]_{DT,GT} = [\![P]\!]_{DT,GT_{IRI}}$$

Dabei entspricht der Ausdruck GT_{IRI} dem benannten Graph-Template GT, das durch die IRI identifiziert wird. Wenn in dem Datensatz-Template kein Graph-Template enthalten ist, welches durch die gegebene IRI identifiziert wird, gilt:

$$[\![Graph(IRI, P)]\!]_{DT,GT} = \emptyset$$

Wenn das Argument des Graph-Operators, anstatt einer URI, einer Variablen entspricht, dann ergibt sich folgende Auflösung. Wobei die Formalisierung sich an den in [84] gemachten Formalismus in Bezug auf die Auflösung des *Graph*-Operators anlehnt.

$$\begin{aligned}[\![Graph(v, P)]\!]_{DT,GT} = &\left([\![Graph(IRI_n, P)]\!]_{DT,GT} \bowtie \{\vartheta: v \longmapsto IRI_n\}\right) \cup \\ &\left([\![Graph(IRI_{n-1}, P)]\!]_{DT,GT} \bowtie \{\vartheta: v \longmapsto IRI_{n-1}\}\right) \cup \dots \\ &\left([\![Graph(IRI_1, P)]\!]_{DT,GT} \bowtie \{\vartheta: v \longmapsto IRI_1\}\right)\end{aligned}$$

Die IRIs $IRI_1, IRI_2, \dots, IRI_n$ entsprechen dabei den Basis-URIs der Services, deren Graph-Templates Bestandteil der benannten Graph-Templates des Datensatz-Templates DT sind.

5.6 Erzeugung und Verschmelzung der OData-URIs

Für jedes Lösungs-Template-Mapping, das für ein Tripel-Pattern mit Bezug auf ein Tripel-Template definiert wurde, muss eine OData-URI generiert werden. Diese dient zur Abfrage aller Daten, die benötigt werden, um die durch das Tripel-Template beschriebenen Tripel erzeugen zu können. Die Erstellung der OData-URIs für ein gegebenes Tripel-Template ist Gegenstand dieses Abschnitts.

Als einleitendes Beispiel wird das Tripel-Template *?order northw:employee_name $LastName* der *Orders*-Entitätsmenge aus Listing 68 betrachtet. Den Erläuterungen aus Abschnitt 5.3.1 folgend, hat dieses Tripel-Template folgenden Pfad: *EntitySet:Orders / EntityType:Order / NavigationProperty:Employee / EntitySet:Employees / EntityType:Employee / Property:LastName*. Demzufolge wird für jede *Orders*-Entität, die über die *Employee*-Navigationseigenschaft mit einer Entität der *Employees*-Entitätsmenge und des Entitätstyps *Employee* verknüpft ist, welche die Eigenschaft *LastName* aufweist, ein entsprechendes Tripel erzeugt. Somit müssen alle *Orders*-Entitäten und die mit diesen Entitäten verknüpften *Employees*-Entitäten mit der *LastName*-Eigenschaft abgerufen werden. Es ergibt sich folgende OData-URI: *http://services.odata.org/Northwind/Northwind.svc/ Orders ?$select=Employee/LastName &$expand=Employee*. Diese kann auf Basis der durch den Pfad gegebenen Informationen erzeugt werden. Die Service-Basis-URI

(*http://services.odata.org/Northwind/Northwind.svc*) ist über die Zugehörigkeit des Tripel-Templates zu dem Graph-Template gegeben. Die Entitätsmenge der OData-URI bildet immer das erste Element des Pfades (*Orders*). Der Entitätstyp (*Order*), welcher das zweite Pfad-Element bildet, hat in diesem Fall keinen Einfluss auf die Erzeugung der OData-URI. Die Navigationseigenschaft (*Employee*) geht in Form der *expand*-Anweisung in die URI ein. Die Information über die durch die Navigationseigenschaft referenzierte Entitätsmenge (*Employees*) und den entsprechenden Entitätstypen (*Employee*) werden von dem RDF-Adapter bei der Überführung der Rückgabedaten nach RDF benötigt. Bei der Erstellung der OData-URIs sind sie, wie weiter unten noch gezeigt wird, nur dann von Bedeutung, wenn der von der Entitätsmenge referenzierte Entitätstyp Untertypen besitzt. Das ist für die Entitätsmenge *Orders* aus Listing 68 nicht der Fall. Die Eigenschaft (*LastName*) wird Bestandteil der *select*-Anweisung. Wobei die vorhergehende Navigationseigenschaft berücksichtigt werden muss.

Neben dem Pfad des Tripel-Templates, müssen bei der Erzeugung der OData-URI auch die Pfade des Subjekt- und des Objekt-Templates berücksichtigt werden. Die Ressourcen-Variable *?order*, die das Subjekt-Template des Tripel-Template bildet, basiert auf dem Entitätstyp *Order*, welcher den Entitätsschlüssel *OrderID* aufweist. Den Erläuterungen aus Abschnitt 5.3.2 folgend, ergibt sich für den Entitätsschlüssel folgender Pfad: *EntitySet:Orders / EntityType:Order / Property:OrderID*. Ausgehend von den Erläuterungen zur Überführung des Pfades des Tripel-Templates, ergibt sich aus dieser Pfadangabe folgende OData-URI: *http://services.odata.org/Northwind/Northwind.svc/Orders?$select=OrderID*. Der Pfad des Literal-Templates (*$LastName*), welches das Objekt-Template bildet, entspricht dem Pfad des Tripel-Templates. Daher ergibt sich dieselbe OData-URI. Die drei sich aus den Pfaden ergebenden OData-URIs identifizieren alle Daten, die abgerufen werden müssen, um alle Tripel, die durch das gegebene Tripel-Template beschrieben werden, erzeugen zu können.

Um die Effizienz der Abfrage zu erhöhen, können die Abfragen zusammengefasst werden. Die mehrmalige Abfrage derselben Daten, wie sie sich aus den gleichen URIs für das Tripel-Template und das Objekt-Template ergibt, ist unnötig. Daher kann die OData-Abfrage des Literal-Templates entfallen. Des Weiteren können die OData-URIs des Tripel-Templates und des Subjekt-Templates zur folgenden URI zusammengefasst werden: *http://services.odata.org/Northwind/Northwind.svc/Orders?$select=OrderID,Employee/LastName&$expand=Employee*. Anstatt über drei Abfragen, können somit alle benötigten Daten über eine einzige Abfrage abgerufen werden.

Die Effizienz der Abfrage kann weiter erhöht werden, indem bei der Erstellung der URI das entsprechende Tripel-Pattern berücksichtigt wird. Sei mit Hinblick auf das obige Tripel-Template folgendes Tripel-Pattern gegeben: *?or northw:employee_name 'Doe'*. Aus den von der obigen URI adressierten Daten können mehr Tripel erzeugt werden, als zur Beantwortung der Anfrage benötigt werden. Neben den Bestellungen, die von dem Angestellten *Doe* bearbeitet werden, werden auch die IDs und die Namen der verantwortlichen Mitarbeiter zurückgegeben, die nicht *Doe* heißen. In diesem Fall wird bei der Kalkulation der Lösungen nur die Teilmenge der durch das Tripel-Template beschriebenen Tripel benötigt, welche als Objekt das Literal *'Doe'* aufweisen. Da die Tripel aus der *LastName*-Eigenschaft der mit den *Order*-Entitäten verknüpften *Employee*-Entitäten erzeugt werden und die entsprechende Navigationseigenschaft eine Kardinalität von 1 aufweist, kann die Abfrage mittels der *filter*-

Anweisung auf die relevanten Entitäten eingeschränkt werden. Es ergibt sich somit folgende URI: *http://services.odata.org/Northwind/Northwind.svc/Orders? $select=OrderID, Employee/LastName &$expand=Employee &$filter=Employee/LastName eq 'Doe'*.

Diesen Überlegungen entsprechend, erfolgt die Überführung eines Tripel-Templates in eine OData-URI in drei Schritten. Zunächst werden die Pfade des Tripel-Templates, des Subjekt- und des Objekt-Templates in OData-URIs überführt. Anschließend werden diese OData-URIs vereinigt. Im dritten Schritt werden die *filter*-Anweisungen, die sich aus dem Tripel-Pattern ergeben, zu der OData-URI hinzugefügt.

Zur Formalisierung dieses Vorgehens sind zunächst weitere Definitionen erforderlich. Im Folgenden werden OData-URIs durch Elemente der Menge OU repräsentiert. Dabei enthält ein $ou \in OU$ alle Informationen, die benötigt werden, um eine URI zur Adressierung einer Menge von Entitäten im Sinne der OData-Spezifikation zu erzeugen (*dataSvcAbs-URI* [137]). Die Service-Basis-URI, die aus dem Schema (*scheme*), dem Host (*host*), der Port *([":" port]*), dem Service-Root (*serviceRoot*), dem Pfad-Prefix (*pathPrefix*) besteht, ist durch $ou.r$ gegeben. Auf die Service-Basis-URI folgt als Teil des Ressourcen-Pfades (*resourcePath*) der Entitätscontainer $ou.ec \in EC$ (*entityContainer*) und die Entitätsmenge $ou.es \in ES$ (*entitySet*). Die Abfrageoptionen (*queryOptions*) *expand* (*expandQueryOp*), *filter* (*filterQueryOp*) und *select* (*selectQueryOp*) werden durch die Mengen $ou.E$, $ou.F$ und $ou.S$ repräsentiert. Dabei entspricht ein $e \in ou.E$ einer *expand*-Klausel (*expandClause*) und $s \in ou.S$ einem *select*-Item (*selectItem*) im Sinne der OData-Spezifikation. Die Elemente der Menge $ou.F$ werden in der sich ergebenden URI konjunktiv verknüpft (logisches AND), wobei ein Element $f \in ou.F$ einem boolschen Ausdruck (*boolCommonExpression*) entspricht.

Aufbauend auf diesen Definitionen, kann für zwei OData-URIs $ou_1, ou_2 \in OU$, deren Service-Basis-URI ($ou_1.r = ou_2.r$), Entitätscontainer ($ou_1.ec = ou_2.ec$) und Entitätsmenge ($ou_1.es = ou_2.es$) übereinstimmen, die Vereinigung ($ou = ou_1 \cup ou_2$) definiert werden. Es gilt: $ou.r = ou_1.r$, $ou.ec = ou_1.ec$ und $ou.es = ou_1.es$. Die Abfrageoptionen werden entsprechend den obigen Erläuterungen zusammengefasst: $ou.S = ou_1.S \cup ou_2.S$, $ou.E = ou_1.E \cup ou_2.E$ und $ou.F = ou_1.F \cup ou_2.F$.

Damit kann nun das oben beschriebene Vorgehen zur Erzeugung der OData-URIs in Algorithmus 17 formalisiert werden. Die Menge PA_{tt} nimmt alle Pfade des Tripel-Templates auf, die bei der Erstellung der OData-URI relevant sind. In Zeile 2 werden der Pfad des Tripel-Templates selbst und die Pfade der Subjekt-Ressourcen-Variable zu dieser Menge hinzugefügt. Falls es sich bei dem Objekt-Template ebenfalls um eine Ressourcen-Variable handelt (Zeile 3), werden auch diese Pfade in die Menge aufgenommen. Wenn das Objekt-Template einem Literal-Template entspricht (Zeile 5), wird der Pfad auf die Eigenschaft des Literal-Templates Element der Menge (Zeile 6). Für Funktions-Templates (Zeile 7) müssen die Pfade zu allen Eigenschaften, welche als Eingabeparameter dienen, zu der Menge hinzugefügt werden (Zeile 8). Aus jedem der ermittelten Pfade wird eine OData-URI erzeugt (Zeile 9), die, entsprechend den obigen Erläuterungen, zu einer einzigen URI zusammenfasst werden. Die Überführung von Pfaden in OData-URIs (*calcOU*-Funktion) ist Gegenstand des nächsten Abschnitts. Die Bestimmung der *filter*-Anweisung auf Basis des gegebenen Tripel-Templates erfolgt in Zeile 10. Sie wird in Abschnitt 5.6.2 im Detail behandelt.

Algorithmus 17: $ou = createODataURI(tt, tp)$

Eingabe: $tt \in TT, tp \in TP$
Ausgabe: $ou \in OU$
1. **begin**
2. $PA_{tt} \longleftarrow \{tt.pa\} \cup tt.s.KP \cup \{tt.s.cp\}$
3. **if** $tt.o \in UT \cup BT$ **then**
4. $PA_{tt} \longleftarrow PA_{tt} \cup tt.o.KP \cup \{tt.o.cp\}$
5. **if** $tt.o \in LT$ **then**
6. $PA_{tt} \longleftarrow PA_{tt} \cup \{tt.o.pp\}$
7. **if** $tt.o \in FT$ **then**
8. $PA_{tt} \longleftarrow PA_{tt} \cup \bigcup_{\forall pr \in tt.o.PR} \{pr.pp\}$
9. $ou \longleftarrow \bigcup_{\forall pa \in PA} calcOU(pa)$
10. $ou.F \longleftarrow calcFilter(tt, tp)$
11. **return** ou
12. **end**

5.6.1 Überführung von Pfaden nach OData-URIs

Wie in Abschnitt 5.3.2 beschrieben, entspricht das erste Element eines Pfades immer der Entitätsmenge, zu deren Graph-Template das entsprechende Tripel-Template gehört. Damit ist die Service-Basis-URI und der Entitätscontainer und die Entitätsmenge des Ressourcen-Pfades der OData-URI gegeben. Den Erläuterungen aus Abschnitt 5.3 entsprechend, folgt auf die Entitätsmenge eine Liste von folgende Elementen: Entitätsmengen, Entitätstypen, Eigenschaften, Navigationseigenschaften und komplexe Typen. Es ist zu definieren, wie diese Elemente auf die Abfrageoptionen der URI abgebildet werden.

Neben der Entitätsmenge, die das erste Element des Pfades bildet, können weitere Entitätsmengen in einem Pfad nur als Folge einer Navigationseigenschaft auftreten. Diese spezifiziert in diesem Fall die Entitätsmenge, der die referenzierte Entität angehören muss, um für das Template, zu dem der Pfad gehört, relevant zu sein. Da OData keine Option bereitstellt, um die referenzierten Entitäten auf bestimmte Entitätsmengen einzuschränken, haben die Entitätsmengen eines Pfades, mit Ausnahme des ersten Pfad-Elements, keinen Einfluss auf die Erstellung der OData-URI.

Ein anderer Fall ergibt sich für Entitätstypen. Ein in einem Pfad vorkommender Entitätstyp sagt aus, dass ausschließlich Entitäten genau dieses Typs abgefragt werden sollen. OData bietet mit der *isof*-Funktion die Möglichkeit, über die *filter*-Anweisung, die abzufragenden Entitäten in Abhängigkeit des Typs der Entitäten selbst oder in Abhängigkeit des Typs von über Eigenschaften oder Navigationseigenschaften referenzierter Instanzen bzw. Entitäten einzuschränken. Das zweite Element eines Pfades ist immer ein Entitätstyp, der den relevanten Typ der Entitäten der angefragten Entitätsmenge festlegt. Wenn der von der Entitätsmenge referenzierte Entitätstyp keine Untertypen besitzt, kann die Entitätsmenge keine Entitäten eines anderen Typs beinhalten. In diesem Fall ist keine zusätzliche *filter*-

Anweisung notwendig. Wenn der referenzierte Entitätstyp über Untertypen verfügt, können die in der Entitätsmenge enthaltenen Entitäten jeden dieser Entitätstypen als Typ aufweisen. Um die Rückgabemenge auf die für den Pfad relevanten Entitäten einzuschränken, wird eine entsprechende *isof*-Anweisung mit dem im Pfad spezifizierten Entitätstypen Teil der *filter*-Anweisung.

Seien als Beispiel die beiden Entitätstypen *Order* und *ExpressOrder* gegeben. Wobei der Entitätstyp *ExpressOrder* ein Untertyp von *Order* ist. Eine Entitätsmenge *Orders*, welche den Entitätstypen *Order* referenziert, kann sowohl Entitäten des Typs *Order*, als auch Entitäten des Typs *ExpressOrder* enthalten. Für den Pfad *EntitySet:Orders / EntityTyp:ExpressOrder / Property:OrderID* sind nur die Entitäten des Typs *ExpressOrder* von Bedeutung. Daher wird das Pfadelement *EntityTyp:ExpressOrder* auf folgende *filter*-Anweisung abgebildet: *isof('NorthwindModel.ExpressOrder')*.

Wenn ein Entitätstyp als Folge einer Navigationseigenschaft auftritt, welches alle Entitätstypen nach dem zweiten Pfad-Element sind, kann eine Filterung nur dann erfolgen, wenn alle vorgehenden Navigationseigenschaften eine Kardinalität von 1 oder 0..1 aufweisen. Der Grund dafür liegt darin, dass die *isof*-Funktion nicht für Kollektionen von über Navigationseigenschaften referenzierten Entitäten definiert ist. Falls alle vorhergehenden Navigationseigenschaften eine Kardinalität von 1 oder 0..1 haben und der von der Navigationseigenschaft referenzierte Entitätstyp Untertypen besitzt, wird eine entsprechende *isof*-Anweisung, unter Berücksichtigung der vorhergehenden Navigationseigenschaften, in die *filter*-Anweisung eingefügt.

Als Beispiel sei folgender Pfad gegeben: *EntitySet:Orders / EntityType:Order / NavigationProperty:Employee / EntitySet: Employees / EntityType: Manager*. Des Weiteren existiert, neben dem Entitätstyp *Manager*, der Entitätstyp *Employee*, der den Basistyp von *Manager* bildet. Wenn nun die Navigationseigenschaft *Employee* auf den Typ *Employee* verweist, können die referenzierten Entitäten von beiden Typen sein. Um mit Hinblick auf den Pfad die Menge der *Orders*-Entitäten auf diejenigen einzuschränken, welche über die Navigationseigenschaft Entitäten des Typs *Manager* referenzieren, wird folgende *isof*-Anweisung erzeugt: *isof(Employee,'NorthwindModel.Manager')*.

Für komplexe Typen innerhalb von Pfaden gilt grundsätzlich das Gleiche wie für Entitätstypen. Sofern die von den Eigenschaften referenzierten komplexen Typen Untertypen aufweisen, werden sie auf *isof*-Anweisungen abgebildet. Diese schränken die Entitätsmenge auf Entitäten ein, die über die entsprechende Eigenschaft eine Instanz des komplexen Typs referenziert.

Die OData-Spezifikation erzwingt die Unterstützung der *isof*-Funktion nicht. Des Weiteren kann eine Implementierung nur einen Teil der erlaubten Ausdrücke zur Adressierung (*commonExpression*) von Instanzen unterstützen. Es sei daher noch einmal darauf hingewiesen, dass die in dieser Arbeit vorgestellten Verfahren von einer vollständigen Umsetzung der OData-Spezifikation ausgehen.

Die Navigationseigenschaften eines Pfades beschreiben immer eine Verbindung zwischen dem Entitätstypen, der durch das zweite Element des Pfades spezifiziert ist und einem anderem Entitätstypen. Wie in Abschnitt 2.3 beschrieben, bietet OData mit der *expand*-Anweisung die Option, ausgehend von den Entitäten einer Entitätsmenge, die von diesen Entitäten über eine oder mehrere Navigationseigenschaften referenzierten Entitäten im

Rahmen einer Abfrage mit abzurufen. Die Navigationseigenschaften des Pfades werden daher in Form einer *expand*-Klausel (*expandClause* [137]) Teil der *expand*-Anweisung der OData-URI.

Ein Pfad kann beliebig viele Eigenschaften enthalten. Wobei die erste Eigenschaft immer Teil eines Entitätstyps ist. Alle weiteren Eigenschaften können nur Teil von komplexen Typen sein. Der OData-Spezifikation entsprechend, kann über die *select*-Anweisung die Rückgabemenge auf Eigenschaften eingeschränkt werden, die Teil von Entitätstypen sind. Die Beschränkung auf bestimmte Eigenschaften komplexer Typen ist hingegen nicht möglich. Daher geht nur die erste im Pfad auftretende Eigenschaft als Teil der *select*-Klausel in die OData-URI ein. Wenn der Pfad auf eine Entitätsmenge, einen Entitätstypen oder eine Navigationseigenschaft verweist, enthält dieser keine Eigenschaft. Im ersten und zweiten Fall ist nur die Existenz einer Entität der entsprechenden Entitätsmenge bzw. des entsprechenden Entitätstypen von Bedeutung. Die Eigenschaften der Entität werden nicht benötigt. Um die zu übertragende Datenmenge möglichst gering zu halten, ist es daher sinnvoll, die Rückgabemenge auf eine einzige Eigenschaft einzuschränken. Für eine Entitätsmenge wird daher über die *select*-Anweisung ein Schlüssel des von der Entitätsmenge referenzierten Entitätstyps abgerufen. Wenn der Pfad auf einen Entitätstypen verweist, wird ein Schlüssel dieses Entitätstyps Bestandteil der *select*-Anweisung. Dabei müssen die in dem Pfad enthaltenen Navigationseigenschaften bei der Erstellung der *select*-Klausel berücksichtigt werden. Wie weiter oben erläutert, gehen die Navigationseigenschaften eines Pfades über die *expand*-Anweisung in die OData-URI ein. Wenn der Pfad mit einer Navigationseigenschaft endet, werden aufgrund der fehlenden *select*-Anweisung alle Eigenschaften der referenzierten Entitäten abgerufen. Da aber nur die Existenz der Referenz relevant ist, wird zur Verringerung der zu übertragenden Datenmenge, wie in den bereits diskutierten Fällen, nur ein Schlüssel der von der Navigationseigenschaft referenzierten Entitäten abgerufen. Dazu wird der erste Schlüssel des entsprechenden Entitätstypen Teil der *select*-Anweisung, wobei dabei ebenfalls alle Navigationseigenschaften des Pfades berücksichtigt werden müssen.

