TUHH
Hamburg University of Technology

Diplomarbeit

Khristianov Nikita
Matr.-Nr. 22736

Architekturen zur Analyse von strombasierten Daten

01.04.2013

Erstgutachter: Prof. Dr. rer. nat. habil. Ralf Möller
Zweitgutachter: Prof. Dr. Karl-Heinz Zimmermann

Hamburg University of Technology (TUHH)
Technische Universität Hamburg-Harburg
Institute for Software Systems
21073 Hamburg

TUHH STS Software Technology Systems

Eidesstattliche Erklärung

Ich, Nikita Khristianov, Matrikel-Nr. 22736, versichere hiermit, dass ich meine Diplomarbeit mit dem Thema

Architekturen zur Analyse von strombasierten Daten

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Alle Abbildungen in dieser Arbeit sind eigene Darstellungen. Wo Abbildungen übernommen und abgeändert wurden, ist die Quelle der ursprünglichen Abbildung angegeben.
Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Hamburg, den 1. April 2013

Nikita Khristianov

Abstract

Im Rahmen dieser Arbeit wurden Architekturen zur Analyse strombasierter RDF-Daten untersucht. Es wurde über die aktuellen Systeme zur Verarbeitung von Anfragen gegen RDF-Datenströme recherchiert. Die Architekturen und die innerhalb der Systeme eingesetzten Erweiterungen der SPARQL-Anfragesprachen von C-SPARQL, SPARQL$_{stream}$ und CQELS wurden untersucht. Des Weiteren wurden folgende für die Auswertung solcher Systeme relevante Kriterien zusammengefasst:

- **Umfang der unterstützten Funktionalität** Hierbei wurden die Unterstützung der Konstrukte der SPARQL 1.0 und 1.1 sowie die Möglichkeit Anfragen gegen gleichzeitig mehrere statische sowie dynamische Datenquellen zu verarbeiten, in Betracht gezogen. Die Fähigkeit Rückschlüsse mittels logischer Inferenz aus den Daten abzuleiten war ein weiteres Kriterium.
- **Leistungsfähigkeit und Skalierbarkeit** Das Verhalten des Systems unter hohen Datenankunftsraten, Ausführung der Anfragen gegen eine hohe Anzahl von Datenquellen und gleichzeitige Ausführung einer hohen Anzahl der Anfragen, da diese den reellen Einsatzszenarien entsprechen, wurden als maßgebliches Kriterium für die Auswertung der untersuchten Systeme festgelegt.

Über die existierenden Benchmarks zur Auswertung solcher Systeme wurde recherchiert und ihre Eignung bezüglich der definierten Untersuchungskriterien geprüft. Es wurden folgende Benchmarks untersucht: NEXMark, LUBM, Linear Road, SRBench und LSBench. Keines der untersuchten Benchmarks verfügte über alle benötigten Bausteine, die das Testen der Systeme gegen alle relevanten Kriterien ermöglichen. Aus diesem Grund wurden im Rahmen dieser Arbeit die benötigten Bausteine für solch ein Benchmark entworfen und implementiert. Der Anwendungsfall des Benchmarks wurde auf den im Rahmen des Optique-Projekts von Siemens AG bereitgestellten Daten der Gasturbinen aufgebaut. Das Benchmark umfasst Generatoren für RDF-Daten, die sowohl dem Anwendungsfall entsprechende als auch generische Daten produzieren. Des Weiteren wurde ein fester Satz von Testanfragen, ein Generator für Leistungsanfragen sowie eine Flex/BlazeDS-Webanwendung zur Durchführung von Tests implementiert.
Basierend auf dem entwickelten Benchmark wurde das C-SPARQL Engine getestet. Aufgrund von Bugs, besonders beim Einsatz mehrerer Datenquellen, konnte das Engine nur eingeschränkt ausgewertet werden. Das C-SPARQL Engine zeigte eine Unterstützung genereller Anfragemuster, zeichnete sich aber bereits für simple Anfragen durch hohe Ausführungszeiten aus. Andere Engines konnten wegen mangelnder Dokumentation und teilweise fehlender öffentlicher Implementierung nicht getestet werden.

Inhaltsverzeichnis

Eidesstattliche Erklärung **3**

Abbildungsverzeichnis **II**

Tabellenverzeichnis **III**

Verzeichnis der Listings **IV**

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 strRS Engines . . . 2
- 1.3 Ziel dieser Arbeit . . . 3
- 1.4 Untersuchungskriterien . . . 3
- 1.5 Aufbau dieser Arbeit . . . 3

2 Basiskonzepte und Techniken des Semantischen Web **5**
- 2.1 RDF . . . 6
  - 2.1.1 Ressourcenidentifikation . . . 7
    - 2.1.1.1 Literale und Datentypen . . . 7
    - 2.1.1.2 Blank Nodes . . . 7
  - 2.1.2 Syntax der RDF-Dokumente . . . 8
    - 2.1.2.1 Tripel-basierte Repräsentation . . . 8
    - 2.1.2.2 RDF/XML-Repräsentation . . . 9
- 2.2 Vokabulare und einfache Ontologien mit RDFS . . . 9
- 2.3 Ontologien mit OWL . . . 10
  - 2.3.1 OWL Syntax . . . 11
    - 2.3.1.1 Eigenschaften . . . 11
    - 2.3.1.2 Klassen und Klassenbeziehungen . . . 11
- 2.4 Anfragesprache SPARQL . . . 13
  - 2.4.1 Syntax und Semantik der SPARQL . . . 13
    - 2.4.1.1 Syntax der SPARQL-Graph-Muster . . . 14
    - 2.4.1.2 Semantik der SPARQL-Graph-Muster . . . 14
    - 2.4.1.3 SPARQL 1.1 . . . 15
    - 2.4.1.4 SPARQL Folgebeziehungsregimes . . . 17

3 Semantische Datenströme: Basiskonzepte, Sprachen und Anfragenengines 18
3.1 Grundlagen der Datenstromverarbeitung . . . 18
3.1.1 Kontinuierliche Anfragen . . . 18
3.1.2 Ausführungsstrategien . . . 20
3.2 Verarbeitung semantischer Datenströme . . . 21
3.2.1 Linked Stream Data . . . 21
3.2.2 Datenmodell . . . 21
3.2.3 Operatoren . . . 22
3.2.4 Anfragesprachen . . . 23
3.2.5 Architekturen der Anfrageengines . . . 24

4 Verwandte Arbeiten 28
4.1 Linear Road Benchmark . . . 28
4.1.1 Usecase . . . 29
4.1.2 Kriterien und Auswertung . . . 29
4.2 LUBM . . . 29
4.2.1 Kriterien zur Definition von Testanfragen . . . 30
4.2.2 Leistungsmetriken . . . 30
4.3 NEXMark . . . 31
4.3.1 Anwendungsfall . . . 31
4.3.2 Leistungsmetriken . . . 32
4.4 SRBench . . . 32
4.4.1 Daten . . . 33
4.4.2 Anfragen . . . 34
4.4.3 Auswertung . . . 34
4.5 LSBench . . . 35
4.5.1 Anwendungsfall . . . 36
4.5.2 Anfragen . . . 37
4.5.3 Auswertung . . . 37
4.5.3.1 Korrektheit . . . 37
4.5.3.2 Leistung . . . 38

5 Benchmark: Entwurf und Implementierung 40
5.1 Anwendungsfall: Siemens Gasturbinen . . . 40
5.1.1 Anpassung der bereitgestellten Daten . . . 40
5.1.1.1 Erweiterung der Ontologie . . . 41
5.1.1.2 Vorbereitung der Messdaten . . . 41
5.1.2 Statische Daten . . . 42
5.1.3 Stromdaten . . . 43

5.2 Anfragen 44
5.2.1 Anfragen für funktionale Tests 45
5.2.2 Anfragen für Skalierbarkeits- und Leistungstests 49
5.3 Implementierung 51
5.3.1 Tripelerzeugung 51
5.3.1.1 Zufallsdaten 51
5.3.1.2 Gasturbinendaten 52
5.3.2 Anfragengenerator 52
5.3.3 Webanwendung 53
5.3.3.1 BlazeDS-Backend 54
5.3.3.2 C-SPARQL 54
5.3.3.3 CQELS 55
5.3.3.4 Flex-Frontend 56