Auf Basis dieser Erläuterungen spezifiziert Algorithmus 18 für einen Pfad pa die OData-URI ou. Zunächst wird in Zeile 2 die Entitätsmenge ermittelt, die dem ersten Element des Pfades entspricht. Ausgehend von der Entitätsmenge werden in den Zeilen 4 und 5 die Service-Basis-URI und der Entitätscontainer bestimmt. In den Zeilen 6 – 19 wird der Pfad, beginnend mit dem zweiten Pfad-Element, durchlaufen. Wenn das Pfad-Element eine Navigationseigenschaft ist (Zeile 8), wird es der *expand*-Klausel hinzugefügt (Zeile 9). Wenn es sich hingegen um eine Eigenschaft handelt, und diese Eigenschaft die erste Eigenschaft im Pfad ist (Zeile 12), wird diese, unter Berücksichtigung der bisher ermittelten Navigationseigenschaften, Teil der *select*-Items (Zeile 13). Dabei gibt die *isFirstP*-Funktion in Zeile 10 den Wahrheitswert *true* zurück, wenn die übergebene Eigenschaft, die im übergebenen Pfad erste Eigenschaft ist. Anderenfalls wird der Wahrheitswert *false* zurückgegeben. Des Weiteren werden alle Eigenschaften in einer Liste von Eigenschaften p_{pa} gespeichert (Zeile 11). Diese wird bei der Bestimmung der *isof*-Anweisungen für komplexe Typen in den Zeilen 14 und 15 benötigt. In Zeile 14 wird zunächst geprüft, ob das aktuelle Pfad-Element ein komplexer Typ ist. Wenn dies zutrifft, und der von der zugehörigen Eigenschaft referenzierte komplexe Typ Untertypen besitzt, wird in Zeile 15 die

entsprechende *isof*-Anweisung erzeugt. Dabei gibt in Zeile 14 die Funktion *getLast* das letzte Element der List p_{pa} zurück, welches der zuletzt betrachteten Eigenschaft im Pfad entspricht.

Algorithmus 18: $ou = calcOU(pa)$

Eingabe: $pa \in PA$
Ausgabe: $ou \in OU$
1. **begin**
2. $i \longleftarrow popFirstElement(pa)$
3. $ou.es \longleftarrow i$
4. $ou.r \longleftarrow getBaseURI(i)$
5. $ou.ec \longleftarrow getEntityContainer(i)$
6. **while** $pa \neq \emptyset$ **do**
7. $i \longleftarrow popFirstElement(pa)$
8. **if** $i \in N$ **then**
9. $e \longleftarrow e \oplus i$
10. **if** $i \in P$ **then**
11. $p_{pa} \longleftarrow p_{pa} \oplus i$
12. **if** $isFirstP(i, pa)$ **then**
13. $s \longleftarrow e \oplus i$
14. **if** $i \in C \wedge |getLast(p_{pa}).C| > 0$ **then**
15. $ou.F \longleftarrow ou.F \cup \{'isof('p_{pa}, i')'\}$
16. **if** $i \in E$ **then**
17. **if** $e = null \wedge |ou.es.t.C| > 0$ **then**
18. $ou.F \longleftarrow ou.F \cup \{'isof('i')'\}$
19. **if** $e \neq null \wedge isFilterable(e) \wedge |getLast(e).t.C| > 0$ **then**
20. $ou.F \longleftarrow ou.F \cup \{'isof('e, i')'\}$
21. **if** $i \in ES \cup N$ **then**
22. $i \longleftarrow i.t$
23. **if** $i \in E$ **then**
24. $s \longleftarrow e \oplus getFirstKey(i)$
25. $ou.E \longleftarrow \{e\}$
26. $ou.S \longleftarrow \{s\}$
27. **return** ou
28. **end**

In den Zeilen 16 bis 20 erfolgt die Behandlung der Entitätstypen. Wenn es sich bei dem aktuell betrachteten Entitätstypen um den ersten Entitätstypen im Pfad handelt, und der von der Basis-Entitätsmenge referenzierte Entitätstyp Untertypen besitzt (Zeile 17), wird in Zeile 18 eine entsprechende *isof*-Anweisung zur *filter*-Anweisung hinzugefügt. Für alle weiteren Entitätstypen wird in Zeile 19 zunächst geprüft, ob eine Filterung möglich ist. Dazu gibt die *isFilterable*-Funktion den Wahrheitswert *true* zurück, wenn alle bisher betrachteten Navigationseigenschaften eine Kardinalität von 1 oder 0..1 aufweisen. Anderenfalls wird *false*

zurückgegeben. Wenn nach dem aktuellen Entitätstypen gefiltert werden kann und es Entitätstypen gibt, die von dem von der Navigationseigenschaft referenzierten Entitätstypen erben, wird in Zeile 20, entsprechend den obigen Erläuterungen, eine *isof*-Anweisung erzeugt. Wenn das letzte Element des Pfades eine Entitätsmenge oder eine Navigationseigenschaft ist (Zeile 21), wird der entsprechende Entitätstyp ermittelt (Zeile 24). Falls in Zeile 22 ein Entitätstyp bestimmt wurde, oder wenn das letzte Pfad-Element selbst ein Entitätstyp ist (Zeile 23), wird in Zeile 24 ein *select*-Item zur Abfrage des ersten Schlüssels des Entitätstyps erzeugt.

5.6.2 Die Erzeugung von Filter-Anweisungen

Wie bereits erläutert, werden URIs erzeugt, die hinsichtlich der zu übertragenden Datenmenge nicht optimal sind, wenn bei der Erzeugung der OData-URIs nur die jeweiligen Tripel-Templates berücksichtigt werden. Zur Optimierung der zu übertragenden Datenmenge können, wie an einem Beispiel zu Beginn des Abschnitts 5.6 gezeigt, auf Basis der Tripel-Patterns *filter*-Anweisungen erstellt werden. Die Erzeugung dieser *filter*-Anweisungen ist Gegenstand dieses Abschnitts.

Bei der Erzeugung der *filter*-Anweisung müssen mehrere Fälle unterschieden werden. Zunächst wird der Fall betrachtet, dass das Tripel-Pattern ein Literal als Objekt aufweist. Entsprechend den Erläuterungen aus Abschnitt 5.4.1, kann es für ein Tripel-Pattern mit Literal-Objekt nur dann ein passendes Tripel-Template geben, wenn das Objekt-Template einem Literal, einem Literal-Template oder einem Funktions-Template entspricht. Wobei die Sprachangaben bzw. die Datentypen übereinstimmen müssen. Wenn es sich bei dem Objekt-Template um ein Literal handelt, ist keine Filterung möglich, da das Objekt der zu erzeugenden Tripel unabhängig von den Werten bestimmter Eigenschaften ist.

Ein Funktions-Template kann Eigenschaften als Eingabeparameter aufweisen, nach denen gefiltert werden kann. Als Beispiel sei das bereits aus Abschnitt 4.6 bekannte Tripel-Template *?employee northw:name fn:concat($FirstName,' ',$LastName)* gegeben. Dieses Tripel-Template weist ein Funktions-Template mit drei Eingabeparametern als Objekt auf. Dabei entsprechen zwei der Eingabeparametern Referenzen auf Eigenschaften. Bei dem Tripel-Pattern *?emp northw:name 'John Doe'* handelt es sich um ein nach Algorithmus 16 dazu *passendes* Tripel-Pattern. Wobei für eine Lösung *'John Doe'=fn:concat($FirstName,' ',$LastName)* gelten muss. Wie bereits in Abschnitt 2.3 erwähnt, stellt OData eine Vielzahl logischer und arithmetischer Operatoren, sowie verschiedene Funktionen zur Verfügung, die im Rahmen der *filter*-Anweisung verwendet werden können. Als Teil dieser Funktionen wird mit *concat* eine Funktion bereitgestellt, welche die gleiche Funktionalität wie die *fn:concat*-Funktion mit zwei Argumenten aus dem XPath-Namensraum anbietet. Für die *fn:concat*-Funktion mit drei Argumenten, wie sie in dem dargestellten Funktions-Template verwendet wird, kann somit ein Mapping definiert werden. Die *fn:concat*-Funktion kann für die drei Parameter *$Firstname*, *' '* und *$LastName* nur dann den Wert *'John Doe'* zurückgeben, wenn die Rückgabe des OData-Funktionsausdruck *concat(FirstName,concat(' ',LastName))* als Teil der *filter*-Anweisung für die Werte der beiden Eigenschaften dem Wert *'John Doe'* entspricht. Es würde sich folgende *filter*-Anweisung ergeben: *filter=concat(FirstName,concat('*

',LastName)) eq 'John Doe'. Viele der XPath-/XQuery-Funktionen und Operatoren können auf OData-Operatoren und Funktionen abgebildet werden. Tabelle 6 zeigt einige Beispiele.

XPath/XQuery	OData
op:numeric-add($arg1,$arg2)	$arg1 add $arg2
op:numeric-subtract($arg1,$arg2)	$arg1 sub $arg2
op:numeric-less-than($arg1,$arg2)	$arg1 lt $arg2
op:numeric-greater-than($arg1,$arg2)	$arg1 gt $arg2
fn:concat($arg1,$arg2,…)	concat($arg1,concat($arg2,...))
fn:year-from-dateTime($arg)	year($arg)

Tabelle 6: Abbildung von XPath-/XQuery-Funktionen und Operatoren auf OData-Funktionen und Operatoren.

Bei der Überführung der XPath-/Query-Funktionen und Operatoren in *filter*-Ausdrücke müssen gegebenenfalls vorhandene Unterschiede in den Datentypen berücksichtigt werden. Wenn der Datentyp des Literals nicht dem Rückgabetyp der XPath-/Query-Funktion entspricht, muss der Wert entsprechend konvertiert werden. Anschließend kann für die *filter*-Anweisung eine Überführung der von XPath und XQuery verwendeten XSD-Datentypen auf die EDM-Datentypen, entsprechend Tabelle 5, erfolgen. Es sei angemerkt, dass dieses Verfahren nur dann zu vollständigen Ergebnissen führt, wenn eine verlustfreie Konvertierung zwischen den Datentypen möglich ist. Des Weiteren ist eine Filterung nur möglich, wenn die Pfade der Literal-Templates, die als Eingabeparameter für die Funktion dienen, filterbar sind. Dass bedeutet, dass die im Pfad enthaltenen Navigationseigenschaften eine Kardinalität von 1 oder 0..1 aufweisen müssen. Wie in Abschnitt 4.6 erwähnt, stellen die XQuery- und XPath-Funktionen nur ein Beispiel für eine Menge von Funktionen dar, die von einer konkreten Implementierung zur Verfügung gestellt werden können. Für eine Funktion außerhalb dieses Namensraums kann die vorgestellte Vorgehensweise ebenfalls verwendet werden, falls die Definition einer Abbildung auf die existierenden OData-Funktionen und Operatoren möglich ist. Wenn dies nicht der Fall ist, besteht für bijektive Funktionen die Möglichkeit, die Umkehrfunktion anzugeben. Der relevante Wert der Eigenschaft kann dann über die Umkehrfunktion berechnet und als Gleichheitsabfrage in die *filter*-Anweisung eingefügt werden. Wenn weder eine Abbildung auf die OData-Funktionen und Operatoren definiert wurde, noch eine Umkehrfunktion existiert, oder die Filterung nach der Eigenschaft entsprechend des Pfades nicht möglich ist, wird für ein Funktions-Template bei einem gegebenen Literal keine *filter*-Anweisung erzeugt.

Bei einem Literal-Template gestaltet sich die Erzeugung der zugehörigen *filter*-Anweisung einfacher als bei einem Funktions-Template. Wenn für ein Literal-Template über das Tripel-Pattern ein Literal gegeben ist, wird zunächst durch Betrachtung des Pfades geprüft, ob nach der referenzierten Eigenschaft gefiltert werden kann. Falls dies möglich ist, und auch eine bijektive Abbildung zwischen den Datentypen gewährleistet ist, kann die Eigenschaft nach dem gegebenen Wert gefiltert werden. Sei als Beispiel das Tripel-Template *?employee northw:last_name $LastName* gegeben. Für das Tripel-Pattern *?emp northw:last_name 'Doe'* wird die Menge der abzufragenden Entitäten über folgende *filter*-Anweisung eingeschränkt: $*filter=LastName eq 'Doe'*.

Neben dem Fall, dass ein Tripel-Pattern ein Literal als Objekt aufweist, müssen auch Tripel-Patterns mit einer IRI als Subjekt oder Objekt bei der Erzeugung der *filter*-Anweisungen berücksichtigt werden. Sofern die Subjekt-IRI dem in Abschnitt 4.3 beschriebenen Format entspricht, ist die Entität, auf der die Ressource basiert, eindeutig spezifiziert. Es besteht nun entweder die Möglichkeit, die URI der Entität direkt aus der gegebenen IRI zu erzeugen oder die Entitätsmenge nach der Entität mit dem gegebenen Schlüssel zu filtern. Um die weiter unten erläuterte Verschmelzung zweier OData-URIs zu vereinfachen, wird der zweite Ansatz gewählt. Als Beispiel sei folgendes Tripel-Pattern gegeben: *<http://northwind.beispiel.org/23523/employee#10249> northw:last_name ?lastName*. Dieses Tripel-Pattern fragt nach dem Nachnamen des Angestellten, der durch die Subjekt-IRI identifiziert wird. Der Service, der Entitätscontainer und die Entitätsmenge, in der die Entität enthalten ist, auf welche die Ressource basiert, sind durch die Entity-Location-ID (*23523*) gegeben. Durch eine Filterung nach dem gegebenen Entitätsschlüssel (*filter=EmployeeID eq 10249*) kann die Rückgabemenge auf die gewünschte Entität eingeschränkt werden. Für eine IRI als Objekt eines Tripel-Patterns erfolgt die Erzeugung der *filter*-Anweisung auf die gleiche Art und Weise. Wobei eine Filterung nur dann möglich ist, wenn, wie weiter oben bereits mehrfach erläutert, die Kardinalitäten der Navigationseigenschaften des entsprechenden Objekt-URI-Templates 1 oder 0..1 betragen.

Auf Basis der vorhergehenden Erläuterungen formalisiert Algorithmus 19 die Bestimmung des Filters für die OData-URI, die aus einem gegebenen Tripel-Template mit Hinblick auf das spezifizierte Tripel-Pattern erzeugt wird. In Zeile 2 wird geprüft, ob das Tripel-Pattern für das Subjekt-Template eine IRI bereitstellt. Wenn dies der Fall ist, wird in den Zeilen 3 – 4 für jeden Schlüssel eine entsprechende *filter*-Anweisung erzeugt. Dabei generiert die *getCommonExpr*-Funktion aus dem übergebenen Pfad die als Teil der *filter*-Anweisung benötigte Angabe zur Adressierung der entsprechenden Eigenschaft (*commonExpression* [137]). Für die Schlüssel des Subjekt-URI-Templates entspricht diese Angabe immer der Eigenschaft, welche durch das letzte Element des Pfades gegeben ist. Wenn der Pfad Navigationseigenschaften enthält, werden diese der Eigenschaft vorangestellt. Die *valueOf*-Funktion in Zeile 4 gibt für den durch den Pfad spezifizierten Schlüssel den aus der IRI des Tripel-Patterns extrahierten Wert dieses Schlüssels in dem Datentyp des Schlüssels zurück. Wenn das Objekt des Tripel-Patterns einem Literal entspricht, und das Tripel-Template ein Literal-Template als Objekt aufweist (Zeile 5), wird in Zeile 6 die zugehörige *filter*-Anweisung erzeugt. Wobei die *convert*-Funktion das Literal in den Datentyp der Eigenschaft überführt, die von dem Literal-Template referenziert wird. Für ein Objekt-URI-Template werden, sofern das Tripel-Pattern eine IRI als Objekt spezifiziert, in den Zeilen 7 – 9 die *filter*-Anweisungen bzgl. der entsprechenden Schlüssel erzeugt. Die Behandlung der Funktions-Templates erfolgt in den Zeilen 10 – 18. Zunächst wird in Zeile 11 mittels der *hasMapping*-Funktion geprüft, ob eine Abbildung des Funktions-Templates auf OData-Funktionen existiert. Wenn, wie beispielhaft an Tabelle 6 dargestellt, eine Abbildung existiert, wird die zugehörige Funktion in Zeile 12 abgerufen und in Zeile 13 Teil einer *filter*-Anweisung. Falls keine Abbildung definiert wurde, wird in den Zeilen 14 – 18 geprüft, ob in Abhängigkeit der Eingabeparameter Umkehrfunktionen (*hasInverse*-Funktion) existieren. Wenn eine Funktion gefunden wurde, wird diese in Zeile 17 mittels der *getInverse*-Funktion abgerufen und in Zeile 18 dem Filter hinzugefügt.

Algorithmus 19: $F = calcFilter(tt, tp)$

Eingabe: $tt \in TT, tt \in TP$

Ausgabe: F Menge von Filter-Anweisungen.

1. **begin**
2. **if** $tt.s \in UT \wedge tp.s \in I$ **then**
3. **foreach** $kp \in tt.s.KP$
4. $F \leftarrow F \cup \{getCommonExpr(kp)\ 'eq'\ valueOf(kp, tp.s)\}$
5. **if** $tt.o \in LT \wedge tp.o \in L \wedge isFilterable(tt.o.pp)$ **then**
6. $F \leftarrow F \cup \{getCommonExpr(tt.o.pp)\ 'eq'\ convert(tp.o, dt(tt.o.p))\}$
7. **if** $tt.o \in UT \wedge tp.o \in I \wedge isFilterable(tt.o.KP)$ **then**
8. **foreach** $kp \in tt.o.KP$
9. $F \leftarrow F \cup \{getCommonExpr(kp)\ 'eq'\ valueOf(kp, tp.o)\}$
10. **if** $tt.o \in FT \wedge tp.o \in L$ **then**
11. **if** $hasMapping(tt.o) \wedge isFilterable(tt.o.PR)$ **then**
12. $f \leftarrow getMapping(tt.o)$
13. $F \leftarrow F \cup \{f\ 'eq'\ convert(tp.o, dt(f))\}$
14. **else then**
15. **foreach** $pr \in tt.o.PR$
16. **if** $hasInverse(tt.o, pr) \wedge isFilterable(pr.pa)$ **then**
17. $f^{-1} \leftarrow getInverse(tt.o, pr)$
18. $F \leftarrow F \cup \{pr\ 'eq'\ convert(f^{-1}(tp.o), dt(pr.p))\}$
19. **return** F
20. **end**

5.6.3 Verschmelzen von OData-URIs

Aus Performancegründen ist es sinnvoll, mehrere OData-Anfragen durch das Verschmelzen der entsprechenden URIs zusammenzufassen. Dazu ist zunächst zu klären, wann zwei OData-URIs zusammengefasst werden können. Dabei wird nur die Untermenge an möglichen URIs betrachtet, die entsprechend den Erläuterungen aus Abschnitt 5.6.1 erzeugt werden. Insbesondere enthalten diese URIs immer eine *select*-Anweisung, wobei die *select*-Klausel niemals das *-Zeichen enthält. Des Weiteren können neben der *select*-Anweisung nur die Abfrageoptionen *expand* und *filter* auftreten. Entsprechend den Erläuterungen aus Abschnitt 5.6.1 werden aus Pfaden immer OData-URIs erzeugt, deren Service-Root-URI zusammen mit dem Ressourcen-Pfad eine Entitätsmenge adressieren. Zwei OData-URIs können nur dann zusammengefasst werden, wenn sie dieselbe Entitätsmenge anfragen. Anders formuliert, ist eine Verschmelzung nur dann möglich, wenn sie hinsichtlich der Service-Root-URI und dem Ressourcen-Pfad übereinstimmen.

Bezüglich der Auswirkung von Abfrageoptionen wird zunächst ein spezieller Fall betrachtet, wie er im Rahmen der Auflösung von einfachen Graph-Mustern auftreten kann. Wie weiter unten noch genauer erläutert wird, ist das Zusammenfassen von URIs mit *filter*-Anweisung oftmals nicht möglich, ohne die zu übertragenden Datenmenge zu erhöhen. Der

Grund dafür liegt darin, dass zwei URIs mit *filter*-Anweisung zwei unterschiedliche Mengen von Entitäten beschreiben können. Dies kann bei einer Vereinigung dazu führen, dass Eigenschaften von Entitäten abgerufen werden, die in der Vereinigung der durch die ursprünglichen URIs adressierten Datenmengen nicht enthalten sind. Falls URIs zusammengeführt werden sollen, die im Rahmen der Auflösung von Tripel-Patterns eines einfachen Graph-Musters erzeugt wurden (siehe Abschnitt 5.5.3), ergibt sich eine andere Situation, wenn die Tripel-Patterns das gleiche Subjekt aufweisen. Seien als Beispiel die beiden Tripel-Pattern *?emp northw:age 30* und *?emp northw:city "Dresden"* gegeben. Auf Basis der beiden Tripel-Templates *?employee northw:age $Age* und *?employee northw:city $City* werden die beiden URIs *Employees?$select=Age&$filter=Age eq 30* und *Employees?$select=City&$filter=City eq 'Dresden'* erstellt. Eine Lösung muss die *?emp*-Variable der beiden Tripel-Patterns auf die gleichen Ressourcen-URIs abbilden. Die Ressourcen-URIs, welche aus den beschriebenen Tripel-Templates erzeugt werden können, werden durch das URI-Template *?employee* beschrieben. Für beide Tripel-Templates können die Subjekte der aus ihnen erzeugten Tripel, nur dann übereinstimmen, wenn sie aus den gleichen Entitäten erzeugt wurden. Dass bedeutet, dass es eine Lösung nur für Ressourcen geben kann, die auf Basis von Entitäten generiert wurden, die in der Schnittmenge der beiden durch die URIs beschriebenen Entitätsmengen enthalten sind. Die Schnittmenge der Entitäten ergibt sich, indem die *filter*-Anweisungen mittels logischem Und miteinander verknüpft werden. Daher können beide URIs zur folgenden URI zusammengefasst werden: *Employees?$select=Age,City&$filter=Age eq 30 and City eq 'Dresden'*. Wobei die *select*-Ausdrücke vereinigt wurden.

Für den allgemeinen Fall müssen, mit Hinblick auf die Abfrageoptionen, die beiden Fälle unterschieden werden, dass die URI eine *filter*-Anweisung enthält oder nicht. Zunächst wird der letzte Fall betrachtet. Da beide URIs die Menge der Eigenschaften über die *select*-Anweisung beschränken, muss die Rückgabemenge der resultierenden Anfrage sowohl die Eigenschaften der ersten URI als auch die Eigenschaften der zweiten URI enthalten. Folglich setzt sich der Wert der *select*-Anweisung der resultierenden URI aus den durch ein Komma separierten Werten der *select*-Anweisungen der beiden Ausgangs-URIs zusammen. Das gleiche gilt für die *expand*-Anweisung. Wenn eine oder beide URIs referenzierte Entitäten über Navigationseigenschaften abrufen, dann muss auch die sich ergebende URI diese referenzierten Entitäten abrufen. Die *expand*-Anweisung der sich ergebenden URI erhält somit als Wert die konkatenierten Werte der *expand*-Anweisungen der Ausgangs-URIs.