6 Ergebnisse und Auswertung **57**
6.1 Funktionalität 57
6.2 Leistungsfähigkeit und Skalierbarkeit 58

7 Fazit und Ausblick **61**
7.1 Ausblick 62

8 Appendix **63**
8.1 Anfragenformulierungen in CSPARQL 63
8.2 Angepasste Ontologien 69

Literaturverzeichnis **80**

Abbildungsverzeichnis

2.1 Semantic Web Stack des W3C 6

3.1 Whitebox-Architektur 25
3.2 Blackbox-Architektur 26

4.1 Nichtübereinstimmung des Resultats für eine Beispielanfrage 32
4.2 SRBench-Datensätze und deren Beziehungen 34
4.3 LSBench Datensätze und deren Beziehungen 36
4.4 Vergleichbarer maximaler Datendurchsatz abhängig von der Größe des statischen Datensatzes 38
4.5 Vergleichbarer maximaler Datendurchsatz bei der Ausführung mehrerer Anfragen 39

6.1 Mittlere Ausführungszeit bei unterschiedlicher Datenankunftsrate . . 59
6.2 Mittlere Ausführungszeit bei gleichzeitiger Ausführung mehrerer Anfragen 60

Tabellenverzeichnis

5.1 In funktionalen Anfragen eingesetzte Merkmale. 49

6.1 Ausführungsumgebung für das Benchmark 57

Verzeichnis der Listings

2.1 Beispiel für Turtle-Syntax 8
2.2 Beispiel für RDF/XML-Syntax 9
2.3 Eigenschaften in einer OWL-Ontologie 11
2.4 Klassendefinition in einer OWL-Ontologie 12
2.5 Beispiel einer SELECT-Anfrage 16
2.6 Datengraph für Folgebeziehungsregime-Beispiel 17
2.7 SPARQL-Anfrage für Folgebeziehungsregime-Beispiel 17

5.1 Erweiterung der Optique-Ontologie 41
5.2 Beispiel erzeugter statischer RDF-Daten 42
5.3 Beispiel erzeugter Stromdaten 43
5.4 Beispiel einer generierten Anfrage für C-SPARQL 50

8.1 Anfrage 1 63
8.2 Anfrage 2 63
8.3 Anfrage 3 63
8.4 Anfrage 4 64
8.5 Anfrage 5 64
8.6 Anfrage 6 64
8.7 Anfrage 7 65
8.8 Anfrage 8 66
8.9 Anfrage 9 66
8.10 Anfrage 10 66
8.11 Anfrage 11 67
8.12 Anfrage 12 67
8.13 Anfrage 13 68
8.14 Anfrage 14 68
8.15 Anfrage 15 68
8.16 Anfrage 16 69
8.17 Ontologie der Gasturbinen-Wissensdomäne 69
8.18 Basisontologie 74

1 Einleitung

1.1 Motivation

Als Folge des derzeitigen technologischen Progresses in allen Bereichen unseres Lebens ist der Umfang daran teilnehmender oder diesen unterstützender Prozesse enorm gewachsen. Ein Beispiel für solch einen Prozess ist das Ablesen von unterschiedlichen Umgebungscharakteristiken eines Systems mithilfe von Sensoren. Solche Prozesse generieren große Mengen von Echtzeitdaten (auch **Big Data**[1] genannt). Die Echtzeitdaten zusammengefasst in Datenströme beschreiben den momentanen Zustand des Teil- oder Gesamtsystems und somit auch seine mögliche Veränderung, die eine sofortige Reaktion benötigt. Die Verarbeitung dieser Daten anhand schon existierender Systeme, wie Datenstrommanagementsysteme (**DSMS**)[2], ist wegen ihrer Menge und Komplexität sehr schwierig bis unmöglich geworden. Das neu angelaufene EU-Projekt namens **Optique**[3] hat als Ziel die Entwicklung einer erweiterbaren Plattform. Es soll eine vollständige und generische Lösung, die den Anforderungen bei der Verarbeitung und dem Zugriff auf Big Data entspricht, darstellen.
Die Entwicklung des Semantischen Web und der damit zusammenhängenden Technologien hat Standards zur Darstellung und Verarbeitung von Daten in einer Form geschaffen, das einen logischen Kontext für diese mitträgt. Diese erlaubt neue Formen der Datenanalyse und somit daraus folgender logischer Rückschlüsse und Entscheidungen. Die wichtigsten davon sind die formale Sprache **RDF** und die **SPARQL**-Anfragesprache für RDF-Daten. Diese Technologien sind allerdings auf statische Daten gerichtet. Um von den Einsatz dieser Technologien in Echtzeitumgebungen zu profitieren, hat eine Weiterentwicklung in den letzten Jahren angefangen, die eine Verarbeitung von gleichzeitig statischen und dynamischen Datenquellen erlaubt. Somit wird ein Übergang vom Reasoning auf der Basis von statischen Daten zum Echtzeit-Reasoning (auch **Stream Reasoning**[4] genannt) ermöglicht.
Um Stream Reasoning möglich zu machen, wird die schon existierende SPARQL-Semantik ähnlich dem, wie **CQL**[5] SQL erweitert, um Datenströme erweitert. Da

[1] `http://en.wikipedia.org/wiki/Big_data`
[2] `http://de.wikipedia.org/wiki/Data_Stream_Management_System`
[3] optique
[4] Valle u. a. [2009]
[5] Arasu u. a. [2006]

Datenströme ständig neue Daten liefern, werden zusätzlich zu der Semantikerweiterung RDF/SPARQL Engines zur Verarbeitung von kontinuierlichen Anfragen gegen Datenströme (weiter **strRS Engines**[6]) entwickelt. Eine derartige Technologie ist ein wichtiger Baustein der im Rahmen des Optique-Projekts zu entwickelnden Plattform.

1.2 strRS Engines

In den letzten Jahren wurden mehrere strRS Engines vorgestellt und an verschiedenen Einrichtungen entwickelt. Da derzeit keine Standardanfragesprache für RDF-Datenströme existiert, wurde von jedem Team auch eine eigene Erweiterung der SPARQL-Anfragesprache entwickelt. Dementsprechend kann jedes Engine kontinuierliche Anfragen, die in entsprechender SPARQL-Erweiterung formuliert sind, verarbeiten. Die Architekturen der strRS Engines können in zwei folgende Kategorien unterteilt werden:

- **Whitebox** In einer Whitebox-Architektur muss das Engine alle Bausteine eines DSMS, wie Executer und Optimizer, sowie Operatoren, z. B. gleitendes Zeitfenster, implementieren[7]. **CQELS**-Engine[8] baut auf dieser Architektur auf.
- **Blackbox** Die Implementierung nach der Blackbox-Architektur empfiehlt den Einsatz von anderen Systemen als Teilkomponenten[9]. Da die Teilkomponente normalerweise unterschiedliche Datenformate und Anfragensprachen einsetzten, muss das Gesamtsystem weitere Komponenten zur Datentransformation, Anfragenumformulierung und Verwaltung der Zusammenarbeit der Teilsysteme implementieren. Zu den Engines, die der Blackbox-Architektur folgen, gehören **SPARQL**$_{stream}$[10] und **CSPARQL**[11].

Zusätzlich zum Einsatz verschiedener Ansätze bei der Implementierung weisen die Engines auch mehrere weitere Unterschiede, z. B. Ausführungsmechanismus, auf. Der Umfang der unterstützten Funktionalität variiert auch zwischen den aktuellen Engines, da all diese sich noch in der Entwicklung befinden. Aus diesem Grund müssen die verschiedenen Engines auf einer gemeinsamen Basis untersucht und ausgewertet werden.

[6] [Zhang u. a. 2012, S. 1]
[7] [Le-Phuoc u. a. 2012b, S. 268]
[8] cqels
[9] [Le-Phuoc u. a. 2012b, S. 269]
[10] sparqlstr
[11] csparql

1.3 Ziel dieser Arbeit

Ziel dieser Arbeit ist eine Entwicklung der Bausteine eines Benchmarks für strRS Engines. Der Anwendungsfall des Benchmarks basiert auf den Gasturbinen-Daten, die von Siemens AG im Rahmen des Optique-Projektes zu Verfügung gestellt wurden. Das Benchmark soll eine Basis für den Vergleich und Auswertung der aktuellen und noch zu entwickelnden strRS Engines bieten. Des Weiteren werden die Engines, die eine öffentliche Implementierung anbieten, basierend auf den entwickelten Bausteinen und relevanten Kriterien getestet.

1.4 Untersuchungskriterien

Bei der Untersuchung der Anforderungen an strRS Engines im Rahmen dieser Arbeit haben sich folgende Kriterien als relevant herausgestellt:

- Umfang der unterstützten SPARQL-Funktionalität
- Gleichzeitiger Einsatz statischer und dynamischer Datenquellen
- Unterstützung des Reasoning
- Skalierbarkeit bei Einsatz mehrerer Datenquellen
- Leistungsfähigkeit abhängig von Eingabegröße und Datenankunftsrate
- Skalierbarkeit bei gleichzeitiger Ausführung mehrerer Anfragen
- Korrektheit der Resultate

Im Rahmen der Arbeit hat sich herausgestellt, dass nur CSPARQL eine Implementierung und entsprechende Dokumentation zu Verfügung stellt, die nach der Kommunikation mit den Entwicklern den Einsatz ermöglicht. Beide anderen Engines konnten wegen mangelnder Dokumentation und Kommunikation mit den Entwicklern nicht getestet werden. Somit fand nur die Auswertung von CSPARQL, wo es die Implementierung ermöglichte, nach oben genannten Kriterien statt.