Wenn eine oder beide URIs *filter*-Anweisungen enthalten, müssen diese bei der Zusammenführung der URIs berücksichtigt werden. Eine *filter*-Anweisung schränkt die Menge der zurückgegebenen Entitäten der Entitätsmenge ein. Falls eine der beiden URIs eine *filter*-Anweisung enthält und die andere nicht, bedeutet dies, dass die Entitäten, die durch die URI mit *filter*-Anweisung spezifiziert sind, auch in der Menge der Entitäten, welche durch die URI ohne *filter*-Anweisung abgefragt werden können, enthalten sind. Folglich könnten die beiden URIs zusammengefasst werden. Wobei die *filter*-Anweisung entfällt und die *select*- und die *expand*-Anweisung, entsprechend den obigen Erläuterungen, zusammengefasst werden. Der Nachteil dabei ist, dass es in bestimmten Fällen zur Übertragung nicht benötigter Daten kommen kann. Seien als Beispiel die folgenden beiden Teil-URIs gegeben (es sind jeweils der Ressourcen-Pfad und die Abfrageoptionen dargestellt): *Customers?$select=Fax*

und *Customers?$select=CompanyName&$filter=Country eq 'Germany'*. Die erste URI fragt die Fax-Nummern aller Kunden ab. Die zweite URI fragt nach den Firmennamen aller Kunden, die in Deutschland angesiedelt sind. Da die Rückgabemenge der ersten Anfrage auch alle Kunden enthält, die in Deutschland angesiedelt sind, könnten beide Anfragen folgendermaßen zusammengefasst werden: *Customers?$select=Fax,CompanyName*. Der Nachteil besteht darin, dass diese Anfrage auch die Firmennamen aller Kunden zurückgibt, die nicht in Deutschland beheimatet sind. Somit ist die Rückgabemenge der sich ergebenden URI eine Obermenge der Vereinigung der Rückgabemengen der beiden Ausgangs-URIs. Um die Übertragung nicht benötigter Daten zu verhindern, werden URIs mit *filter*-Anweisungen nur dann zusammengefasst, wenn die Menge der über die *select*-Anweisung spezifizierten Eigenschaften der URI mit *filter*-Anweisung eine Teilmenge der Eigenschaften-Menge der URI ohne *filter*-Anweisung ist. Eine Ausnahme ergibt sich, wenn die *select*-Anweisungen übereinstimmen. Für zwei URIs mit *filter*-Anweisung müssen die über *select* spezifizierten Mengen ebenfalls übereinstimmen. Eine andere Situation ergibt sich, wenn die *filter*-Anweisungen der beiden URIs einander entsprechen. In diesem Fall beschreiben beide URIs dieselben Entitätsmengen. Somit können die URIs unabhängig davon zusammengefasst werden, ob die *select*-Anweisungen übereinstimmen oder nicht. Bzgl. der *filter*-Anweisungen gilt: Wenn nur eine der beiden URIs eine *filter*-Anweisung enthält, entfällt die *filter*-Anweisung in der sich ergebenden URI. Falls beide URIs *filter*-Anweisungen aufweisen, werden diese über eine logische OR-Verknüpfung (*Or*) zusammengefasst. Für zwei übereinstimmende *filter*-Anweisungen entspricht die *filter*-Anweisung der sich ergebenden URI der gemeinsamen *filter*-Anweisung der Ausgangs-URIs.

Basierend auf diesen Überlegungen, formalisiert Algorithmus 20 die Verschmelzung zweier URIs.

Algorithmus 20: $ou = mergeURIs(ou_1, ou_2)$

Eingabe: $ou_1, ou_2 \in OU$

Ausgabe: $ou \in OU$ oder $null$ wenn eine Verschmelzung nicht möglich ist.

1. **begin**
2. **if** $ou_1.r \neq ou_2.r \vee ou_1.ec \neq ou_2.ec \vee ou_1.es \neq ou_2.es$ **then**
3. **return** $null$
4. **if** $!\big((ou_1.F \neq \emptyset \rightarrow ou_1.S \subseteq ou_2.S) \wedge (ou_2.F \neq \emptyset \rightarrow ou_2.S \subseteq ou_1.S)\big)$
5. $\wedge\, ou_1.F \neq ou_2.F$ **then**
6. **return** $null$
7. **if** $ou_1.F \neq \emptyset \wedge ou_2.F \neq \emptyset$ **then**
8. **if** $ou_1.F \neq ou_2.F$ **then**
9. $ou.F \longleftarrow \{\big(and_{\forall f_1 \in ou_1.F} f_1\big)\ or\ \big(and_{\forall f_2 \in ou_2.F} f_2\big)\}$
10. **else then**
11. $ou.F \longleftarrow ou_1.F$
12. $ou.S \longleftarrow ou_1.S \cup ou_2.S$
13. $ou.E \longleftarrow ou_1.E \cup ou_1.E$
14. **return** ou
15. **end**

In Zeile 2 wird geprüft, ob beide URIs hinsichtlich der Service-Basis-URI, dem Entitätscontainer und der Entitätsmenge übereinstimmen. Anschließend werden die *filter*- und *select*-Anweisungen, entsprechend den soeben gemachten Erläuterungen, in den Zeilen 4 und 5 miteinander verglichen. Wenn beide URIs *filter*-Anweisungen enthalten (Zeile 7), werden diese, wenn sie sich unterscheiden, wie in Zeile 9 dargestellt, konjunktiv und disjunktiv miteinander verknüpft. Für übereinstimmende *filter*-Anweisungen wird der sich ergebenden *filter*-Anweisung in Zeile 11 diese gemeinsame *filter*-Anweisung zugewiesen. Die Vereinigung der *select*- und *expand*-Anweisungen erfolgt in den Zeilen 12 und 13.

Aufbauend auf diesem Algorithmus, kann die Verschmelzung zweier Mengen von OData-URIs definiert werden, wie sie bei der Auflösung von SPARQL-Algebra-Ausdrücken, entsprechend der in Abschnitt 5.5 definierten Evaluierungssemantik, benötigt wird. Das Verfahren ist durch Algorithmus 21 formalisiert. Für jede OData-URI der ersten Menge (Zeile 2) wird geprüft, ob in der zweiten Menge eine URI existiert, die mit dieser URI verschmolzen werden kann (Zeile 4). Wenn dies der Fall ist, werden die URIs entsprechend Algorithmus 20 vereinigt (Zeile 5) und die URI aus der zweiten Menge entfernt (Zeile 6). Dieser Vorgang wird mit der sich ergebenden URI solange wiederholt, bis in der zweiten Menge von OData-URIs keine URI mehr enthalten ist, mit der eine weitere Verschmelzung möglich ist. Anschließend wird die sich aus den Verschmelzungen ergebende URI der zweiten Menge hinzugefügt (Zeile 7).

Algorithmus 21: $OU = merge(OU_1, OU_2)$

Eingabe: OU_1, OU_2 Zu verschmelzende Mengen von OData-URIs.
Ausgabe: OU Die sich ergebende Menge an OData-URIs.

1. **begin**
2. **foreach** $ou_1 \in OU_1$
3. $ou_m \longleftarrow ou_1$
4. **while** $OU_2 \neq \emptyset \wedge \exists ou_c \in OU_2 \longrightarrow mergeURIs(ou_m, ou_c) \neq null$ **then**
5. $ou_m \longleftarrow mergeURIs(ou_m, ou_c)$
6. $OU_2 \longleftarrow OU_2 \backslash \{ou_c\}$
7. $OU_2 \longleftarrow OU_2 \cup \{ou_m\}$
8. **return** OU_2
9. **end**

Teil III

Evaluierung

6 Implementierung und Evaluierung

Die in den vorherigen Abschnitten vorgestellten Verfahren wurden, entsprechend der Architektur in Abbildung 2, implementiert. Wie in Kapitel 5 dargestellt, basiert das Verfahren zur Ermittlung der relevanten OData-Services für eine gegebene SPARQL-Anfrage auf einer angepassten SPARQL-Algebra Evaluierungssemantik. Anstatt gegen RDF-Datensätze werden die Anfragen gegen Datensatz-Templates aufgelöst. Eine Implementierung, welche diese Evaluierungssemantik realisiert, ist daher mit einem *normalen* SPARQL-Prozessor vergleichbar, welcher Ausdrücke der SPARQL-Algebra gegen RDF-Datensätze auflöst. Die SPARQL-Algebra Evaluierungssemantik, wie sie durch die SPARQL-Spezifikation definiert ist, macht keine Aussagen darüber, wie ein konkreter SPARQL-Prozessor implementiert werden sollte. Vielmehr gibt es zahlreiche Arten diese umzusetzen (siehe z.B. [33, 143, 194]). Die unterschiedlichen Vorgehensweisen haben zu einer Vielzahl verschiedener SPARQL-Prozessoren geführt, die sich hinsichtlich der Performance teilweise erheblich unterscheiden [142].

So wie es mehrere Möglichkeiten gibt, einen SPARQL-Prozessor mit Bezug auf die Evaluierungssemantik der SPARQL-Spezifikation zu realisieren, sind auch mehrere Vorgehensweisen zur Realisierung eines SPARQL-Prozessors, mit Hinblick auf die im letzten Kapitel vorgestellte Evaluierungssemantik zur Ermittlung aller relevanten Tripel-Templates, vorstellbar. Diese würden sich wahrscheinlich in Abhängigkeit der Größe und Struktur der Datensatz-Templates und der SPARQL-Anfragen hinsichtlich der Performance ähnlich stark unterscheiden, wie die aktuell verfügbaren SPARQL-Prozessoren.

Das Ziel der im Rahmen dieser Arbeit erstellten prototypischen Implementierung ist es nicht, mit Hinblick auf möglichst große Datenmengen oder bestimmte SPARQL-Anfragen bezüglich der Performance, mit bestehenden SPARQL-Prozessoren in Konkurrenz zu treten. Vielmehr realisiert diese Implementierung eine möglichst direkte Umsetzung der Evaluierungssemantik, wie sie im vorherigen Kapitel beschrieben ist. Mit Ausnahme verschiedener Indexstrukturen, werden dabei keine Optimierungstechniken berücksichtigt, wie sie z.B. in [166] beschrieben werden. Der Grund dafür ist, dass so unmittelbar die Realisierbarkeit des vorgestellten Verfahrens gezeigt werden kann, ohne durch optimierungsspezifische Implementierungsdetails verdeckt zu werden. Des Weiteren ist, wie weiter unten noch gezeigt wird, die Performance für die im Rahmen des ERP-Szenarios relevante Anzahl an Services bzw. die sich daraus ergebende Größe des Datensatz-Templates ausreichend.

Die Realisierung eines SPARQL-Prozessors zur möglichst performanten Verarbeitung großer Datensatz-Templates ist Bestandteil zukünftiger Arbeiten. Das Ziel dieser Evaluierung besteht daher darin zu zeigen, dass es möglich ist, das in dieser Arbeit vorgestellte Verfahren derart umzusetzen, dass Anfragen in für praktische Anwendungsfälle akzeptablen Zeiten ausgeführt werden können. Da die im Rahmen dieser Arbeit entwickelte Implementierung das erste System zur Ausführung von SPARQL-Anfragen gegen OData-Schnittstellen ist, besteht keine Möglichkeit, die erreichte Performance mit Hinblick auf andere Systeme zu vergleichen. Es soll aber gezeigt werden, inwieweit sich die Ermittlung der relevanten OData-URIs, die Abfrage der benötigten Daten und die Überführung der Daten nach RDF, als

Overhead auf die Gesamtzeit auswirkt. Für den Fall, dass im Rahmen zukünftiger Arbeiten weitere Implementierungen erstellt werden, wären auch Vergleichstests vorstellbar, wie sie aktuell für die verschiedenen SPARQL-Prozessoren durchgeführt werden.

Der Prototyp wurde mittels Java und dem Java Development Kit 7 [4] implementiert. Zur Verarbeitung der semantischen Daten wurde Apache Jena [5] in Version 2.10.0 verwendet. Es bietet sowohl einen Triple Store als auch einen SPARQL-Prozessor (*ARQ*) [17]. Des Weiteren wird Jena von der Query Engine (siehe Abbildung 2) zur Überführung von SPARQL-Anfragen in Ausdrücke der SPARQL-Algebra eingesetzt. Die Erzeugung der Tripel durch den RDF-Adapter erfolgt ebenfalls mittels Jena. OData-Services werden von der Execution Engine mit odata4j [6] angefragt. Dabei wird ein auf odata4j 0.7 basierender Snapshot verwendet. Das Parsen der semantisch erweiterten Service-Metadaten-Dokumente bei der Registrierung der Services in der Service Registry erfolgt ebenfalls mit Hilfe von odata4j. Dabei ist zu beachten, dass odata4j sich in der verwendeten Version in einem frühen Zustand befindet und nicht die vollständige CSDL-Spezifikation umgesetzt ist. Beispielsweise werden keine Hierarchien komplexer Typen unterstützt. Zum Parsen und Verarbeiten der XQuery und XPath Funktionen und Operatoren, welche als Funktions-Template in einem Tripel-Template auftreten können, wird Xalan [7] verwendet. Xalan unterstützt in Version 2.7.1 nicht alle in [131] spezifizierten Funktionen und Operatoren. Die zu Testzwecken benötigten Funktionen sind jedoch abgedeckt.

6.1 Northwind-Service SPARQL-Endpunkt

Im Folgenden wird die Zeit untersucht, die zur Abarbeitung einer Anfrage benötigt wird. Dabei wird die Gesamtzeit entsprechend der von den einzelnen Komponenten benötigten Zeiten aufgeschlüsselt. Die Untersuchung erfolgt mittels des öffentlich verfügbaren Northwind-Services [136], welcher bereits zu Beginn von Kapitel 4 vorgestellt und im Rahmen der Arbeit durchgehend verwendet wurde. Der für die Anfragen relevante Ausschnitt aus dem semantisch erweiterten Service-Metadaten-Dokument [135] ist in Anhang C zu finden. Als Test-Abfragen kommen die zwölf in Anhang D dargestellten SPARQL-Anfragen zum Einsatz. Für jede SPARQL-Anfrage sind die von der Query Engine ermittelten OData-URIs dargestellt. Des Weiteren sind die Größe der übertragenen Datenmenge und die Anzahl der aus der Datenmenge erzeugten Tripel angegeben. Die SPARQL-Anfragen wurden auf Basis der von Morsey et al. in [142] ermittelten Anfragen erstellt.

In [142] wird ein SPARQL-Benchmark beschrieben, auf Basis dessen mehrere bekannte Tripel Stores miteinander verglichen werden. Der Benchmark basiert auf einer Menge von 25 SPARQL Anfrage-Templates, welche aus der Analyse der an den offiziellen SPARQL Endpunkt von DBpedia [48] gesendeten Nutzer-Anfragen ermittelt wurden. Diese Templates reflektieren typische Anfragen von Nutzern und decken die üblichen SPARQL Features ab. Für den in diesem Abschnitt beschriebenen Test wurden aus diesen Templates zwölf Anfragen an den Northwind-Service hergeleitet. Dazu wurden zunächst alle Anfrage-Templates entfernt, die sich zu einem anderen Anfrage-Template zwar hinsichtlich der verwendeten Ergebnismengen-Modifikatoren unterscheiden aber bzgl. der Graph-Pattern-Operatoren übereinstimmen. Der Grund dafür besteht darin, dass, wie in Kapitel 5 beschrieben, bei der

Ermittlung der relevanten OData-URIs nur die Graph-Pattern-Operatoren von Bedeutung sind. Aus jedem der übrig gebliebenen Anfrage-Templates wurde eine konkrete, an den Northwind-Service angepasste SPARQL-Anfrage erstellt. Jede dieser Anfragen wurde 10-mal gegen die prototypische Implementierung des SPARQL-OData-Layers mit registriertem Northwind-Service ausgeführt.

Als Testsystem diente ein Laptop mit einem Intel i5-2540M Prozessor mit zwei Kernen, welche jeweils mit 2,6 GHz getaktet sind. Das System verfügt über 8 GB RAM, wobei die minimale und maximale JVM Heap Größen auf 2 GB festgesetzt wurden. Als Betriebssystem kam ein 64-bit Windows 7 zum Einsatz. Der Rechner war über eine 30 MBit Leitung mit dem Internet verbunden.

Die Testergebnisse sind in Form der berechneten Durchschnittswerte in Tabelle 7 dargestellt. Jede Reihe enthält die ermittelten Zeiten einer SPARQL-Anfrage. Die Werte sind in Millisekunden angegeben. Zeiten unterhalb einer Millisekunde sind durch <1 gekennzeichnet. Unterschiede im Nanosekunden-Bereich sind nicht dargestellt, da sie für praktisch relevante Anwendungsfälle nicht von Bedeutung sind. In Spalte 1 ist jeweils der Name der entsprechenden SPARQL-Anfrage dargestellt, wie er in Anhang D angegeben ist. Spalte 2 enthält die vollständige Zeit, die zur Abarbeitung der SPARQL-Anfrage benötigt wurde. Die Spalten 3 bis 6 geben die von den jeweiligen Komponenten benötigten Zeiten an.

Anfrage	Gesamt	Query Engine	Execution Engine	RDF Adapter	SPARQL Prozessor
1 Tripel-Pattern	220	15	192	1	9
2 Tripel-Patterns	1018	< 1	981	34	3
3 Tripel-Patterns	192	4	182	1	3
4 Tripel-Patterns	3482	3	3433	43	1
5 Tripel-Patterns	780	1	761	10	6
Union	3216	6	3162	48	< 1
Optional	3188	1	3127	57	1
Filter	207	17	188	< 1	1
Union, Optional	2187	3	2157	23	3
Union, Filter	879	4	861	9	4
Optional, Filter	2922	3	2823	93	1
Graph	1723	6	1697	17	3

Tabelle 7: Ergebnisse des Performance-Tests für den Northwind-Service in ms.

Es sei darauf hingewiesen, dass die Namen der SPARQL-Anfragen nichts über deren Verarbeitungskomplexität aussagen. Beispielsweise nimmt die Abarbeitung der Anfrage *2 Tripel-Patterns* mehr Zeit in Anspruch als die Anfrage *3 Tripel-Patterns*, obwohl vermutet

werden könnte, dass die Verarbeitung aufgrund des zusätzlichen Tripel-Patterns aufwendiger sein könnte. Tatsächlich müssen aber zur Beantwortung der *2 Tripel-Patterns* Anfrage mehr Daten übertragen werden als zur Beantwortung der *3 Tripel-Patterns* Anfrage, was eine höhere Gesamtzeit zur Folge hat. Indikatoren für die Komplexität der Anfragen sind die in Anhang D angegebene Größe der zu übertragenden Datenmengen und die Anzahl der notwendigen HTTP-Aufrufe in Form der ermittelten URIs.

Wie aus Tabelle 7 ersichtlich, wird ein Großteil der zur Abarbeitung einer Anfrage benötigten Zeit von der Execution Engine beansprucht. Diese liegt, mit Ausnahme einer Anfrage (*1 Tripel-Pattern*), für alle anderen Anfragen jeweils über 90 % der Gesamtzeit. Für fünf Anfragen liegt der Wert sogar über 98 %. Dies liegt unter anderem daran, dass zur Abarbeitung einer Anfrage oftmals mehrere HTTP Aufrufe durchgeführt werden müssen. Wobei die zu übertragenden Datenmengen für die gezeigten Anfragen über ein halbes Megabyte betragen können. Beispielsweise sind zur Beantwortung der *4 Tripel-Patterns* Anfrage drei GET Abfragen notwendig, die über 800 KB an Daten abrufen. Folglich wird die Gesamtzeit maßgeblich von der zur Verfügung stehenden Bandbreite der Netzverbindung und der Antwortzeit des Servers bestimmt. Des Weiteren wird auch das Parsen der von den OData-Services zurückgegebenen XML-Nachrichten von der Execution Engine übernommen. Die dafür benötigten Zeiten wirken sich daher ebenfalls auf die in Spalte 4 dargestellten Zeiten aus. Bei den ermittelten Werten muss berücksichtigt werden, dass sich die verwendete Bibliothek zur Abfrage der OData-Services (*odata4j*) noch in einem frühen Zustand befindet. Es ist zu erwarten, dass sich mit fortschreitender Entwicklung und insbesondere mit dem Einsatz von Batch-Anfragen (siehe Abschnitt 2.3) Verbesserungen hinsichtlich der Performance ergeben.

Die Zeiten, die von der Query Engine zur Ermittlung der OData-URIs benötigt werden (Spalte 3), sind für alle Anfragen vernachlässigbar. Das gleiche gilt für die vom SPARQL-Prozessor zur tatsächlichen Ausführung der SPARQL-Anfragen benötigten Zeiten. Wobei aufgrund der Ergebnisse in [142] zu erwarten ist, dass unter Verwendung anderer SPARQL-Prozessoren weitere Verbesserungen hinsichtlich der Geschwindigkeit zu erreichen wären, und dass auch die Verarbeitung sehr großer Tripel-Mengen in akzeptabler Zeit durchgeführt werden könnte. Die Zeiten in Spalte 4 geben dabei als Vergleichswert auch die Zeiten an, die benötigt werden würden, wenn die relevanten Daten bereits als RDF in einem Triple Store vorliegen würden. Wie anhand der dargestellten Zeiten in Spalte 5 zu erkennen ist, liefert die prototypische Implementierung des RDF-Adapters für den betrachteten Anwendungsfall ebenfalls Zeiten, die vernachlässigt werden können.

Zusammenfassend kann gesagt werden, dass die in Tabelle 7 dargestellten Messergebnisse zeigen, dass der Overhead, der durch die Berechnung der OData-URIs und die Überführung der Rückgabedaten nach RDF erzeugt wird, in diesem Fall vernachlässigbar ist. Die Gesamtzeit wird maßgeblich von den zur Abfrage des Services benötigten Zeiten bestimmt. Wobei diese insbesondere von der Netzanbindung und dem Durchsatz des Servers abhängen und von der konkreten Implementierung nur wenig beeinflusst werden können.

6.2 ERP Services

Wie in dem letzten Abschnitt gezeigt, ist der Overhead, der durch die Kalkulation der relevanten OData URIs verursacht wird, vernachlässigbar. Der Großteil der zur Abarbeitung einer Anfrage benötigten Zeit wird von der Execution Engine zum Ausführen der entsprechenden HTTP-Anfragen in Anspruch genommen. Die in Tabelle 7 dargestellten Werte wurden für einen registrierten Service, den Northwind-Service, ermittelt. Es stellt sich nun die Frage, inwieweit sich das Verhältnis dieser Werte für eine größere Anzahl von Services ändert.

Wie bereits zu Beginn von Kapitel 6 erläutert, sind dabei insbesondere die für das ERP-Szenario relevanten Servicemengen von Interesse. Die Anzahl der relevanten Services kann dabei, in Abhängigkeit des Anwendungsszenarios, stark schwanken. Daher kann keine allgemein gültige Größe angegeben werden. Da die Anzahl der Services aber mit der Anzahl der für das Anwendungsszenario relevanten Geschäftsobjekte korreliert, kann eine grobe Abschätzung nach oben hin abgegeben werden. Dazu werden die Geschäftsobjekte herangezogen, für die standardmäßig SAP Enterprise Services ausgeliefert werden. Entsprechend des Enterprise Service Workplace [8] sind das aktuell[28] 331 Geschäftsobjekte für das ERP-System. Wenn neben dem ERP-System auch andere Bestandteile der SAP Business Suite, wie CRM (*Customer Relationship Management*), SCM (*Supply Chain Management*) und SRM (*Supplier Relationship Management*), betrachtet werden, erhöht sich die Anzahl der Geschäftsobjekte auf 610. Neben den vordefinierten Geschäftsobjekten besteht die Möglichkeit, weitere Geschäftsobjekte zu erstellen. Ausgehend von der Anzahl der Geschäftsobjekte des ERP, CRM, SCM und SRM dürften sich für die meisten Anwendungsfälle, selbst mit benutzerdefinierten Geschäftsobjekten, selten mehr als 1000 relevante Geschäftsobjekte ergeben. Wenn davon ausgegangen wird, dass für jedes Geschäftsobjekt ein OData-Service existiert, dann ergeben sich als Obergrenze 1000 zu betrachtende Services.

Wie bereits im vorherigen Abschnitt erläutert, wird die Zeit, die von der Execution Engine zum Abrufen und Parsen der relevanten Daten benötigt wird, maßgeblich von der Größe der abzurufenden Daten und dem Durchsatz zwischen dem OData-Server und der Execution Engine bestimmt. Da die Menge der in einem ERP-System gespeicherten Daten und die zur Verfügung stehende Bandbreite von Fall zu Fall sehr unterschiedlich sein können, ist es diesbezüglich nicht möglich, allgemeine Annahmen zu treffen. Das gleiche gilt für den RDF-Adapter und den SPARQL-Prozessor, deren benötigte Zeit ebenfalls wesentlich von der Größe und Struktur der Datenmenge abhängt. Daher wird im Folgenden die von der Query Engine benötigte Zeit in Abhängigkeit der Anzahl der registrierten Services ermittelt.

Als Test-Service dient ein OData-Service zur Abfrage aller Kundenaufträge (*Sales Order Service*) aus einem ERP-System (angelehnt an [9]). Der relevante Ausschnitt aus dem semantisch erweiterten Service-Metadaten-Dokument kann in Anhang E gefunden werden. Der Service ermöglicht die Abfrage aller Kundenaufträge (*sales orders*), der zugehörigen Auftragspositionen (*sales order items*), der entsprechenden Produkte der Auftragspositionen (*products*) und der Geschäftspartner (*business partners*). Als Testabfragen wurden, wie im

[28] April 2014

letzten Abschnitt beschrieben, zwölf Abfragen aus den in [142] beschriebenen Anfrage-Templates hergeleitet. Diese sind zusammen mit den ermittelten OData-URIs in Anhang F aufgelistet. Wie auch bei den Test-Abfragen in Anhang D, geben die Namen der in Spalte 1 von Tabelle 8 dargestellten Test-Abfragen keine Auskunft über die Komplexität der bezeichneten Anfragen. Sie beschreiben ausschließlich die unterschiedlichen Strukturen. Zur Simulation weiterer Services wurde ebenfalls der Sales Order Service verwendet. Wobei die semantische Annotation des Service-Metadaten-Dokuments für jeden zusätzlichen Service automatisch abgeändert wurde. Jede Anfrage wurde fünfzig Mal gegen das System mit einer jeweils unterschiedlichen Anzahl registrierter Services durchgeführt. Die sich aus den ermittelten Werten ergebenden berechneten Durchschnittswerte sind in Tabelle 8 dargestellt.

Anfrage	1	100	1000	2000	3000	4000	5000
1 Tripel-Pattern	< 1	< 1	< 1	< 1	< 1	< 1	< 1
2 Tripel-Patterns	< 1	< 1	< 1	1	< 1	< 1	< 1
3 Tripel-Patterns	1	1	< 1	< 1	< 1	< 1	< 1
4 Tripel-Patterns	2	< 1	< 1	< 1	< 1	< 1	< 1
5 Tripel-Patterns	1	< 1	< 1	< 1	< 1	< 1	< 1
Union	3	< 1	2	5	7	9	13
Optional	1	< 1	< 1	0	0	0	< 1
Filter	2	1	2	5	8	10	13
Union, Optional	1	1	< 1	1	< 1	< 1	1
Union, Filter	< 1	1	< 1	0	< 1	1	< 1
Optional, Filter	5	2	3	1	2	3	< 1
Graph	2	1	1	0	1	< 1	< 1

Tabelle 8: Ergebnisse des Performance-Tests für den Sales Order Service in ms.

Die erste Spalte enthält die Bezeichnungen der Anfragen entsprechend Anhang F. Alle weiteren Spalten zeigen die benötigten Zeiten der Query Engine zur Ermittlung der relevanten OData-URIs in Abhängigkeit der Anzahl der registrierten Services. Die zweite Spalte enthält die Werte für einen registrierten Service, die dritte Spalte für 100 registrierte Service usw. Die Werte sind in Millisekunden angegeben. Unterschiede im Nanosekunden-Bereich sind für den praktischen Einsatz nicht von Bedeutung. Daher sind Werte unterhalb einer Millisekunde, wie auch in Tabelle 7, durch <1 gekennzeichnet. Wie aus der Tabelle ersichtlich ist und wie aufgrund der verwendeten Indexstrukturen zu erwarten war, verhält sich die Laufzeit für die meisten Testanfragen weitgehend konstant. Ausnahmen bilden die beiden Anfragen *Union* und *Filter*, deren Laufzeitverhalten ab 1000 registrierter Services im Wesentlichen linear ist. Jedoch ist für alle Anfragen die durch die Query Engine benötigte Zeit vernachlässigbar. Die

dargestellten Ergebnisse zeigen aber auch, dass für die betrachteten Service-Mengen eine spürbare Performancesteigerung durch die Optimierung der Auswertung der Evaluierungssemantik nicht zu erreichen ist. Vielmehr wird auch in diesem Fall die Gesamtzeit maßgeblich von der konkreten Systemlandschaft bestimmt werden.

Es sei noch einmal darauf hingewiesen, dass die Implementierung eine sehr direkte Umsetzung der Evaluierungssemantik realisiert und für Servicemengen optimiert wurde, die im ERP-Szenario zu erwarten sind. Für diese arbeitet das System, wie gerade gezeigt, sehr effizient. Für größere Servicemengen ist dies jedoch nicht zwangsläufig der Fall. Wie bereits weiter oben erläutert, ist eine Implementierung mit Hinblick auf sehr große Servicemengen Bestandteil zukünftiger Arbeiten. Das dargestellte Szenario und wahrscheinlich auch die meisten anderen in der Praxis vorkommenden Anwendungsfälle können aber für gängige Anfragen bereits mit diesem System sehr effizient realisiert werden.

7 Zusammenfassung und Ausblick

Geschäftsdaten werden in ERP-Systemen durch Geschäftsobjekte repräsentiert. Ein Geschäftsobjekt kann mehrere Attribute und Referenzen zu anderen Geschäftsobjekten enthalten. Eine Menge sich gegenseitig referenzierender Geschäftsobjekte bildet einen Geschäftsobjektgraphen. Die Abfrage von Geschäftsobjektgraphen kann mit den existierenden Schnittstellen nur sehr schwierig realisiert werden. Zur Vereinfachung der Anfrage können semantische Technologien, wie RDF und SPARQL, eingesetzt werden. Abfragen von Geschäftsobjekten mit Hinblick auf ihre Position innerhalb des Geschäftsobjektgraphs können mit der graphbasierten Abfragesprache SPARQL wesentlich kompakter und intuitiver formuliert werden, als es über die bestehenden Schnittstellen möglich ist.

Motiviert durch diese Beobachtung, wurden existierende Verfahren zu Bereitstellungen von SPARQL-Endpunkten auf Basis der von den ERP-Systemen verwendeten relationalen Datenbanken untersucht. Im Rahmen dieser Untersuchungen hat sich ergeben, dass solche Ansätze, welche direkt auf der relationalen Datenbank aufsetzen, aufgrund der Größe und Komplexität des in dem ERP-System verwendeten Datenbankschemas nicht praktikabel sind. Ausgehend von diesem Ergebnis, wurden bestehende Schnittstellen des ERP-Systems, hinsichtlich ihrer Eignung zur Realisierung eines SPARQL-Endpunktes, miteinander verglichen. Wie sich gezeigt hat, weist die auf dem REST-basierten OData-Protokoll basierende Schnittstelle verschiedene Vorteile gegenüber den anderen Schnittstellen auf. Insbesondere kann die Menge der zur Beantwortung einer Anfrage notwendigen Service-Aufrufe und die zu übertragende Datenmenge durch die von OData bereitgestellten Abfrageoptionen wesentlich reduziert werden.

Auf Basis dieser Beobachtungen wurden existierende Verfahren und Systeme hinsichtlich ihrer Eignung zur Realisierung eines SPARQL-Endpunktes auf Basis einer OData-Schnittstelle untersucht. Das Ergebnis hat gezeigt, dass, nach bestem Wissen, kein System existiert, welches speziell mit Hinblick auf OData-Services entwickelt wurde. Daher wurde im nächsten Schritt eine Architektur entwickelt, die als Grundlage für die Implementierung eines entsprechenden Systems dienen kann. Ausgehend von dieser Architektur, wurden die Bereiche ermittelt, die von dem aktuellen Stand der Forschung nicht ausreichend berücksichtigt wurden. Dies betrifft insbesondere die semantische Beschreibung von OData-Services und die Auflösung von SPARQL-Anfragen gegen OData-Services. Ersteres ist eine Voraussetzung für die Auflösung der Anfragen. Für beide Bereiche wurde ein Vergleich verwandter Arbeiten durchgeführt. Die semantische Beschreibung von Web Services (Semantic Web Services) ist ein seit langem gut untersuchter Forschungsbereich. Es existieren zahlreiche Technologien und Sprachen zur semantischen Beschreibung sowohl von SOAP-/WSDL-basierten als auch von REST-basierten Web Services. Jedoch wurde keines dieser Verfahren speziell mit Hinblick auf OData und das Entitätsdatenmodell entwickelt. Von den bestehenden Ansätzen werden daher weder die spezifischen Konstrukte des Entitätsdatenmodells berücksichtigt noch sind sie als Erweiterung von CSDL definiert. Des Weiteren wurden viele der bestehenden Ansätze mit einer anderen Zielsetzung entwickelt, was dazu führt, dass sie für das anvisierte Szenario, die Überführung von SPARQL-Anfragen in OData-Service-Aufrufe, überdimensioniert sind oder nur schwierig verwendet werden

können. Ähnliches gilt für die bereits existierenden Verfahren zur Bereitstellung von SPARQL-Endpunkten auf Basis von Web Services. Da keiner dieser Ansätze OData-spezifische Konstrukte, wie z.B. Abfrageoptionen, berücksichtigt, sind die sich ergebenden Service-Kompositionen weder hinsichtlich der Anzahl der benötigten Service-Aufrufe noch in Bezug auf die zu übertragenden Datenmengen optimal. Außerdem unterstützen viele der existierenden Ansätze nur bestimmte Teilmengen von SPARQL als Abfragesprache.

Ausgehend von dem Vergleich der verwandten Arbeiten wurde eine semantische Beschreibung speziell für OData-Services und dem diesen zugrundeliegendem Entitätsdatenmodell entwickelt. Die Konzepte zur semantischen Beschreibung wurden dabei, ausgehend von den verschiedenen Konstrukten des Entitätsdatenmodells, mit Bezug auf die von RDF bereitgestellten Konzepte, definiert. Sie ermöglichen die Spezifikation von Abbildungen von konkreten Ausprägungen des Entitätsdatenmodells auf RDF-Graphen. Dazu wurde CSDL um zusätzliche Attribute erweitert, welche die Definition von Tripel-Templates ermöglichen. Ein Tripel-Template beschreibt eine Menge von Tripeln, die aus den Daten eines Services erzeugt werden können. Die Menge aller durch ein semantisch annotiertes CSDL-Dokument definierten Tripel-Templates bildet das Graph-Template des Services. Es beschreibt den Aufbau und die Struktur des RDF-Graphen, der durch Überführung der von den Service bereitgestellten Daten generiert werden kann. Ein Graph-Template bzw. die Tripel-Templates eines Graph-Templates enthalten alle Informationen, die notwendig sind, um die zur Erzeugung der durch sie beschriebenen Tripel notwendigen Daten abrufen zu können.

Aufbauend auf einer Formalisierung der zur semantischen Beschreibung eingeführten Konzepte wurde mit Hinblick auf eine zu entwickelnde Evaluierungssemantik für Graph-Templates die Auflösung der semantischen Annotationen formalisiert. Dies umfasst insbesondere das Auflösen der zur vereinfachten Beschreibung eingeführten Abkürzungen. Zur Ermittlung der Graph-Templates wurden Algorithmen vorgestellt, welche das Zusammenfassen der Tripel-Templates formalisieren. Dabei werden auch die Informationen darüber, welche Daten zur Erzeugung der durch ein Tripel-Template beschriebenen Tripel benötigt werden, in Form sogenannter Pfade formal gefasst. Ausgehend von diesen Verfahren wurde eine spezielle Evaluierungssemantik definiert, welche die Auflösung von SPARQL-Algebra-Ausdrücken gegen semantisch beschriebene OData-Services erlaubt. Dazu wurde zunächst für den allgemeinen Fall ein Vergleich von Tripel-Patterns mit Tripel-Templates durchgeführt, um die zur Beantwortung einer Anfrage möglicherweise relevanten Tripel-Templates bestimmen zu können. Auf Basis des Vergleichs wurde ein Algorithmus entwickelt, der entscheidet, ob es für ein gegebenes Tripel-Pattern und ein gegebenes Tripel-Template eine Lösung geben könnte, welche das Tripel-Pattern auf ein Tripel der durch das Tripel-Template beschriebenen Menge an Tripeln abbildet. Anschließend wurde die Abbildung von Variablen und leeren Knoten in Form von Lösungs-Template-Mappings (LTMs), RDF-Instanz-Template-Mappings (RITMs) und Pattern-Instanz-Template-Mappings (PITMs) definiert. Die Definitionen erfolgten dabei angelehnt an die entsprechenden Definitionen des Pattern-Instanz-Mappings, Lösungs-Mappings und RDF-Instanz-Mappings der SPARQL-Spezifikation. Im nächsten Schritt wurde die Erzeugung von LTMs, RITMs und PITMs für Tripel-Patterns mit Hinblick auf zu diesen passende Tripel-Templates erläutert. Ausgehend von diesen Erläuterungen wurde der Lösungsbegriffs für Tripel-Patterns mit Bezug auf Graph-Templates definiert. Auf Basis des Lösungsbegriffs wurde die Evaluierungssemantik

für die Graph-Pattern-Operatoren der SPARQL-Algebra hinsichtlich der besonderen Eigenschaften der Graph-Templates angepasst. Diese Evaluierungssemantik erlaubt die Auflösung von SPARQL-Anfragen gegen die Graph-Templates der semantische beschriebenen OData-Services. Das Ergebnis der Auflösung ist eine Menge von Lösungs-Template-Mappings, die mögliche Lösungen, im Sinne der SPARQL-Spezifikation, auf die von den Graph-Templates der Services beschriebenen RDF-Graphen spezifizieren. Dabei werden alle zur Berechnung dieser Lösungen relevanten Tripel-Templates bestimmt.

Um die durch diese Tripel-Templates beschriebenen Tripel erzeugen zu können, wurde ein Verfahren vorgestellt, das die Tripel-Templates in OData-URIs überführt. Dabei werden, unter Berücksichtigung der zugehörigen Tripel-Patterns, Abfrageoptionen in die URIs eingefügt, welche die zu übertragende Datenmenge reduzieren. Die OData-URIs adressieren die zur Erzeugung der Tripel benötigten Daten. Wobei diese aus Performance-Gründen im Rahmen der Auflösung des SPARQL-Algebra-Ausdrucks soweit wie möglich miteinander verschmolzen werden.

Das vorgestellte Verfahren wurde prototypisch implementiert. Der Prototyp stellt eine sehr direkte Umsetzung der Evaluierungssemantik dar. Typische Implementierungsstrategien und Optimierungsverfahren, wie sie bei der Implementierung von SPARQL-Prozessoren üblich sind, wurden nicht umgesetzt, um die Realisierbarkeit des vorgestellten Verfahrens zu zeigen, ohne dass dies durch optimierungsspezifische Implementierungsdetails verdeckt wird. Der Prototyp wurde anhand zweier Szenarien evaluiert. Das erste Szenario umfasste die Bereitstellung eines SPARQL-Endpunktes für den öffentlich verfügbaren Northwind-Service. Die Geschwindigkeitstests wurden mittels mehrerer SPARQL-Anfragen durchgeführt, deren Struktur auf häufig vorkommenden Anfragen an den SPARQL-Endpunkt von DBpedia basiert. Sie reflektieren typische Nutzeranfragen und decken die üblichen SPARQL-Features ab. Die Ergebnisse des Performance-Tests haben gezeigt, dass bei der Verarbeitung der Anfragen, die zur Kalkulation der URIs benötigten Zeiten vernachlässigbar sind. Die Gesamtzeit wird maßgeblich von der Antwortzeit des Servers und der zur Verfügung stehenden Bandbreite beeinflusst. Bei dem zweiten Szenario wurden die Tests gegen den Service eines ERP-Systems zur Abfrage aller Kundenaufträge durchgeführt. Dabei wurde die Zeit zur Ermittlung der relevanten OData-URIs für eine gegebene SPARQL-Anfrage in Abhängigkeit der Anzahl der registrierten Services betrachtet. Wobei sich auf Service-Mengen beschränkt wurde, die für ERP-Systeme zu erwarten sind. Die Ergebnisse haben gezeigt, dass selbst bei einer sehr direkten Umsetzung der Evaluierungssemantik für einige tausend registrierte Services Zeiten erreicht werden können, die vernachlässigbar sind. Der implementierte Prototyp dürfte somit die meisten praxisrelevanten Anwendungsszenarien abdecken.

Diese Arbeit hat gezeigt, dass es möglich ist, SPARQL-Endpunkte auf Basis von OData-Schnittstellen zu realisieren. Im Rahmen dieser Arbeit wurden mehrere, neuartige Konzepte vorgestellt, die den aktuellen Stand der Forschung erweitern. Es wurde eine Sprache zur semantischen Beschreibung von OData-Services entwickelt, mit der es möglich ist, Abbildungen auf RDF zu definieren. Des Weiteren wurde eine Evaluierungssemantik zur Auflösung von SPARQL-Anfragen gegen semantisch beschriebene OData-Services definiert. Außerdem wurde gezeigt, wie im Rahmen der Auflösung OData-URIs zur Adressierung der

relevanten Daten erstellt werden können. In ihrer Gesamtheit ermöglichen sie die Realisierung von Systemen zur Ausführung von SPARQL-Anfragen gegen OData-Services.

Mit solch einem System können nicht nur SPARQL-Endpunkte für ERP-Systeme bereitgestellt werden, vielmehr ist es möglich, jeden existierenden OData-Service für die Anfrage mittels SPARQL zugänglich zu machen. Damit werden große Mengen an Daten, die bisher hinter OData-Schnittstellen verborgen waren, für die Integration mit dem Semantic Web verfügbar gemacht. Dabei kann SPARQL auch bestimmte Anfragen, insbesondere graphbasierte Anfragen, an die Systeme vereinfachen, indem es eine kompaktere und intuitivere Formulierung der Anfragen unterstützt. Des Weiteren können Systeme, die eine OData-Schnittstelle anbieten und deren Integration miteinander bisher nicht oder nur sehr schwierig zu realisieren war, z.B. mittels Systeme zur föderierten Abfrage miteinander integriert werden.