1.5 Aufbau dieser Arbeit

In Kapitel 2 werden verwandte Arbeiten im Bereich Benchmarking vorgestellt. Es wurden Benchmarks gegen einfache und RDF-Datenströme und auch statische RDF-Daten vorgestellt.
Kapitel 3 widmet sich der Einführung in Technologien des Semantischen Webs. Dabei werden vor allem die existierenden Standardsprachen zur Darstellung von Daten,

Ontologien und Formulierung von Anfragen vorgestellt.
Kapitel 4 führt in die Verarbeitung von RDF-Datenströmen ein und stellt verschieden strRS Engines sowie entsprechende Erweiterungen der SPARQL-Anfragensprachen vor.
In Kapitel 5 werden der Entwurf und die Details der Implementierung der Benchmarkbausteine beschrieben. Diese umfassten alle Schritte von der Vorbereitung der Daten für den Anwendungsfall bis zur Erzeugung von RDF-Datenströmen basierend auf diesen Daten und Definition von Testanfragen.
Im Kapitel 6 wird eine Auswertung des CSPARQL-Engines durchgeführt. Diese umfasst funktionale und Leistungstests basierend auf dem Satz von Testanfragen.
Kapitel 7 fasst die Resultate dieser Arbeit zusammen.

2 Basiskonzepte und Techniken des Semantischen Web

Das World Wide Web ist ein von Menschen kreiertes Werkzeug, das nicht nur zur Verwaltung unserer verschiedensten Aktivitäten, sondern auch zur Weiterentwicklung und Gestaltung neuer beiträgt. Das Web selbst hat die Umstände unserer Realität so verändert, dass es selbst weiterentwickelt werden muss, um diesen gewachsen zu sein. Eine der wichtigsten Veränderungen letztes Jahrzehntes ist die Menge der produzierten Daten. Diese werden zwar verwaltet, analysiert und verwendet, aber die aus der Menge und Heterogenität resultierende Komplexität erfordert neue Methoden und Werkzeuge.
Semantisches Web ist eine Initiative des World Wide Web Consortium(W3C)[12] zur Weiterentwicklung des Web. Dieser Begriff wurde von Timothy John Berners-Lee[13] geprägt und umfasst Standards und Technologien, die eine Grundlage sind für Repräsentation von Daten in einer, nicht nur für Menschen, sondern auch für Maschinen, relevanten und interpretierbaren Form. Das Hinzufügen eines Kontextes und semantischer Bedeutung zu Informationen erleichtert eine automatisierte Verarbeitung, Analyse und Interoperabilität dieser oder bietet dafür sogar neue, vorher nicht erreichbare, Möglichkeiten. Die wichtigsten, aber nicht die einzigen Standards, die das Semantische Web bilden, sind:

- **Resource Description Framework** (RDF) - Framework zu Beschreibung und Modellierung von Informationen
- **Resource Description Framework Schema** (RDFS) - Eine Menge von Klassen zur Beschreibung genereller Struktur der RDF-Ressourcen in Form von Vokabularen und einfachen Ontologien
- **Web Ontology Language** (OWL) - Beschreibungssprache zur Modellierung von Ontologien der Anwendungsdomänen
- **SPARQL Protocol And RDF Query Language** (SPARQL) - Anfragesprache für RDF-Ressourcen