Mit Hinblick auf zukünftige Arbeiten können folgende Punkte genannt werden, welche diese Arbeit ergänzen bzw. erweitern können:

- **Berücksichtigung von Service-Operationen:** OData ermöglicht, neben der Realisierung von REST-basierten Services, auch die Bereitstellungen sogenannter Service-Operationen. Service-Operationen können Ein- und Ausgabeparameter, einen Rückgabewert und Nebeneffekte haben. Sie sind somit mit typischen SOAP-/WSDL-basierten Services vergleichbar. Die Berücksichtigung solcher Service-Operationen bei der Auflösung der SPARQL-Anfragen ist Bestandteil zukünftiger Arbeiten. Wobei davon auszugehen ist, dass aufgrund der Ähnlichkeit zu SOAP-/WSDL-basierten Services, diesbezüglich auf bestehenden Arbeiten aufgesetzt werden kann.
- **RDF 1.1, SPARQL 1.1 und OData 4.0:** Während der Erstellung dieser Arbeit wurden die betrachteten Technologien laufend weiterentwickelt. Die zuständigen W3C Arbeitsgruppen veröffentlichten neue Versionen von RDF [71, 147] und SPARQL [91, 179]. OData wurde in Version 4.0 [150, 151, 152] von OASIS als Standard anerkannt. Im Rahmen der Standardisierungen wurde zahlreiche Neuerungen und Erweiterungen eingeführt, deren Berücksichtigung Bestandteil zukünftiger Arbeiten ist.
- **SAP Erweiterungen:** Um bestimmte Anwendungsfälle abzudecken, hat SAP verschiedene Erweiterungen für OData definiert [90]. Die Berücksichtigung dieser Erweiterungen bei der Auflösung der SPARQL-Anfragen ist Bestandteil zukünftiger Arbeiten.
- **Verarbeitung großer Graph-Templates:** Zur Verarbeitung großer Service-Mengen bzw. großer Graph-Templates ist es notwendig, Erfahrungen, die bei der Realisierung von SPARQL-Prozessoren gewonnen wurden, bei der Implementierung der in dieser Arbeit vorgestellten Evaluierungssemantik zu berücksichtigen. Es ist zu erwarten, dass bei der Umsetzung typischer SPARQL Implementierungsstrategien und Optimierungstechniken Systeme entwickelt werden können, die, mit Bezug auf die Anzahl der Tripel-Templates, eine ähnliche Skalierung aufweisen wie bestehende SPARQL-Prozessoren. Die Evaluierung dieser Annahme mittels einer noch zu entwickelnden Implementierung ist Gegenstand zukünftiger Arbeiten. In diesem

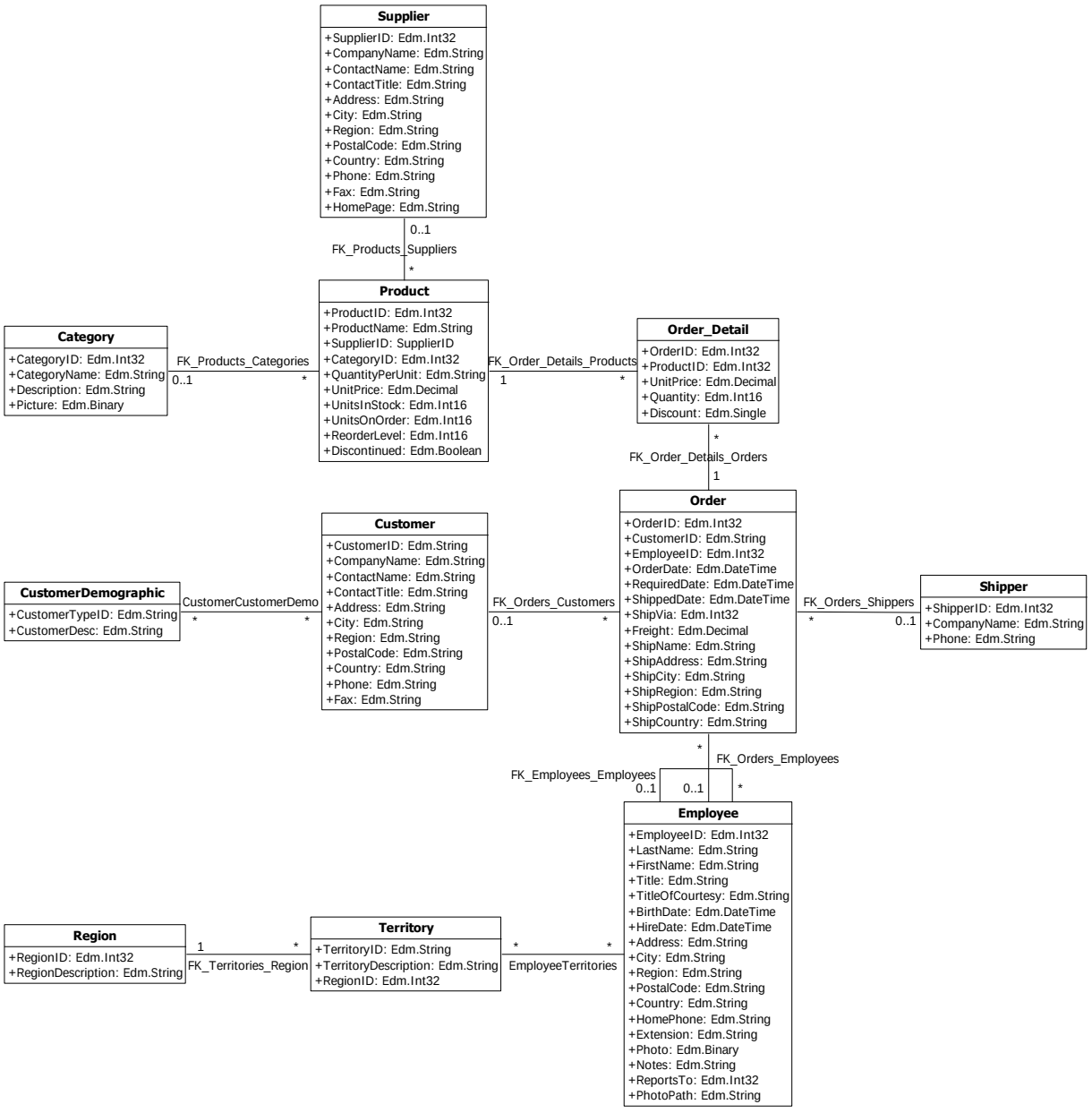
Zusammenhang ist auch die Erstellung entsprechender statistischer Signifikanztests in Erwägung zu ziehen.

- **Tools:** Die Akzeptanz einer Technologie oder eines Verfahrens wird maßgeblich von der Verfügbarkeit und der Ausgereiftheit zugehöriger Tools beeinflusst. Die Entwicklung solcher Tools, insbesondere mit Hinblick auf die semantische Erweiterung von CSDL-Dokumenten, ist Bestandteil zukünftiger Arbeiten.

Teil IV

Anhang

A.Entitätsdatenmodell des Northwind-Services



B. CSDL-Erweiterungen in XSD

```
<xsd:schema elementFormDefault="qualified"
 attributeFormDefault="qualified"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:sem="http://research.sap.de/OData/SemanticDesc"
 xmlns:edmx="http://schemas.microsoft.com/ado/2008/10/edmx"
 targetNamespace="http://research.sap.de/OData/SemanticDesc">

 <xsd:attribute name="Type" type="xsd:string"/>
 <xsd:attribute name="URI" type="xsd:string"/>
 <xsd:attribute name="URI_struc" type="sem:URIStructure"/>
 <xsd:attribute name="Mapping" type="xsd:string"/>
 <xsd:attribute name="Datatype" type="xsd:string"/>
 <xsd:attribute name="Lang" type="xsd:string"/>
 <xsd:attribute name="Constraint" type="xsd:string"/>

 <xsd:simpleType name="URIStructure">
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="hash"/>
   <xsd:enumeration value="303"/>
   <xsd:enumeration value="combined"/>
  </xsd:restriction>
 </xsd:simpleType>

</xsd:schema>
```

```
<xsd:schema elementFormDefault="qualified"
 attributeFormDefault="qualified"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:edm="http://schemas.microsoft.com/ado/2008/09/edm"
 xmlns:sem="http://research.sap.de/OData/SemanticDesc"
targetNamespace="http://schemas.microsoft.com/ado/2008/
                 09/edm">

 <xsd:import
  namespace="http://research.sap.de/OData/SemanticDesc" />

 <xsd:redefine
  schemaLocation="System.Data.Resources.CSDLSchema_2.xsd">

  <xsd:complexType name="TEntityType">
```

```
   <xsd:complexContent>
    <xsd:extension base="edm:TEntityType">
     <xsd:attribute ref="sem:Type"/>
     <xsd:attribute ref="sem:URI"/>
     <xsd:attribute ref="sem:URI_struc"/>
     <xsd:attribute ref="sem:Mapping"/>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="TComplexType">
   <xsd:complexContent>
    <xsd:extension base="edm:TComplexType">
     <xsd:attribute ref="sem:Type"/>
     <xsd:attribute ref="sem:Mapping"/>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>

<!-- Der Datentyp mit der Bezeichnung edm:TEntitySet steht für
den entsprechenden anonymen Datentyp aus dem edm-Namensraum. -
->
  <xsd:complexType name="TEntitySet">
   <xsd:complexContent>
    <xsd:extension base="edm:TEntitySet">
     <xsd:attribute ref="sem:Type"/>
     <xsd:attribute ref="sem:URI"/>
     <xsd:attribute ref="sem:URI_struc"/>
     <xsd:attribute ref="sem:Mapping"/>
     <xsd:attribute ref="sem:Constraint"/>
   </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="TEntityProperty">
   <xsd:complexContent>
    <xsd:extension base="edm:TEntityProperty">
     <xsd:attribute ref="sem:Mapping"/>
     <xsd:attribute ref="sem:Datatype"/>
     <xsd:attribute ref="sem:Lang"/>
     <xsd:attribute ref="sem:Type"/>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
```

```
  <xsd:complexType name="TComplexTypeProperty">
   <xsd:complexContent>
    <xsd:extension base="edm:TComplexTypeProperty">
     <xsd:attribute ref="sem:Mapping"/>
     <xsd:attribute ref="sem:Datatype"/>
     <xsd:attribute ref="sem:Lang"/>
     <xsd:attribute ref="sem:Type"/>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="TNavigationProperty">
   <xsd:complexContent>
    <xsd:extension base="edm:TNavigationProperty">
     <xsd:attribute ref="sem:Mapping"/>
     <xsd:attribute ref="sem:Type"/>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="TDataServices">
   <xsd:extension base="edmx:TDataServices">
     <xsd:attribute ref="sem:URI"/>
     <xsd:attribute ref="sem:URI_struc"/>
     <xsd:attribute ref="sem:Datatype"/>
   </xsd:extension>
  </xsd:complexType>

 </xsd:redefine>
</xsd:schema>
```

C. Semantisch erweitertes EDM des Northwind-Services

Dieser Anhang enthält einen semantisch erweiterten Ausschnitt des aus [135] entnommenen EDMs des Northwind-Services.

```
<edmx:DataServices sem:Datatype="untyped"
 sem:URI="northwind.beispiel.org" >

<EntityType Name="Customer"
 sem:Mapping="?customer rdf:type northw:Customer .
  ?customer northw:order ?order .
  ?customer northw:employee ?employee .
  ?customer northw:customer_demographic ?customerDemo">
 <Key>
  <PropertyRef Name="CustomerID" />
 </Key>
 <Property Name="CustomerID" Type="Edm.String"
  sem:Mapping="?customer northw:customer_id .
   ?customer northw:id . ?customer northw:identifier" />
 <Property Name="CompanyName" Type="Edm.String"
  sem:Mapping="?customer northw:company_name" />
 <Property Name="ContactName" Type="Edm.String"
  sem:Mapping="?customer northw:contact_name" />
 <Property Name="ContactTitle" Type="Edm.String"
  sem:Mapping="?customer northw:contact_title" />
 <Property Name="Address" Type="Edm.String"
  sem:Mapping="?customer northw:address"/>
 <Property Name="City" Type="Edm.String"
  sem:Mapping="northw:city"/>
 <Property Name="Region" Type="Edm.String"
  sem:Mapping="?customer northw:region"/>
 <Property Name="PostalCode" Type="Edm.String"
  sem:Mapping="?customer northw:postal_code" />
 <Property Name="Country" Type="Edm.String"
  sem:Mapping="?customer northw:country"/>
 <Property Name="Phone" Type="Edm.String"
  sem:Mapping="?customer northw:phone" />
 <Property Name="Fax" Type="Edm.String"
  sem:Mapping="?customer northw:fax" />
 <NavigationProperty Name="Orders" />
 <NavigationProperty Name="CustomerDemographics" />
</EntityType>

<EntityType Name="Employee"
 sem:Mapping="?employee rdf:type northw:Employee .
  ?employee northw:order ?order .
  ?employee northw:handles ?order .
  ?employee northw:territory ?territory .
  ?employee northw:responsible_for ?territory ">
 <Key>
```

```
  <PropertyRef Name="EmployeeID" />
 </Key>
 <Property Name="EmployeeID" Type="Edm.Int32"
  sem:Mapping="?employee northw:employee_id"/>
 <Property Name="LastName" Type="Edm.String"
  sem:Mapping="?employee northw:last_name
               $LastName^^xsd:string"/>
 <Property Name="FirstName" Type="Edm.String"
  sem:Mapping="?employee northw:first_name" />
 <Property Name="Title" Type="Edm.String"
  sem:Mapping="?employee northw:title $Title@de"/>
 <Property Name="TitleOfCourtesy" Type="Edm.String"
  sem:Mapping="?employee northw:title_of_courtesy"/>
 <Property Name="BirthDate" Type="Edm.DateTime"
  sem:Mapping="?employee northw:birth_date"/>
 <Property Name="HireDate" Type="Edm.DateTime"
  sem:Mapping="?employee northw:hire_date"/>
 <Property Name="Address" Type="Edm.String"
  sem:Mapping="?employee northw:address"/>
 <Property Name="City" Type="Edm.String"
  sem:Mapping="?employee northw:city"/>
 <Property Name="Region" Type="Edm.String"
  sem:Mapping="?employee northw:region"/>
 <Property Name="PostalCode" Type="Edm.String"
  sem:Mapping="?employee northw:postal_code"/>
 <Property Name="Country" Type="Edm.String"
  sem:Mapping="?employee northw:country $@en"/>
 <Property Name="HomePhone" Type="Edm.String"
  sem:Mapping="?employee northw:home_phone"/>
 <Property Name="Extension" Type="Edm.String"
  sem:Mapping="?employee northw:extension"/>
 <Property Name="Photo" Type="Edm.Binary"
  sem:Mapping="?employee northw:photo"/>
 <Property Name="Notes" Type="Edm.String"
  sem:Mapping="?employee northw:notes $@en"/>
 <Property Name="ReportsTo" Type="Edm.Int32"
  sem:Mappping="?employee northw:reports_to"/>
 <Property Name="PhotoPath" Type="Edm.String"
  sem:Mapping="?employee northw:photo_path"/>
 <NavigationProperty Name="Employees1" />
 <NavigationProperty Name="Orders" />
 <NavigationProperty Name="Territories" />
</EntityType>

<EntityType Name="Order"
 sem:Mapping="?order rdf:type northw:Order .
  ?order northw:order_detail ?orderDetail .
  ?order northw:employee ?employee .
  ?order northw:customer ?customer .
  ?order northw:shipper ?shipper">
 <Key>
```

```
  <PropertyRef Name="OrderID" />
 </Key>
 <Property Name="OrderID" Type="Edm.Int32"
  sem:Mapping="?order northw:order_id .
   ?order northw:id . ?order northw:identifier" />
 <Property Name="CustomerID" Type="Edm.String"
  sem:Mapping="?order northw:customer_id"/>
 <Property Name="EmployeeID" Type="Edm.Int32"
  sem:Mapping="?order northw:employee_id"/>
 <Property Name="OrderDate" Type="Edm.DateTime"
  sem:Mapping="?order northw:order_date"/>
 <Property Name="RequiredDate" Type="Edm.DateTime"
  sem:Mapping="?order northw:required_date"/>
 <Property Name="ShippedDate" Type="Edm.DateTime"
  sem:Mapping="?order northw:shipped_date"/>
 <Property Name="ShipVia" Type="Edm.Int32"
  sem:Mapping="?order northw:ship_via" />
 <Property Name="Freight" Type="Edm.Decimal"
  sem:Mapping="?order northw:freight"/>
 <Property Name="ShipName" Type="Edm.String"
  sem:Mapping="?order northw:ship_name"/>
 <Property Name="ShipAddress" Type="Edm.String"
  sem:Mapping="?order northw:ship_address"/>
 <Property Name="ShipCity" Type="Edm.String"
  sem:Mapping="?order northw:ship_city"/>
 <Property Name="ShipRegion" Type="Edm.String"
  sem:Mapping="?order northw:ship_region"/>
 <Property Name="ShipPostalCode" Type="Edm.String"
  sem:Mapping="?order northw:ship_postal_code"/>
 <Property Name="ShipCountry" Type="Edm.String"
  sem:Mapping="?order northw:ship_country $@en"/>
 <NavigationProperty Name="Customer" />
 <NavigationProperty Name="Employee" />
 <NavigationProperty Name="Order_Details" />
 <NavigationProperty Name="Shipper" />
</EntityType>

<EntityType Name="Product"
 sem:Mapping="?product rdf:type northw:Product .
  ?product northw:order_detail ?orderDetail .
  ?product northw:category ?category .
  ?product northw:supplier ?supplier">
 <Key>
  <PropertyRef Name="ProductID" />
 </Key>
 <Property Name="ProductID" Type="Edm.Int32"
  sem:Mapping="?product northw:product_id"/>
 <Property Name="ProductName" Type="Edm.String"
  sem:Mapping="?product northw:product_name"/>
 <Property Name="SupplierID" Type="Edm.Int32"
  sem:Mapping="?product northw:supplier_id"/>
```

```
 <Property Name="CategoryID" Type="Edm.Int32"
  sem:Mapping="?product northw:category_id"/>
 <Property Name="QuantityPerUnit" Type="Edm.String"
  sem:Mapping="?product northw:quantity_per_unit"/>
 <Property Name="UnitPrice" Type="Edm.Decimal"
  sem:Mapping="?product northw:unit_price"/>
 <Property Name="UnitsInStock" Type="Edm.Int16"
  sem:Mapping="?product northw:units_in_stock"/>
 <Property Name="UnitsOnOrder" Type="Edm.Int16"
  sem:Mapping="?product northw:units_on_order $UnitsOnOrder .
  ?product northw:units_ordered $UnitsOnOrder"/>
 <Property Name="ReorderLevel" Type="Edm.Int16"
  sem:Mapping="?product northw:reorder_level"/>
 <Property Name="Discontinued" Type="Edm.Boolean"
  sem:Mapping="?product northw:discontinued"/>
 <NavigationProperty Name="Category" />
 <NavigationProperty Name="Order_Details" />
 <NavigationProperty Name="Supplier" />
</EntityType>

<EntityType Name="Territory"
 sem:Mapping="?territory rdf:type northw:Territory .
  ?territory northw:region ?region .
  ?territory northw:employee ?employee">
 <Key>
  <PropertyRef Name="TerritoryID" />
 </Key>
 <Property Name="TerritoryID" Type="Edm.String"
  sem:Mapping="?territory northw:territory_id" />
 <Property Name="TerritoryDescription" Type="Edm.String"
  sem:Mapping="?territory northw:description"/>
 <Property Name="RegionID" Type="Edm.Int32"
  sem:Mapping="?territory northw:region_id" />
 <NavigationProperty Name="Region" />
 <NavigationProperty Name="Employees" />
</EntityType>
</Schema>

<Schema Namespace="ODataWeb.Northwind.Model" >
 <EntityContainer Name="NorthwindEntities" >
  <EntitySet Name="Customers"
   EntityType="NorthwindModel.Customer" />
  <EntitySet Name="Employees"
   EntityType="NorthwindModel.Employee" />
  <EntitySet Name="Orders"
   EntityType="NorthwindModel.Order" />
  <EntitySet Name="Products"
   EntityType="NorthwindModel.Product" />
  <EntitySet Name="Territories"
   EntityType="NorthwindModel.Territory" />
 </EntityContainer>
```

```
</Schema>
</edmx:DataServices>
</edmx:Edmx>
```

D.Northwind-Service Test-Anfragen

1 Tripel-Pattern

```
SELECT *
WHERE {
 <http://northwind.beispiel.org/
  296509342532779722202849713275971452O5/customer#ALFKI>
  northw:company_name ?companyName }
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Customers?$select=CompanyName,CustomerID&$filter=CustomerID eq 'ALFKI'*

Datenmenge: 1,12 KB
Tripel: 1

2 Tripel-Patterns

```
SELECT ?contactName
WHERE {
 ?customer rdf:type northw:Customer .
 ?customer northw:contact_name ?contactName }
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Customers?$select=CustomerID,ContactName*

Datenmenge: 57,9 KB
Tripel: 182

3 Tripel-Patterns

```
SELECT ?firstName
WHERE {
 ?employee rdf:type northw:Employee .
 ?employee northw:last_name 'Davolio'^^xsd:string .
 ?employee northw:first_name ?firstName }
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Employees?$select=EmployeeID,LastName,FirstName&$filter=LastName eq 'Davolio'*

Datenmenge: 1,14 KB
Tripel: 3

4 Tripel-Patterns

```
SELECT *
WHERE {
 ?employee rdf:type northw:Employee .
 ?order northw:ship_via '3' .
 ?customer northw:order ?order .
 ?customer rdf:type northw:Customer }
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Employees?$select=EmployeeID*
- *http://services.odata.org/V2/Northwind/Northwind.svc/Orders?$select=ShipVia,OrderID&$filter=ShipVia eq 3*
- *http://services.odata.org/V2/Northwind/Northwind.svc/Customers?$select=Orders/OrderID,CustomerID&$expand=Orders*

Datenmenge: 809 KB
Tripel: 1185

5 Tripel-Patterns

```
SELECT ?productID ?productName ?supplierID ?categoryID
WHERE {
 ?product rdf:type northw:Product .
 ?product northw:product_id ?productID .
 ?product northw:product_name ?productName .
 ?product northw:supplier_id ?supplierID .
 ?product northw:category_id ?categoryID }
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Products?$select=ProductID,ProductName,SupplierID,CategoryID*

Datenmenge: 58 KB
Tripel: 385

Union

```
SELECT ?var5 ?var6 ?order ?orderDate ?freight
WHERE {
 {<http://northwind.beispiel.org/
   29650934253277972220284971327597145205/customer#ALFKI>
   ?var5 ?var6 .
  ?var6 northw:order_date ?orderDate }
 UNION
```

```
 {?order ?var5 <http://northwind.beispiel.org/
  29650934253277972220284971327597145205/customer#ALFKI> .
  ?order northw:freight ?freight}}
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Orders?$select=Customer/CustomerID,Freight&$expand=Customer&$filter=Customer/CustomerID eq 'ALFKI'*
- *http://services.odata.org/V2/Northwind/Northwind.svc/Customers?$select=Orders/OrderID,CustomerID&$expand=Orders&$filter=CustomerID eq 'ALFKI'*
- *http://services.odata.org/V2/Northwind/Northwind.svc/Orders?$select=OrderDate,OrderID*

Datenmenge: 533 KB
Tripel: 848

Optional

```
SELECT ?freight ?shippedDate ?shipName
WHERE {
 ?order rdf:type northw:Order .
 ?order northw:freight ?freight .
 ?order northw:ship_via '2' .
 ?order northw:shipped_date ?shippedDate .
 OPTIONAL { ?order northw:ship_name ?shipName . } .}
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Orders?$select=ShipName,OrderID*
- *http://services.odata.org/V2/Northwind/Northwind.svc/Orders?$select=Freight,ShipVia,ShippedDate&$filter=ShipVia eq 2*

Datenmenge: 722 KB
Tripel: 2627

Filter

```
SELECT *
WHERE {
 <http://northwind.beispiel.org/
  29650934253277972220284971327597145205/customer#BOTTM>
  ?var2 ?var1 .
 FILTER (?var2 = northw:city || ?var2 = northw:region) }
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Customers?$select=City,CustomerID,Region&$filter=CustomerID eq 'BOTTM'*

Datenmenge: 1,12 KB
Tripel: 2

Union, Optional

```
SELECT *
WHERE {
 ?territory rdf:type northw:Territory .
 ?territory northw:description ?description .
 ?employee rdf:type northw:Employee .
 { ?employee northw:territory ?territory }
 UNION
 { ?employee northw:responsible_for ?territory }
 { ?employee northw:order ?order }
 UNION
 { ?employee northw:handles ?order }
 OPTIONAL { ?employee northw:city ?city }
 OPTIONAL { ?employee northw:title ?title }}
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Employees?$select=Title,EmployeeID,City,Territories/TerritoryID,Orders/OrderID&$expand=Territories,Orders*
- *http://services.odata.org/V2/Northwind/Northwind.svc/Territories?$select=TerritoryID,TerritoryDescription*

Datenmenge: 635 KB
Tripel: 1891

Union, Filter

```
SELECT ?product ?unitsOrdered
WHERE {
 {?product rdf:type northw:Product .
  ?product northw:units_on_order ?unitsOrdered .
  FILTER (xsd:integer(?unitsOrdered) > 20)}
 UNION
 {?product rdf:type northw:Product .
  ?product northw:units_ordered ?unitsOrdered .
  FILTER (xsd:integer(?unitsOrdered) < 5)}}
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Products?$select=ProductID,UnitsOnOrder*

Datenmenge: 49,4 KB
Tripel: 231

Optional, Filter

```
SELECT ?requiredDate
WHERE {
 ?order northw:required_date ?requiredDate .
 ?order rdf:type northw:Order .
 ?order northw:ship_country ?country .
 ?order northw:ship_via ?shipVia .
 ?order northw:freight ?freight .
 ?order northw:ship_name ?ShipName .
 ?order ?var33 ?var34 .
 OPTIONAL { ?order northw:ship_region ?shipRegion . }
 FILTER (
  ?var33 = <http://services.odata.org/Northwind#order_id> ||
  ?var33 = <http://services.odata.org/Northwind#id> ||
  ?var33 = <http://services.odata.org/Northwind#identifier> )
 FILTER (xsd:integer(?var34) > 30 ) .
 FILTER (xsd:integer(?shipVia) < 2) .
 FILTER (?country = 'France'@en ||
         ?country = <http://test1.de/123/#France> ||
         ?country = <http://test2.de/123/#France>)}
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Orders?$select=ShipRegion,OrderID,RequiredDate,ShipCountry,ShipVia,Freight,ShipName*

Datenmenge: 731 KB
Tripel: 7793

Graph

```
SELECT *
FROM NAMED
<http://services.odata.org/V2/Northwind/Northwind.svc/>
WHERE {
 GRAPH ?service {
 ?employee rdf:type northw:Employee ;
           northw:country ?country ;
           northw:notes ?notes ;
           northw:birth_date ?birthDate ;
           northw:hire_date ?hireDate ;
 { ?employee northw:first_name 'Robert' . }
 UNION
 { ?order northw:employee ?employee ;
          northw:ship_via '3' . }
 OPTIONAL { ?employee northw:region ?region }
 OPTIONAL { ?employee northw:postal_code ?postalCode }
 OPTIONAL { ?employee northw:country ?city }
```

```
 OPTIONAL { ?employee northw:extension ?extension }
 FILTER (langMatches( lang(?country), 'en') &&
         langMatches( lang(?notes), 'en'))}}
```

URLs:

- *http://services.odata.org/V2/Northwind/Northwind.svc/Employees?$select=Extension,EmployeeID,Country,PostalCode,Region,Notes,BirthDate,HireDate*
- *http://services.odata.org/V2/Northwind/Northwind.svc/Employees?$select=FirstName&$filter=FirstName eq 'Robert'*
- *http://services.odata.org/V2/Northwind/Northwind.svc/Orders?$select=Employee/EmployeeID,OrderID,ShipVia&$expand=Employee&$filter=ShipVia eq 3*

Datenmenge: 384 KB
Tripel: 579

E. Semantisch erweiterterter Sales Order Service

```
<edmx:DataServices m:DataServiceVersion="2.0"
 sem:Datatype="untyped" sem:URI="erp.beispiel.org">

<EntityType Name="SalesOrder"
 sem:Mapping="?salesOrder rdf:type boo:SalesOrder .
 ?salesOrder boo:business_partner ?businessPartner
 ?salesOrder boo:sales_order_item ?salesOrderItem">
 <Key>
  <PropertyRef Name="SalesOrderID"/>
 </Key>
 <Property Name="SalesOrderID" Type="Edm.String"
  sem:Mapping="?salesOrder boo:id"/>
 <Property Name="NetSum" Type="Edm.Decimal"
  sem:Mapping="?salesOrder boo:net_sum"/>
 <Property Name="Tax" Type="Edm.Decimal"
  sem:Mapping="?salesOrder boo:tax"/>
 <Property Name="Currency" Type="Edm.String"
  sem:Mapping="?salesOrder boo:currency"/>
 <Property Name="ChangedAt" Type="Edm.DateTime"
  sem:Mapping="?salesOrder boo:changed_at"/>
 <Property Name="Note" Type="Edm.String"
  sem:Mapping="?salesOrder boo:note"/>
 <Property Name="CreatedAt" Type="Edm.DateTime"
  sem:Mapping="?salesOrder boo:created_at" />
 <Property Name="TotalSum" Type="Edm.Decimal"
  sem:Mapping="?salesOrder boo:total_sum"/>
 <Property Name="Status" Type="Edm.String"
  sem:Mapping="?salesOrder boo:status" />
 <Property Name="CustomerName" Type="Edm.String"
  sem:Mapping="?salesOrder boo:customer_name" />
 <Property Name="BusinessPartnerID" Type="Edm.String"
  sem:Mapping="?salesOrder boo:business_partner_id" />
 <NavigationProperty Name="BusinessPartner" />
 <NavigationProperty Name="SalesOrderItems" />
</EntityType>

<EntityType Name="SalesOrderItem"
 sem:Mapping="?salesOrderItem rdf:type boo:SalesOrderItem .
  ?salesOrderItem boo:product ?product .
  ?salesOrderItem boo:ordered ?product .
  ?salesOrderItem boo:sales_order_header ?salesOrder .
  ?salesOrderItem boo:sales_order ?salesOrder">
 <Key>
 <PropertyRef Name="Position"/>
 <PropertyRef Name="SalesOrderID"/>
 </Key>
 <Property Name="QuantityUnit" Type="Edm.String"
  sem:Mapping="?salesOrderItem boo:quantity_unit"/>
```

```
 <Property Name="Tax" Type="Edm.Decimal"
  sem:Mapping="?salesOrderItem boo:tax"/>
 <Property Name="NetSum" Type="Edm.Decimal"
  sem:Mapping="?salesOrderItem boo:net_sum"/>
 <Property Name="TotalSum" Type="Edm.Decimal"
  sem:Mapping="?salesOrderItem boo:total_sum"/>
 <Property Name="ProductName" Type="Edm.String"
  sem:Mapping="?salesOrderItem boo:product_name"/>
 <Property Name="ProductID" Type="Edm.String"
  sem:Mapping="?salesOrderItem boo:product_id"/>
 <Property Name="Position" Type="Edm.String"
  sem:Mapping="?salesOrderItem boo:position"/>
 <Property Name="SalesOrderID" Type="Edm.String"
  sem:Mapping="?salesOrderItem boo:sales_order_id"/>
 <Property Name="Currency" Type="Edm.String"
  sem:Mapping="?salesOrderItem boo:currency"/>
 <Property Name="Quantity" Type="Edm.Decimal"
  sem:Mapping="?salesOrderItem boo:quantity"/>
 <Property Name="DeliveryDate" Type="Edm.DateTime"
  sem:Mapping="?salesOrderItem boo:delivery_date"/>
 <NavigationProperty Name="Product" />
 <NavigationProperty Name="SalesOrderHeader" />
</EntityType>

<EntityType Name="Product"
 sem:Mapping="?product rdf:type boo:Product">
 <Key>
  <PropertyRef Name="ProductID"/>
 </Key>
 <Property Name="ProductID" Type="Edm.String"
  sem:Mapping="?product boo:id"/>
 <Property Name="TypeCode" Type="Edm.String"
  sem:Mapping="?product boo:type_code"/>
 <Property Name="Category" Type="Edm.String"
  sem:Mapping="?product boo:category $Category^^xsd:string"/>
 <Property Name="Name" Type="Edm.String"
  sem:Mapping="?product boo:name"/>
 <Property Name="Description" Type="Edm.String"
  sem:Mapping="?product boo:description $@en"/>
 <Property Name="SupplierID" Type="Edm.String"
  sem:Mapping="?product boo:supplier_id"/>
 <Property Name="SupplierName" Type="Edm.String"
  sem:Mapping="?product boo:supplier_name"/>
 <Property Name="TaxTarifCode" Type="Edm.Byte"
  sem:Mapping="?product boo:tax_tarif_code"/>
 <Property Name="MeasureUnit" Type="Edm.String"
  sem:Mapping="?product boo:measure_unit"/>
 <Property Name="WeightMeasure" Type="Edm.Decimal"
  sem:Mapping="?product boo:weight_measure"/>
 <Property Name="WeightUnit" Type="Edm.String"
  sem:Mapping="?product boo:weight_unit"/>
```

```
 <Property Name="Price" Type="Edm.Decimal"
  sem:Mapping="?product boo:price"/>
 <Property Name="CurrencyCode" Type="Edm.String"
  sem:Mapping="?product boo:currency_code"/>
 <Property Name="Width" Type="Edm.Decimal"
  sem:Mapping="?product boo:width"/>
 <Property Name="Depth" Type="Edm.Decimal"
  sem:Mapping="?product boo:depth"/>
 <Property Name="Height" Type="Edm.Decimal"
  sem:Mapping="?product boo:height"/>
 <Property Name="DimUnit" Type="Edm.String"
  sem:Mapping="?product boo:dim_unit"/>
 <Property Name="ProductPicUrl" Type="Edm.String"
  sem:Mapping="?product boo:product_pic_url"/>
 <Property Name="ProductPicUrlMimeType" Type="Edm.String"
  sem:Mapping="?product boo:product_pric_url_mime_type"/>
</EntityType>

<EntityType Name="BusinessPartner"
 sem:Mapping="?businessPartner rdf:type boo:BusinessPartner .
  ?businessPartner boo:sales_order ?salesOrder">
 <Key>
  <PropertyRef Name="BusinessPartnerID"/>
 </Key>
 <Property Name="BusinessPartnerID" Type="Edm.String"
  sem:Mapping="?businessPartner boo:business_partner_id .
  ?businessPartner boo:identifier . ?businessPartner boo:id"/>
 <Property Name="Role" Type="Edm.String"
  sem:Mapping="?businessPartner boo:role"/>
 <Property Name="EmailAddress" Type="Edm.String"
  sem:Mapping="?businessPartner boo:email_address"/>
 <Property Name="PhoneNumber" Type="Edm.String"
  sem:Mapping="?businessPartner boo:phone_number"/>
 <Property Name="FaxNumber" Type="Edm.String"
  sem:Mapping="?businessPartner boo:fax_number"/>
 <Property Name="WebAddress" Type="Edm.String"
  sem:Mapping="?businessPartner boo:web_address"/>
 <Property Name="CompanyName" Type="Edm.String"
  sem:Mapping="?businessPartner boo:company_name"/>
 <Property Name="LegalForm" Type="Edm.String"
  sem:Mapping="?businessPartner boo:legal_form"/>
 <Property Name="CurrencyCode" Type="Edm.String"
  sem:Mapping="?businessPartner boo:currency_code"/>
 <Property Name="City" Type="Edm.String"
  sem:Mapping="?businessPartner boo:city"/>
 <Property Name="PostalCode" Type="Edm.String"
  sem:Mapping="?businessPartner boo:postal_code"/>
 <Property Name="Street" Type="Edm.String"
  sem:Mapping="?businessPartner boo:street"/>
 <Property Name="Building" Type="Edm.String"
  sem:Mapping="?businessPartner boo:building"/>
```

```
 <Property Name="Country" Type="Edm.String"
  sem:Mapping="?businessPartner boo:country"/>
 <Property Name="AddressType" Type="Edm.String"
  sem:Mapping="?businessPartner boo:address_type"/>
 <Property Name="AddressValStartDate" Type="Edm.DateTime"
  sem:Mapping="?businessPartner boo:address_val_start_date"/>
 <Property Name="AddressValEndDate" Type="Edm.DateTime"
  sem:Mapping="?businessPartner boo:address_val_end_date"/>
 <Property Name="CreatedBy" Type="Edm.String"
  sem:Mapping="?businessPartner boo:created_by"/>
 <Property Name="CreatedAt" Type="Edm.DateTime"
  sem:Mapping="?businessPartner boo:created_at"/>
 <Property Name="ChangedBy" Type="Edm.String"
  sem:Mapping="?businessPartner boo:changed_by"/>
 <Property Name="ChangedAt" Type="Edm.DateTime"
  sem:Mapping="?businessPartner boo:changed_at"/>
 <NavigationProperty Name="SalesOrders" />
</EntityType>

<EntityContainer Name="DEMO_ERP"
 m:IsDefaultEntityContainer="true">
 <EntitySet Name="SalesOrders"
  EntityType="DEMO_ERP.SalesOrder" />
 <EntitySet Name="SalesOrderItems"
  EntityType="DEMO_ERP.SalesOrderItem" />
 <EntitySet Name="Products"
  EntityType="DEMO_ERP.Product" />
 <EntitySet Name="BusinessPartners"
  EntityType="DEMO_ERP.BusinessPartner"/>
</EntityContainer>

</Schema>
 </edmx:DataServices>
</edmx:Edmx>
```

F. Sales Order Service Test-Anfragen

1 Tripel-Pattern

```
SELECT ?tax
WHERE {
 <http://erp.beispiel.org/
  -541775059977034423569084860705877596 32/salesOrder#12412>
  boo:tax ?tax }
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/SalesOrders?$select=Tax,SalesOrderID&$filter=SalesOrderID eq '12412'*

2 Tripel-Patterns

```
SELECT ?customerName
WHERE {
 ?salesOrder rdf:type boo:SalesOrder .
 ?salesOrder boo:customer_name ?customerName }
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/SalesOrders?$select=SalesOrderID,CustomerName*

3 Tripel-Patterns

```
SELECT ?name
WHERE {
 ?product rdf:type boo:Product .
 ?product boo:category 'Notebooks'^^xsd:string .
 ?product boo:name ?name }
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/Products?$select=ProductID,Category,Name&$filter=Category eq 'Notebooks'*

4 Tripel-Patterns

```
SELECT *
WHERE {
 ?product rdf:type boo:Product .
 ?salesOrder boo:currency 'EUR' .
 ?businessPartner boo:sales_order ?salesOrder .
 ?businessPartner rdf:type boo:BusinessPartner }
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/BusinessPartners?$select=SalesOrders/SalesOrderID,BusinessPartnerID&$expand=SalesOrders*
- *https://test_erp_service.beispiel.org/SalesOrderService/Products?$select=ProductID*
- *https://test_erp_service.beispiel.org/SalesOrderService/SalesOrders?$select=Currency,SalesOrderID&$filter=Currency eq 'EUR'*

5 Tripel-Patterns

```
SELECT ?name ?description ?price ?currencyCode
WHERE {
?product rdf:type boo:Product .
?product boo:name ?name .
?product boo:description ?description .
?product boo:price ?price .
?product boo:currency_code ?currencyCode }
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/Products?$select=ProductID,Name,Description,Price,CurrencyCode*

Union

```
SELECT ?var5 ?var6 ?businessPartner ?position ?companyName
WHERE {
 {<http://erp.beispiel.org/
   -5417750599770344235690848607058775963 2/salesOrder#12412>
   ?var5 ?var6 .
  ?var6 boo:position ?position }
 UNION
 {?businessPartner ?var5 <http://erp.beispiel.org/
  -5417750599770344235690848607058775963 2/salesOrder#12412> .
   ?businessPartner boo:company_name ?companyName }}
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/BusinessPartners?$select=SalesOrders/SalesOrderID,BusinessPartnerID,CompanyName&$expand=SalesOrders*
- *https://test_erp_service.beispiel.org/SalesOrderService/SalesOrderItems?$select=Position,SalesOrderID*
- *https://test_erp_service.beispiel.org/SalesOrderService/SalesOrders?$select=SalesOrderItems/Position,SalesOrderID,SalesOrderItems/SalesOrderID&$expand=SalesOrderItems&$filter=SalesOrderID eq '12412'*

Optional

```
SELECT ?netSum ?quantity ?deliveryDate
WHERE {
 ?salesOrderItem rdf:type boo:SalesOrderItem .
 ?salesOrderItem boo:net_sum ?netSum .
 ?salesOrderItem boo:currency 'EUR' .
 ?salesOrderItem boo:quantity ?quantity .
 OPTIONAL { ?salesOrderItem boo:delivery_date ?deliveryDate .}
.}
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/SalesOrderItems?$select=DeliveryDate,Position,SalesOrderID*
- *https://test_erp_service.beispiel.org/SalesOrderService/SalesOrderItems?$select=NetSum,Currency,Quantity&$filter=Currency eq 'EUR'*

Filter

```
SELECT *
WHERE {
 <http://erp.beispiel.org/
  -541775059977034423569084860705877596 32/salesOrder#12412>
  ?var2 ?var1 .
 FILTER (?var2 = boo:net_sum || ?var2 = boo:tax) }
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/SalesOrders?$select=Tax,SalesOrderID,NetSum&$filter=SalesOrderID eq '12412'*

Union, Optional

```
SELECT *
WHERE {
 ?product rdf:type boo:Product .
 ?product boo:description ?description .
 ?salesOrderItem rdf:type boo:SalesOrderItem .
 {  ?salesOrderItem boo:product ?product . }
 UNION
 { ?salesOrderItem boo:ordered ?product . }
 { ?salesOrderItem boo:sales_order_header ?salesOrder }
 UNION
 { ?salesOrderItem boo:sales_order ?salesOrder }
 OPTIONAL { ?salesOrderItem boo:quantity ?quantity }
 OPTIONAL { ?salesOrderItem boo:delivery_date ?deliveryDate }}
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/SalesOrderItems?$select=DeliveryDate,Position,SalesOrderID,Quantity,Product/ProductID,SalesOrderHeader/SalesOrderID&$expand=Product,SalesOrderHeader*
- *https://test_erp_service.beispiel.org/SalesOrderService/Products?$select=ProductID,Description*

Union, Filter

```
SELECT *
WHERE {
 {?product rdf:type boo:Product .
  ?product boo:price ?price .
  FILTER (xsd:float(?price) > 20.0)}
 UNION
 {?product rdf:type boo:Product .
  ?product boo:price ?price .
  FILTER (xsd:float(?price) < 5.0)}}
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/Products?$select=ProductID,Price*

Optional, Filter

```
SELECT ?createdAt
WHERE {
 ?businessPartner boo:created_at ?createdAt .
 ?businessPartner rdf:type boo:BusinessPartner .
 ?businessPartner boo:country ?country .
 ?businessPartner boo:postal_code ?postalCode .
 ?businessPartner boo:city ?city .
 ?businessPartner boo:street ?street .
 ?businessPartner ?var33 ?var34 .
 OPTIONAL { ?businessPartner boo:address_type ?addressType .}
 FILTER (
  ?var33 = <http://services.odata.org/BusinessObjectOntology#
            business_partner_id> ||
 ?var33 = <http://services.odata.org/BusinessObjectOntology#
           identifier> ||
 ?var33 = <http://services.odata.org/BusinessObjectOntology#
          id> )
 FILTER (xsd:integer(?var34) > 30 ) .
 FILTER (xsd:integer(?postalCode) > 1000) .
 FILTER (?country = 'France'@en ||
 ?country = <http://test1.de/123/#France> ||
 ?country = <http://test2.de/123/#France>) }
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/BusinessPartners?$select=AddressType,BusinessPartnerID,CreatedAt,Country,PostalCode,City,Street*

Graph

```
SELECT *
FROM NAMED
<https://test_erp_service.beispiel.org/SalesOrderService>
WHERE {
 GRAPH ?service {
 ?product rdf:type boo:Product ;
          boo:description ?description ;
          boo:price ?price ;
          boo:width ?width ;
          boo:category ?category .
 { ?product boo:name 'Notebook ABC' }
 UNION
 { ?salesOrderItem boo:product ?product ;
                   boo:currency 'USD' . }
 OPTIONAL { ?product boo:depth ?depth }
 OPTIONAL { ?product boo:dim_unit ?dimUnit }
 OPTIONAL { ?product boo:supplier_name ?supplierName }
 OPTIONAL { ?product boo:weight_unit ?weightUnit }
 OPTIONAL { ?product boo:tax_tarif_code ?taxTarifCode }
  FILTER (langMatches( lang(?description), 'en') &&
langMatches( lang(?category), 'en') )}}
```

URLs:

- *https://test_erp_service.beispiel.org/SalesOrderService/SalesOrderItems?$select=Product/ProductID,Position,SalesOrderID,Currency&$expand=Product&$filter=Currency eq 'USD'*
- *https://test_erp_service.beispiel.org/SalesOrderService/Products?$select=Name&$filter=Name eq 'Notebook ABC'*
- *https://test_erp_service.beispiel.org/SalesOrderService/Products?$select=TaxTarifCode,ProductID,WeightUnit,SupplierName,DimUnit,Depth,Description,Price,Width,Category*

G. CD

Auf der beiliegenden CD befinden sich:

1. Die elektronische Fassung der Arbeit als PDF- und Word-Datei.
2. Die prototypische Implementierung des vorgestellten Verfahrens inklusive der Tests als Eclipse-Projekt.
3. Das semantisch erweiterte Service-Metadaten-Dokument des Northwind-Services.
4. Das semantisch erweiterte Service-Metadaten-Dokument des Sales Order Services.
5. Die formale Beschreibung der CSDL-Erweiterungen in XSD.

Publikationen

- M. Kirchhoff and K. Geihs. Querying SAP ERP with SPARQL. In *Proceedings of the 8th International Conference on Semantic Systems*, I-SEMANTICS ’12, pages 173–176, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1112-0. doi: 10.1145/2362499.2362525. URL http://doi.acm.org/10.1145/2362499.2362525.

- M. Kirchhoff, N. C. Liebau, and B. Schennerlein. Asset Information Management - Sicherer Zugriff auf verteilte Instandhaltungsdaten. *IT & Production*, April: 68–69, 2013.

- M. Kirchhoff and K. Geihs. Semantic description of OData services. In *Proceedings of the Fifth Workshop on Semantic Web Information Management*, SWIM ’13, pages 2:1–2:8, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2194-5. doi: http://dx.doi.org/10.1145/2484712.2484714. URL http://doi.acm.org/http://dx.doi.org/-10.1145/2484712.2484714.

- M. Kirchhoff and K. Geihs. Integrating OData Services into the Semantic Web: A SPARQL interface for OData. In *Proceedings of the 14th International Conference on Knowledge Management and Knowledge Technologies*, i-Know ’14, New York, NY, USA, to be published. ACM.

Literaturverzeichnis

[1] Standardized Technical Architecture Modeling - Conceptual and Design Level. Technical report, SAP AG, 2007. http://www.fmc-modeling.org/download/fmc-and-tam/SAP-TAM_Standard.pdf.

[2] 2011 erp report, 2011. Panorama Consulting Group. http://panorama-consulting.com/resource-center/2011-erp-report/.

[3] [MS-ODATA]: Open Data Protocol (OData) Specification V3, July 2012. URL http://-download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/-[MS-ODATA].pdf. Microsoft Corporation.

[4] Java Platform Standard Edition 7 Documentation, [Online; Zugegriffen am 03.04.2014]. http://docs.oracle.com/javase/7/docs/.

[5] Apache Jena, [Online; Zugegriffen am 03.04.2014]. https://jena.apache.org/.

[6] odata4j, [Online; Zugegriffen am 03.04.2014]. https://code.google.com/p/odata4j/.

[7] Xalan-Java, [Online; Zugegriffen am 03.04.2014]. http://xml.apache.org/xalan-j/.

[8] Enterprise Service Workplace, [Online; Zugegriffen am 09.04.2014]. http://esworkplace.sap.com/sdn.

[9] SAP Netweaver Gateway Demo System, [Online; Zugegriffen am 09.04.2014]. http://scn.sap.com/docs/DOC-31221.

[10] HermiT OWL Reasoner, [Online; Zugegriffen am 10.05.2014]. http://www.hermit-reasoner.com/.

[11] Stardog, [Online; Zugegriffen am 10.05.2014]. http://stardog.com/.

[12] Jena TDB, [Online; Zugegriffen am 11.05.2014]. http://jena.apache.org/documentation/tdb/.

[13] AllegroGraph, [Online; Zugegriffen am 13.05.2014]. http://franz.com/agraph/allegrograph/.

[14] Krextor, [Online; Zugegriffen am 20.05.2014]. http://trac.kwarc.info/krextor/.

[15] XML2RDF, [Online; Zugegriffen am 21.05.2014]. http://rhizomik.net/html/redefer/xml2rdf/.

[16] Virtuoso Sponger, [Online; Zugegriffen am 22.05.2014]. http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtSponger.

[17] ARQ - A SPARQL Processor for Jena, [Online; Zugegriffen am 23.05.2014]. http://jena.apache.org/documentation/query/index.html.

[18] D2RQ - Accessing Relational Databases as Virtual, [Online; Zugegriffen am 23.05.2014]. http://d2rq.org/.

[19] KAON2, [Online; Zugegriffen am 23.05.2014]. http://kaon2.semanticweb.org/.

[20] Ontobroker, [Online; Zugegriffen am 23.05.2014]. http://www.semafora-systems.com/de/produkte/ontobroker/.

[21] Pellet, [Online; Zugegriffen am 23.05.2014]. http://clarkparsia.com/pellet/.

[22] B. Adida, M. Birbeck, S. McCarron, and I. Herman. RDFa Core 1.1 Syntax and processing rules for embedding RDF through attributes. W3C Recommendation, W3C, June 2012. http://www.w3.org/TR/2012/REC-rdfa-core-20120607/.

[23] Aduna. openRDF.org ...home of Sesame, 2012. URL http://www.openrdf.org/. [Online;Zugegriffen am 20.11.2012].

[24] H. Alani, S. Dasmahapatra, N. Gibbins, H. Glaser, S. Harris, Y. Kalfoglou, K. O'Hara, and N. Shadbolt. Managing reference: Ensuring referential integrity of ontologies for the semantic web. In A. Gómez-Pérez and V. Benjamins, editors, *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, volume 2473 of *Lecture Notes in Computer Science*, pages 235–246. Springer Berlin / Heidelberg, 2002. ISBN 978-3-540-44268-4. URL http://dx.doi.org/10.1007/3-540-45810-7_29. 10.1007/3-540-45810-7_29.

[25] R. Alarcon and E. Wilde. Linking Data from RESTful Services. In *Third Workshop on Linked Data on the Web (LODW2010)*, 2010. URL http://events.linkeddata.org/ldow2010/-papers/ldow2010_paper10.pdf.

[26] R. Alarcón and E. Wilde. RESTler: crawling RESTful services. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 1051–1052, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772799. URL http://doi.acm.org/10.1145/1772690.1772799.

[27] R. Alarcón and E. Wilde. From RESTful Services to RDF: Connecting the Web and the Semantic Web. *CoRR*, abs/1006.2718, 2010.

[28] A. Alowisheq and D. E. Millard. EXPRESS: EXPressing REstful Semantic Services. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 03*, WI-IAT '09, pages 453–456, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3801-3. doi: 10.1109/WI-IAT.2009.324. URL http://dx.doi.org/10.1109/WI-IAT.2009.324.

[29] A. Alowisheq, D. E. Millard, and T. Tiropanis. EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 941–948, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-04929-3. doi: 10.1007/978-3-642-04930-9_59. URL http://-dx.doi.org/10.1007/978-3-642-04930-9_59.

[30] R. Angles and C. Gutierrez. The Expressive Power of SPARQL. In *Proceedings of the 7th International Conference on The Semantic Web*, ISWC '08, pages 114–129, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-88563-4. doi: 10.1007/978-3-540-88564-1_8. URL http://dx.doi.org/10.1007/978-3-540-88564-1_8.

[31] G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. MIT Press, Cambridge, MA, USA, 2004. ISBN 0262012103.

[32] M. Arenas and J. Pérez. Querying semantic web data with SPARQL. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '11, pages 305–316, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0660-7. doi: http://doi.acm.org/10.1145/1989284.1989312. URL http://doi.acm.org/10.1145/-1989284.1989312.

[33] M. Atre, V. Chaoji, M. J. Zaki, and J. A. Hendler. Matrix "bit" loaded: A scalable lightweight join query processor for rdf data. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 41–50, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772696. URL http://doi.acm.org/10.1145/-1772690.1772696.

[34] M. Barhamgi and D. Benslimane. Composing Data-Providing Web Services. In *VLDB09 (PhD Workshop)*, Aug. 2009. URL http://liris.cnrs.fr/publis/?id=4214.

[35] M. Barhamgi, D. Benslimane, and A. M. Ouksel. Composing and optimizing data providing web services. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 1141–1142, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-085-2. doi: 10.1145/1367497.1367696. URL http://doi.acm.org/10.1145/1367497.1367696.

[36] M. Barhamgi, D. Benslimane, and B. Medjahed. A Query Rewriting Approach for Web Service Composition. *IEEE Trans. Serv. Comput.*, 3 (3): 206–222, July 2010. ISSN 1939-1374. doi: 10.1109/TSC.2010.4. URL http://dx.doi.org/10.1109/TSC.2010.4.

[37] R. Battle and E. Benson. Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). *Web Semantics: Science, Services and Agents on the World Wide Web*, 6: 61–69, February 2008. ISSN 1570-8268. doi: 10.1016/j.websem.2007.11.002. URL http://dl.acm.org/citation.cfm?id=1346366.1346692.

[38] D. Beckett. RDF/XML Syntax Specification (Revised). W3C Recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[39] D. Beckett and J. Broekstra. SPARQL Query Results XML Format. W3C Recommendation, W3C, Jan. 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-XMLres-20080115/.

[40] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G. Carothers. Turtle Terse RDF Triple Language. W3C Candidate Recommendation, W3C, Feb. 2013. URL http://-www.w3.org/TR/2012/WD-turtle-20120710/. http://www.w3.org/TR/2013/CR-turtle-20130219/.

[41] T. Berners-Lee. Cool URIs don't change, 1998. URL http://www.w3.org/Provider/-Style/URI.

[42] T. Berners-Lee. Linked Data - Design Issues, 06 2009. URL http://www.w3.org/-DesignIssues/LinkedData. [Online; Zugriff: 12.10.2012].

[43] T. Berners-Lee and D. Connolly. Notation3 (N3): A readable RDF syntax. W3C Team Submission, W3C, March 2011. URL http://www.w3.org/TeamSubmission/2011/SUBM-n3-20110328/.

[44] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284 (5): 34–43, May 2001. URL http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[45] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), Jan. 2005. URL http://www.ietf.org/rfc/rfc3986.txt.

[46] T. Berners-lee, D. Connolly, L. Kagal, Y. Scharf, and J. Hendler. N3logic: A Logical Framework For The World Wide Web. *Theory Pract. Log. Program.*, 8 (3): 249–269, May 2008. ISSN 1471-0684. doi: 10.1017/S1471068407003213. URL http://dx.doi.org/10.1017/-S1471068407003213.

[47] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009. URL http://-tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf.

[48] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semant.*, 7 (3): 154–

165, Sept. 2009. ISSN 1570-8268. doi: 10.1016/j.websem.2009.07.002. URL http://-dx.doi.org/10.1016/j.websem.2009.07.002.
[49] C. Bizer, J. Volz, G. Kobilarov, and M. Gaedke. Silk - A Link Discovery Framework for the Web of Data. In *18th International World Wide Web Conference*, April 2009. URL http://www2009.eprints.org/227/.
[50] S. Boag, M. Kay, D. Chamberlin, A. Berglund, J. Simeon, J. Robie, and M. Fernandez. XML path language (XPath) 2.0 (second edition). W3C recommendation, W3C, Dec. 2010. http://www.w3.org/TR/2010/REC-xpath20-20101214/.
[51] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1247–1250, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-102-6. doi: 10.1145/1376616.1376746. URL http://doi.acm.org/10.1145/1376616.1376746.
[52] C. Bournez, K. Dubost, T. Guild, and Y. Lafon. Common HTTP Implementation Problems. W3C Note, W3C, Jan. 2003. http://www.w3.org/TR/2003/NOTE-chips-20030128/.
[53] C. Bussler, D. Fensel, and A. Maedche. A conceptual architecture for semantic web enabled web services. *SIGMOD Rec.*, 31 (4): 24–29, Dec. 2002. ISSN 0163-5808. doi: 10.1145/637411.637415. URL http://doi.acm.org/10.1145/637411.637415.
[54] C. Bussler, E. Cimpian, D. Fensel, J. M. Gomez, A. Haller, T. Haselwanter, M. Kerrigan, A. Mocan, M. Moran, E. Oren, B. Sapkota, I. Toma, J. Viskova, T. Vitvar, M. Zaremba, and M. Zaremba. Web Service Execution Environment (WSMX). W3C member submission, W3C, June 2005. http://www.w3.org/Submission/WSMX/.
[55] L. Cabral, J. Domingue, E. Motta, T. R. Payne, and F. Hakimpour. Approaches to Semantic Web Services: an Overview and Comparisons. In C. Bussler, J. Davies, D. Fensel, and R. Studer, editors, *ESWS*, volume 3053 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2004. ISBN 3-540-21999-4.
[56] J. J. Carroll and G. Klyne. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.
[57] D. Chamberlin, D. Florescu, M. Fernandez, J. Simeon, J. Robie, and S. Boag. XQuery 1.0: An XML Query Language (Second Edition). W3C recommendation, W3C, Dec. 2010. http://www.w3.org/TR/2010/REC-xquery-20101214/.
[58] L. Chepelev and M. Dumontier. Semantic Web integration of Cheminformatics resources with the SADI framework. *Journal of Cheminformatics*, 3 (1): 16+, 2011. ISSN 1758-2946. doi: 10.1186/1758-2946-3-16. URL http://dx.doi.org/10.1186/1758-2946-3-16.
[59] R. Chinnici, S. Weerawarana, J.-J. Moreau, and A. Ryman. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation, W3C, June 2007. http://www.w3.org/TR/2007/REC-wsdl20-20070626.
[60] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, W3C, Mar. 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.
[61] J. Clark. XSL Transformations (XSLT) Version 1.0. W3C recommendation, W3C, Nov. 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[62] K. G. Clark, E. Torres, and L. Feigenbaum. SPARQL Protocol for RDF. W3C Recommendation, W3C, Jan. 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-protocol-20080115/.
[63] D. Connolly. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). W3C recommendation, W3C, Sept. 2007. http://www.w3.org/TR/2007/REC-grddl-20070911/.
[64] D. Corkill. Blackboard Systems. *AI Expert*, 6 (9), January 1991. URL http://-mas.cs.umass.edu/paper/218.
[65] G. Correndo, M. Salvadores, I. Millard, H. Glaser, and N. Shadbolt. SPARQL query rewriting for implementing data integration over linked data. In *Proceedings of the 2010 EDBT/ICDT Workshops*, EDBT '10, pages 4:1–4:11, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-990-9. doi: http://doi.acm.org/10.1145/1754239.1754244. URL http://-doi.acm.org/10.1145/1754239.1754244.
[66] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006. URL http://www.ietf.org/rfc/rfc4627.txt.
[67] R. Cyganiak and C. Bizer. Pubby - A Linked Data Frontend for SPARQL Endpoints. URL http://www4.wiwiss.fu-berlin.de/pubby/. Zugriff am 10.10.2012.
[68] R. Cyganiak and L. Sauermann. Cool URIs for the Semantic Web. W3C Note, W3C, Dec. 2008. http://www.w3.org/TR/2008/NOTE-cooluris-20081203/.
[69] R. Cyganiak and D. Wood. RDF 1.1 Concepts and Abstract Syntax. W3C Working Draft, W3C, June 2012. URL http://www.w3.org/TR/2012/WD-rdf11-concepts-20120605/.
[70] R. Cyganiak, S. Sundara, and S. Das. R2RML: RDB to RDF Mapping Language. W3C recommendation, W3C, Sept. 2012. http://www.w3.org/TR/2012/REC-r2rml-20120927/.
[71] R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, Feb. 2014. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.
[72] J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, U. Keller, M. Kifer, B. König-Ries, J. Kopecky, R. Lara, H. Lausen, E. Oren, A. Polleres, D. Roman, J. Scicluna, and M. Stollberg. Web Service Modeling Ontology (WSMO). W3C member submission, W3C, June 2005. http://www.w3.org/Submission/2005/SUBM-WSMO-20050603/.
[73] J. de Bruijn, D. Fensel, U. Keller, M. Kifer, H. Lausen, R. Krummenacher, A. Polleres, and L. Predoiu. Web Service Modeling Language (WSML). W3C member submission, W3C, June 2005. http://www.w3.org/Submission/2005/SUBM-WSML-20050603/.
[74] M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs). RFC 3987 (Proposed Standard), Jan. 2005. URL http://www.ietf.org/rfc/rfc3987.txt.
[75] L. Dusseault and J. Snell. PATCH Method for HTTP. RFC 5789 (Proposed Standard), Mar. 2010. URL http://www.ietf.org/rfc/rfc5789.txt.
[76] O. Erling and I. Mikhailov. Rdf support in the virtuoso dbms. In T. Pellegrini, S. Auer, K. Tochtermann, and S. Schaffert, editors, *Networked Knowledge - Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 7–24. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-02183-1. doi: 10.1007/978-3-642-02184-8_2. URL http://dx.doi.org/-10.1007/978-3-642-02184-8_2.

[77] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1 (2): 113 – 137, 2002. ISSN 1567-4223. doi: 10.1016/S1567-4223(02)00015-7. URL http://www.sciencedirect.com/science/article/pii/-S1567422302000157.

[78] D. Fensel, C. Bussler, and A. Maedche. Semantic web enabled web services. In I. Horrocks and J. Hendler, editors, *The Semantic Web — ISWC 2002*, volume 2342 of *Lecture Notes in Computer Science*, pages 1–2. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-43760-4. doi: 10.1007/3-540-48005-6_1. URL http://dx.doi.org/10.1007/3-540-48005-6_1.

[79] D. Fensel, F. Fischer, J. Kopecký, R. Krummenacher, D. Lambert, and T. Vitvar. WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web. W3C Member Submission, W3C, Aug. 2010. http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/.

[80] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. URL http://-www.ietf.org/rfc/rfc2616.txt. Updated by RFCs 2817, 5785, 6266, 6585.

[81] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. AAI9980887.

[82] R. T. Fielding and R. N. Taylor. Principled design of the modern Web architecture. *ACM Trans. Internet Technol.*, 2 (2): 115–150, May 2002. ISSN 1533-5399. doi: 10.1145/514183.514185. URL http://dx.doi.org/10.1145/514183.514185.

[83] P. Gearon, A. Passant, and A. Polleres. SPARQL 1.1 Update. W3C Working Draft, W3C, Jan. 2012. http://www.w3.org/TR/2012/WD-sparql11-update-20120105/.

[84] B. Glimm. Using SPARQL with RDFS and OWL Entailment. In A. Polleres, C. d'Amato, M. Arenas, S. Handschuh, P. Kroner, S. Ossowski, and P. Patel-Schneider, editors, *Reasoning Web. Semantic Technologies for the Web of Data*, volume 6848 of *Lecture Notes in Computer Science*, pages 137–201. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-23031-8. doi: 10.1007/978-3-642-23032-5_3. URL http://dx.doi.org/10.1007/978-3-642-23032-5_3.

[85] B. Glimm and M. Krötzsch. SPARQL beyond subgraph matching. In *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I*, ISWC'10, pages 241–256, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-17745-X, 978-3-642-17745-3. URL http://dl.acm.org/citation.cfm?id=1940281.1940298.

[86] J. Gregorio. Do we need WADL? URL http://bitworking.org/news/193/Do-we-need-WADL. [Zugriff am 24.09.2012].

[87] J. Gregorio and B. de hOra. The Atom Publishing Protocol. RFC 5023 (Proposed Standard), Oct. 2007. URL http://www.ietf.org/rfc/rfc5023.txt. http://www.ietf.org/rfc/rfc5023.txt.

[88] R. V. Guha and D. Brickley. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[89] M. J. Hadley. Web Application Description Language (WADL). Technical report, Sun Microsystems Inc., 2009. URL http://java.net/projects/wadl/sources/svn/content/trunk/www/-wadl20090202.pdf.

[90] R. Handl, G. Krause, M. Lacasse, and M. Zurmühl. OData for SAP Products - Version 2.02. Technical report, SAP AG, 2013. Internal / Confidential.
[91] S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, Mar. 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.
[92] S. Harris, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 Query Language. W3C Working Draft, W3C, Jan. 2012. http://www.w3.org/TR/2012/WD-sparql11-query-20120105/.
[93] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich. Data summaries for on-demand queries over Linked Data. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 411–420, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi: http://doi.acm.org/10.1145/1772690.1772733. URL http://doi.acm.org/10.1145/1772690.1772733.
[94] O. Hartig and R. Heese. The sparql query graph model for query optimization. In *Proceedings of the 4th European Conference on The Semantic Web: Research and Applications*, ESWC '07, pages 564–578, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72666-1. doi: 10.1007/978-3-540-72667-8_40. URL http://dx.doi.org/10.1007/-978-3-540-72667-8_40.
[95] O. Hartig, C. Bizer, and J.-C. Freytag. Executing SPARQL Queries over the Web of Linked Data. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 293–309, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-04929-3. doi: http://dx.doi.org/10.1007/978-3-642-04930-9_19. URL http://dx.doi.org/10.1007/978-3-642-04930-9_19.
[96] P. Hayes. RDF Semantics. W3C Recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.
[97] P. Hayes, P. F. Patel-Schneider, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-owl-semantics-20040210/.
[98] M. Hert, G. Reif, and H. C. Gall. A comparison of rdb-to-rdf mapping languages. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics '11, pages 25–32, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0621-8. doi: 10.1145/2063518.2063522. URL http://doi.acm.org/10.1145/2063518.2063522.
[99] P. Hitzler, M. Krötzsch, S. Rudolph, and Y. Sure. *Semantic Web: Grundlagen (eXamen.press)*. Springer, 1 edition, Oct. 2007. ISBN 3540339930.
[100] IANA. MIME Media Types, 06 2012. URL http://www.iana.org/assignments/media-types/index.html.
[101] Y. Jing, D. Jeong, and D.-K. Baik. SPARQL graph pattern rewriting for OWL-DL inference queries. *Knowledge and Information Systems*, 20: 243–262, 2009. ISSN 0219-1377. doi: 10.1007/s10115-008-0169-8. URL http://dx.doi.org/10.1007/s10115-008-0169-8.
[102] S. Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648 (Proposed Standard), Oct. 2006. URL http://www.ietf.org/rfc/rfc4648.txt.
[103] R. Khare and T. Çelik. Microformats: a pragmatic path to the semantic web. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 865–866, New York, NY, USA, 2006. ACM. ISBN 1-59593-323-9. doi: 10.1145/1135777.1135917. URL http://doi.acm.org/10.1145/1135777.1135917.

[104] M. Kirchhoff. Konzept und prototypische Implementierung eines ontologie-basierten Anfragesystems für die Modellierung der Prozess-Integration eines Services in eine erweiterbare Geschäftsanwendung. Master's thesis, Universität Kassel, 2011.
[105] M. Kirchhoff and K. Geihs. Querying SAP ERP with SPARQL. In *Proceedings of the 8th International Conference on Semantic Systems*, I-SEMANTICS '12, pages 173–176, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1112-0. doi: 10.1145/2362499.2362525. URL http://doi.acm.org/10.1145/2362499.2362525.
[106] M. Kirchhoff and K. Geihs. Semantic description of OData services. In *Proceedings of the Fifth Workshop on Semantic Web Information Management*, SWIM '13, pages 2:1–2:8, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2194-5. doi: http://dx.doi.org/10.1145/2484712.2484714. URL http://doi.acm.org/http://dx.doi.org/-10.1145/2484712.2484714.
[107] M. Kirchhoff and K. Geihs. Integrating OData Services into the Semantic Web: A SPARQL interface for OData. In *Proceedings of the 14th International Conference on Knowledge Management and Knowledge Technologies*, i-Know '14, New York, NY, USA, to be published. ACM.
[108] M. Kirchhoff, N. C. Liebau, and B. Schennerlein. Asset Information Management - Sicherer Zugriff auf verteilte Instandhaltungsdaten. *IT & Production*, April: 68–69, 2013.
[109] J. Kopecky and T. Vitvar. WSMO-Lite: Lowering the SemanticWeb Services Barrier with Modular and Light-weight Annotations. In *Semantic Computing, 2008 IEEE International Conference on*, pages 238–244. IEEE, 2008.
[110] J. Kopecký, K. Gomadam, and T. Vitvar. hRESTS: An HTML Microformat for Describing RESTful Web Services. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '08, pages 619–625, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3496-1. doi: 10.1109/WIIAT.2008.379. URL http://dx.doi.org/10.1109/WIIAT.2008.379.
[111] J. Kopecky, T. Vitvar, D. Fensel, and K. Gomadam. hRESTS & MicroWSMO. Technical report, Technical report, available at http://cms-wg. sti2. org/TR/d12, 2009.
[112] R. Krummenacher, B. Norton, and A. Marte. Towards linked open services and processes. In *Proceedings of the Third future internet conference on Future internet*, FIS'10, pages 68–77, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-15876-5, 978-3-642-15876-6. URL http://dl.acm.org/citation.cfm?id=1929268.1929276.
[113] G. Ladwig and T. Tran. Linked data query processing strategies. In *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I*, ISWC'10, pages 453–469, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-17745-X, 978-3-642-17745-3. URL http://dl.acm.org/citation.cfm?id=1940281.1940311.
[114] A. Langegger. Virtual data integration on the web: novel methods for accessing heterogeneous and distributed data with rich semantics. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '08, pages 559–562, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-349-5. doi: 10.1145/1497308.1497410. URL http://doi.acm.org/10.1145/1497308.1497410.
[115] A. Langegger and W. Wöß. SemWIQ - Semantic Web Integrator and Query Engine. In *3rd International Applications of Semantic Web Workshop (AST'08)*, volume Informatik 2008

of *Lecture Notes in Informatics (LNI) 133 (Band 1) 134 (Band 2)*. Gesellschaft für Informatik e.V. (GI), 2008. ISBN 978-3-88579-227-7.
[116] A. Langegger, W. Wöß, and M. Blöchl. A semantic web middleware for virtual data integration on the web. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, pages 493–507, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-68233-3, 978-3-540-68233-2. URL http://dl.acm.org/-citation.cfm?id=1789394.1789441.
[117] M. Lanthaler and C. Gutl. A semantic description language for RESTful Data Services to combat Semaphobia. In *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference on*, pages 47 –53, 31 2011-june 3 2011. doi: 10.1109/DEST.2011.5936597.
[118] M. Lanthaler and C. Gutl. Aligning Web Services with the Semantic Web to Create a Global Read-Write Graph of Data. In *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*, pages 15 –22, sept. 2011. doi: 10.1109/ECOWS.2011.17.
[119] M. Lanthaler and C. Gütl. Seamless integration of RESTful services into the web of data. *Advances in Multimedia - Special issue on Web Services in Multimedia Communication*, 2012: 1:1–1:14, Jan. 2012. ISSN 1687-5680. doi: 10.1155/2012/586542. URL http://-dx.doi.org/10.1155/2012/586542.
[120] M. Lanthaler, M. Granitzer, and C. Gütl. Semantic Web Services: State of the Art. *Proceedings of the IADIS International Conference on Internet Technologies Society ITS 2010*, pages 107–114, 2010.
[121] J. Lathem, K. Gomadam, and A. Sheth. SA-REST and (S)mashups : Adding Semantics to RESTful Services. In *Semantic Computing, 2007. ICSC 2007. International Conference on*, pages 469–476, 2007. doi: 10.1109/ICSC.2007.94.
[122] H. Lausen and J. Farrell. Semantic Annotations for WSDL and XML Schema. W3C Recommendation, W3C, Aug. 2007. http://www.w3.org/TR/2007/REC-sawsdl-20070828/.
[123] C. K. Liu and D. Booth. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. W3C Recommendation, W3C, June 2007. http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626.
[124] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz. Reference model for service oriented architecture 1.0. Standard, OASIS, 2006. http://docs.oasis-open.org/soa-rm/v1.0/.
[125] A. Maedche and B. Motik. Repräsentations- und Anfragesprachen für Ontologien - eine Übersicht. *Datenbank-Spektrum*, 6: 43–53, 2003.
[126] K. Makris, N. Bikakis, N. Gioldasis, C. Tsinaraki, and S. Christodoulakis. Towards a Mediator Based on OWL and SPARQL. In *Proceedings of the 2nd World Summit on the Knowledge Society: Visioning and Engineering the Knowledge Society. A Web Science Perspective*, WSKS '09, pages 326–335, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-04753-4. doi: http://dx.doi.org/10.1007/978-3-642-04754-1_34. URL http://-dx.doi.org/10.1007/978-3-642-04754-1_34.
[127] K. Makris, N. Gioldasis, N. Bikakis, and S. Christodoulakis. Ontology mapping and SPARQL rewriting for querying federated RDF data sources. In *Proceedings of the 2010 international conference on On the move to meaningful internet systems: Part II*, OTM'10,

pages 1108–1117, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16948-1, 978-3-642-16948-9. URL http://dl.acm.org/citation.cfm?id=1926129.1926173.
[128] K. Makris, N. Bikakis, N. Gioldasis, and S. Christodoulakis. SPARQL-RW: transparent query access over mapped RDF data sources. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 610–613, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0790-1. doi: 10.1145/2247596.2247678. URL http://doi.acm.org/10.1145/2247596.2247678.
[129] M. Maleshkova, C. Pedrinaci, and J. Domingue. Supporting the creation of semantic RESTful service descriptions. In *3rd International SMR2 2009 Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web, collocated with the 8th International Semantic Web Conference (ISWC-2009)*, 2009. URL http://oro.open.ac.uk/-23106/.
[130] A. Malhotra and P. V. Biron. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation, W3C, Oct. 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.
[131] A. Malhotra, J. Melton, N. Walsh, and M. Kay. XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition). W3C Recommendation, W3C, Dec. 2012. http://www.w3.org/TR/2010/REC-xpath-functions-20101214/.
[132] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. W3C Member Submission, W3C, Nov. 2004. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.
[133] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16 (2): 46–53, Mar. 2001. ISSN 1541-1672. doi: 10.1109/5254.920599. URL http://-dx.doi.org/10.1109/5254.920599.
[134] Microsoft. Entity data model. URL http://msdn.microsoft.com/de-de/library/vstudio/-ee382825.aspx. [Online; Zugegriffen am 15.11.2012].
[135] Microsoft Corporation. Northwind-OData-Service Metadata Document, . http://services.odata.org/V2/Northwind/Northwind.svc/$metadata.
[136] Microsoft Corporation. Northwind-OData-Service V2, . http://services.odata.org/V2/Northwind/Northwind.svc/.
[137] Microsoft Corporation. [MS-ODATA]: Open Data Protocol (OData) Specification V2, June 2011. http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-ODATA].pdf.
[138] Microsoft Corporation. [MC-CSDL]: Conceptual Schema Definition File Format Version 2.0, June 2011. http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MC-CSDL].pdf.
[139] Microsoft Corporation. [MC-EDMX]: Entity Data Model for Data Services Packaging Format, June 2011. http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MC-EDMX].pdf.
[140] E. Miller and F. Manola. RDF Primer. W3C Recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[141] K. Mohebbi, S. Ibrahim, and N. Idris. Contemporary Semantic Web Service Frameworks: An Overview and Comparisons, 3 (3), 65-76. *International Journal on Web Service Computing (IJWSC)*, 2012.
[142] M. Morsey, J. Lehmann, S. Auer, and A.-C. N. Ngomo. Dbpedia sparql benchmark: Performance assessment with real queries on real data. In *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, ISWC'11, pages 454–469, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-25072-9. URL http://dl.acm.org/-citation.cfm?id=2063016.2063046.
[143] T. Neumann and G. Weikum. The rdf-3x engine for scalable management of rdf data. *The VLDB Journal*, 19 (1): 91–113, Feb. 2010. ISSN 1066-8888. doi: 10.1007/s00778-009-0165-y. URL http://dx.doi.org/10.1007/s00778-009-0165-y.
[144] B. Norton and R. Krummenacher. Consuming Dynamic Linked Data. In *1st International Workshop on Consuming Linked Data*, 2010.
[145] M. Nottingham and R. Sayre. The Atom Syndication Format. RFC 4287 (Proposed Standard), Dec. 2005. URL http://www.ietf.org/rfc/rfc4287.txt. Updated by RFC 5988.
[146] T. Ottmann and P. Widmayer. *Algorithmen und Datenstrukturen (German Edition)*. Spektrum Akademischer Verlag, 2002. ISBN 3827410290.
[147] P. Patel-Schneider and P. Hayes. RDF 1.1 semantics. W3C recommendation, W3C, Feb. 2014. http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/.
[148] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34: 16:1–16:45, September 2009. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/1567274.1567278. URL http://doi.acm.org/10.1145/-1567274.1567278.
[149] A. Phillips and M. Davis. Tags for Identifying Languages. RFC 5646 (Best Current Practice), Sept. 2009. URL http://www.ietf.org/rfc/rfc5646.txt.
[150] M. Pizzo, R. Handl, and M. Zurmuehl. OData Version 4.0 Part 3: Common Schema Definition Language (CSDL). OASIS standard, OASIS, Feb. 2014. http://docs.oasis-open.org/odata/odata/v4.0/os/part3-csdl/odata-v4.0-os-part3-csdl.pdf.
[151] M. Pizzo, R. Handl, and M. Zurmuehl. OData Version 4.0 Part 1: Protocol. OASIS standard, OASIS, Feb. 2014. http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.pdf.
[152] M. Pizzo, R. Handl, and M. Zurmuehl. OData Version 4.0 Part 2: URL Conventions. OASIS standard, OASIS, Feb. 2014. http://docs.oasis-open.org/odata/odata/v4.0/os/part2-url-conventions/odata-v4.0-os-part2-url-conventions.pdf.
[153] N. Preda, F. M. Suchanek, G. Kasneci, T. Neumann, M. Ramanath, and G. Weikum. ANGIE: active knowledge for interactive exploration. *Proc. VLDB Endow.*, 2 (2): 1570–1573, Aug. 2009. ISSN 2150-8097. URL http://dl.acm.org/citation.cfm?id=1687553.1687594.
[154] N. Preda, G. Kasneci, F. M. Suchanek, T. Neumann, W. Yuan, and G. Weikum. Active knowledge: dynamically enriching RDF knowledge bases by web services. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 399–410, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0032-2. doi: 10.1145/1807167.1807212. URL http://doi.acm.org/10.1145/1807167.1807212.

[155] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, W3C, Jan. 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.
[156] B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, pages 524–538, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-68233-3, 978-3-540-68233-2. URL http://dl.acm.org/-citation.cfm?id=1789394.1789443.
[157] J. Rao and X. Su. A Survey of Automated Web Service Composition Methods. In J. Cardoso and A. Sheth, editors, *Semantic Web Services and Web Process Composition*, volume 3387 of *Lecture Notes in Computer Science*, pages 43–54. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-24328-1. URL http://dx.doi.org/10.1007/978-3-540-30581-1_5. 10.1007/978-3-540-30581-1_5.
[158] Raytheon BBN Technologies. Asio Tool Suite. URL http://bbn.com/technology/-knowledge/asio_tool_suite. Abgerufen am 17.09.2012.
[159] A. Riazanov, J. Laurila, and C. Baker. Deploying mutation impact text-mining software with the SADI Semantic Web Services framework. *BMC Bioinformatics*, 12 (Suppl 4): S6+, 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-S4-S6. URL http://dx.doi.org/-10.1186/1471-2105-12-S4-S6.
[160] L. Richardson and S. Ruby. *Restful web services*. O'Reilly, first edition, 2007. ISBN 9780596529260.
[161] A. Rodriguez. RESTful Web services: The basics, Nov. 2008. URL http://-www.ibm.com/developerworks/webservices/library/ws-restful/ws-restful-pdf.pdf.
[162] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Appl. Ontol.*, 1 (1): 77–106, Jan. 2005. ISSN 1570-5838. URL http://dl.acm.org/citation.cfm?id=1412350.1412357.
[163] RSS Advisory Board. RSS 2.0 Specification, Mar. 2009. URL http://-www.rssboard.org/rss-specification.
[164] S. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. T. Jr, S. Auer, J. Sequeda, and A. Ezzat. A survey of current approaches for mapping of relational databases to rdf, 01 2009. URL http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf.
[165] M. Salvadores, G. Correndo, B. Rodriguez-Castro, N. Gibbins, J. Darlington, and N. R. Shadbolt. LinksB2N: Automatic Data Integration for the Semantic Web. In *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part II*, OTM '09, pages 1121–1138, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-05150-0. doi: 10.1007/978-3-642-05151-7_27. URL http://dx.doi.org/10.1007/978-3-642-05151-7_27.
[166] M. Schmidt, M. Meier, and G. Lausen. Foundations of SPARQL query optimization. In *Proceedings of the 13th International Conference on Database Theory*, ICDT '10, pages 4–33, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-947-3. doi: http://doi.acm.org/10.1145/1804669.1804675. URL http://doi.acm.org/10.1145/-1804669.1804675.

[167] J. Sequeda, E. Prud'hommeaux, A. Bertails, and M. Arenas. A Direct Mapping of Relational Data to RDF. W3C recommendation, W3C, Sept. 2012. http://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/.
[168] J. F. Sequeda, S. H. Tirmizi, O. Corcho, and D. P. Miranker. Survey of directly mapping sql databases to the semantic web. *The Knowledge Engineering Review*, 26: 445–486, 12 2011. ISSN 1469-8005. doi: 10.1017/S0269888911000208. URL http://-journals.cambridge.org/article_S0269888911000208.
[169] J. F. Sequeda, M. Arenas, and D. P. Miranker. On directly mapping relational databases to rdf and owl. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 649–658, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1229-5. doi: 10.1145/2187836.2187924. URL http://doi.acm.org/10.1145/2187836.2187924.
[170] A. P. Sheth, K. Gomadam, and J. Lathem. SA-REST: Semantically Interoperable and Easier-To-Use Services and Mashups. *Internet Computing, IEEE*, 11 (6): 91–94, 2007.
[171] S. Speiser and A. Harth. Towards Linked Data Services. In A. Polleres and H. Chen, editors, *ISWC Posters&Demos*, volume 658 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.
[172] S. Speiser and A. Harth. Taking the LIDS off data silos. In *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, pages 44:1–44:4, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0014-8. doi: http://doi.acm.org/10.1145/1839707.1839761. URL http://doi.acm.org/10.1145/-1839707.1839761.
[173] S. Speiser and A. Harth. Integrating linked data and services with linked data services. In *Proceedings of the 8th extended semantic web conference on The semantic web: research and applications - Volume Part I*, ESWC'11, pages 170–184, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-21033-4. URL http://dl.acm.org/-citation.cfm?id=2008892.2008907.
[174] M. Sporny. RDFa Lite 1.1. W3C Recommendation, W3C, June 2012. http://www.w3.org/TR/2012/REC-rdfa-lite-20120607/.
[175] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds. SPARQL basic graph pattern optimization using selectivity estimation. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 595–604, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-085-2. doi: 10.1145/1367497.1367578. URL http://-doi.acm.org/10.1145/1367497.1367578.
[176] M. Stollberg, C. Feier, D. Roman, and D. Fensel. Semantic Web Services - Concepts and Technology. In *Language Technology, Ontologies, and the Semantic Web*. Kluwer Publishers, 2006.
[177] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7. doi: 10.1145/1242572.1242667. URL http://doi.acm.org/10.1145/1242572.1242667.
[178] The Apache Software Foundation. Apache Jena, Dec. 2011. URL http://-jena.apache.org/. [Zugriff: 16.10.2012].
[179] The W3C SPARQL Working Group. SPARQL 1.1 Overview. W3C recommendation, W3C, Mar. 2013. http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/.

[180] Unicode Consortium. *The Unicode standard, version 3.0 / the Unicode Consortium*. Addison-Wesley, Reading, Mass. :, 2000. ISBN 0201616335.

[181] B. Vandervalk. The SHARE System - A Semantic Web Based Approach for Evaluating Queries Across Distributed Bioinformatics Databases and Software. Master's thesis, University of British Columbia, Apr. 2011.

[182] B. Vandervalk, E. McCarthy, and M. Wilkinson. SHARE: A Semantic Web Query Engine for Bioinformatics. In A. Gomez-Perez, Y. Yu, and Y. Ding, editors, *The Semantic Web*, volume 5926 of *Lecture Notes in Computer Science*, pages 367–369. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-10870-9. URL http://dx.doi.org/10.1007/978-3-642-10871-6_27. 10.1007/978-3-642-10871-6_27.

[183] B. P. Vandervalk, E. L. McCarthy, and M. D. Wilkinson. Share: A web service based framework for distributed querying and reasoning on the semantic web. *CoRR*, abs/1305.4455, 2013.

[184] R. Verborgh and R. Van de Walle. Application of semantic web technologies for multimedia interpretation. In *Proceedings of the 20th international conference companion on World Wide Web*, WWW '11, pages 427–432, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0637-9. doi: 10.1145/1963192.1963355. URL http://doi.acm.org/10.1145/-1963192.1963355.

[185] R. Verborgh, D. Van Deursen, J. De Roo, E. Mannens, and R. Van de Walle. SPARQL Endpoints as Front-end for Multimedia Processing Algorithms. In *Proceedings of the Fourth International Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web at the 9th International Semantic Web Conference (ISWC 2010)*, Nov. 2010. URL http://ceur-ws.org/Vol-667/smr22010_submission_4.pdf.

[186] R. Verborgh, D. Deursen, E. Mannens, C. Poppe, and R. Walle. Enabling context-aware multimedia annotation by a novel generic semantic problem-solving platform. *Multimedia Tools and Applications*, 61: 105–129, 2012. ISSN 1380-7501. doi: 10.1007/s11042-010-0709-6. URL http://dx.doi.org/10.1007/s11042-010-0709-6.

[187] T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel. WSMO-lite annotations for web services. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, pages 674–689, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-68233-3, 978-3-540-68233-2. URL http://dl.acm.org/-citation.cfm?id=1789394.1789455.

[188] W3C. Linked data. http://www.w3.org/standards/semanticweb/data. URL http://-www.w3.org/standards/semanticweb/data. [Online; Zugriff: 02 November, 2011].

[189] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25 (3): 38–49, Mar. 1992. ISSN 0018-9162. doi: 10.1109/2.121508. URL http://-dx.doi.org/10.1109/2.121508.

[190] M. Wilkinson, L. McCarthy, B. Vandervalk, D. Withers, E. Kawas, and S. Samadian. SADI, SHARE, and the in silico scientific method. *BMC Bioinformatics*, 11 (Suppl 12): S7+, 2010. ISSN 1471-2105. doi: 10.1186/1471-2105-11-S12-S7. URL http://dx.doi.org/10.1186/-1471-2105-11-S12-S7.

[191] M. Wilkinson, B. Vandervalk, and L. McCarthy. The Semantic Automated Discovery and Integration (SADI) Web service Design-Pattern, API and Reference Implementation.

Journal of Biomedical Semantics, 2: 1–23, 2011. doi: 10.1186/2041-1480-2-8. URL http://-dx.doi.org/10.1186/2041-1480-2-8.

[192] M. D. Wilkinson, B. P. Vandervalk, and E. L. McCarthy. SADI Semantic Web Services – 'cause you can't always GET what you want! In M. Kirchberg, P. C. K. Hung, B. Carminati, C.-H. Chi, R. Kanagasabai, E. D. Valle, K.-C. Lan, and L.-J. Chen, editors, *APSCC*, pages 13–18. IEEE, 2009. ISBN 978-1-4244-5336-8.

[193] G. T. Williams. SPARQL 1.1 Service Description. W3C Working Draft, W3C, Oct. 2009. http://www.w3.org/TR/2009/WD-sparql11-service-description-20091022/.

[194] K. Zeng, J. Yang, H. Wang, B. Shao, and Z. Wang. A distributed graph engine for web scale rdf data. *Proc. VLDB Endow.*, 6 (4): 265–276, Feb. 2013. ISSN 2150-8097. URL http://-dl.acm.org/citation.cfm?id=2535570.2488333.

[195] W. Zhao, X. Meng, J. Chen, and C. Liu. Integrating Information-Providing Web Services into the Data Integration System. In *Proceedings of the 2008 IEEE International Conference on Web Services*, ICWS '08, pages 801–802, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3310-0. doi: 10.1109/ICWS.2008.63. URL http://-dx.doi.org/10.1109/ICWS.2008.63.

[196] W. Zhao, X. Meng, J. Chen, and C. Liu. Unify the description of Web services and RDF data sources towards uniform data integration. *IET Conference Publications*, 2008 (CP544): 676–680, 2008. doi: 10.1049/cp:20080895. URL http://link.aip.org/link/abstract/-IEECPS/v2008/iCP544/p676/s1.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Dissertation selbstständig, ohne unerlaubte Hilfe Dritter angefertigt und andere als die in der Dissertation angegebenen Hilfsmittel nicht benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, habe ich als solche kenntlich gemacht. Dritte waren an der inhaltlich-materiellen Erstellung der Dissertation nicht beteiligt; insbesondere habe ich hierfür nicht die Hilfe eines Promotionsberaters in Anspruch genommen. Kein Teil dieser Arbeit ist in einem anderen Promotions- oder Habilitationsverfahren verwendet worden.

Dresden, im Juni 2014

Marc Kirchhoff