[12] `http://www.w3.org`
[13] `http://en.wikipedia.org/wiki/Tim_Berners-Lee`

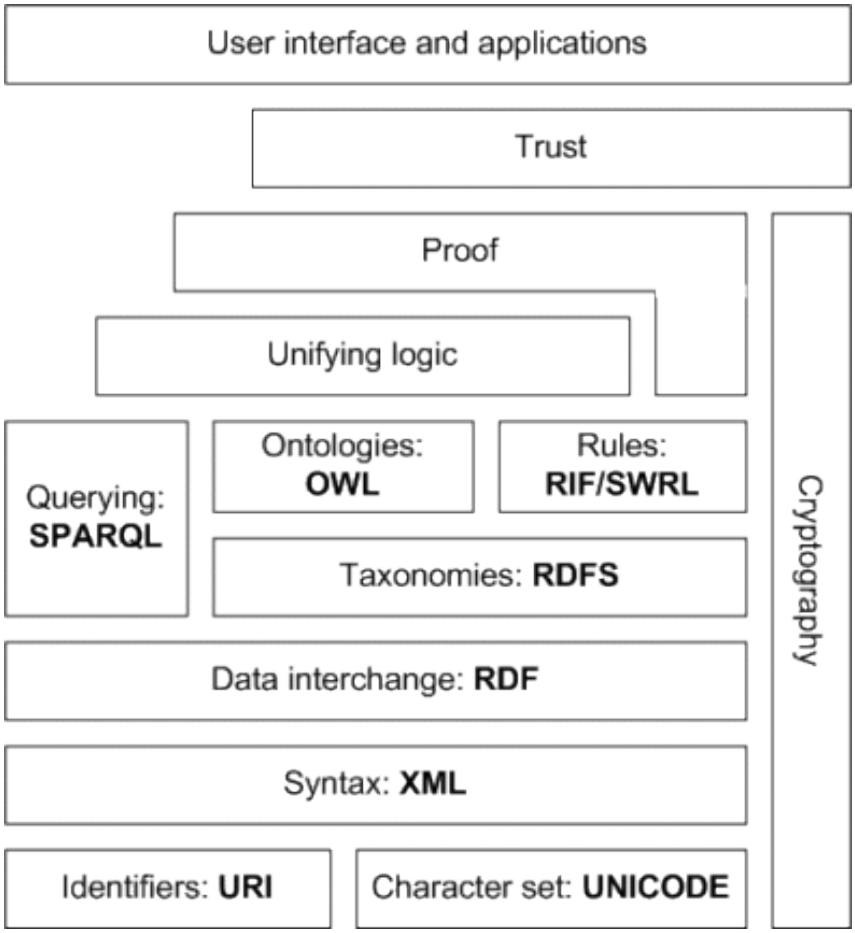

Abbildung 2.1: Semantic Web Stack des W3C
swwiki

2.1 RDF

Das Semantische Web wird auch "Web of Data"(engl. Netz von Daten) genannt, somit spielt die Darstellung von Informationen eine grundlegende Rolle. Informationen werden mithilfe der formalen Sprache RDF beschrieben in einer Form, die nicht nur diese selbst, sondern auch deren Bedeutung und Beziehungen zu anderen beinhaltet. Diese Herangehensweise ermöglicht eine sinnvolle Kombination der Daten aus verschiedenen Quellen und Verarbeitung.
Ein RDF-Dokument stellt einen gerichteten bezeichneten Graphen dar. Die Knoten spielen die Rolle der Subjekte und Objekte und die gerichteten Kanten die Rolle der Prädikate, beschreiben also die Beziehung zwischen diesen Knoten. Somit besteht ein RDF-Graph, auch als RDF-Datensatz bezeichnet, aus einer Menge der RDF-Tripel.

2.1.1 Ressourcenidentifikation

Eine Kante mit den damit verbundenen Knoten stellt eine Aussage dar. Diese wird über Ressourcen getroffen. Eine Ressource kann für informative (z. B. Webseite oder Bild), nicht-informative (z. B. Stadt) oder auch abstrakte Objekte stehen und wird mit einem sogenannten **IRI** (Internationalized Resource Indetifier)[14] identifiziert oder im Falle einer anonymen Ressource mit einem **Blank Node** (engl. leerer Knoten) [15]. Ein Subjekt kann somit mit einem IRI oder einen Blank Node angegeben werden. Das Prädikat ist immer über ein IRI mit einer konkreten Ressource verbunden, die die Beziehung beschreibt. Das Objekt kann dagegen eine Ressource (IRI/Blank Node) oder ein Literal (eine Zeichenfolge zur Darstellung der Werte von Basistypen)[16] sein.
Ein RDF-Tripel stellt somit eine Aussage dar und ist formalisiert als $(s, p, o) \in IB \times I \times IBL$. Die Bezeichnung I,B und L stehen hier und allgemein im Rahmen dieser Arbeit für IRI, Blank Node und Literal, die zusammengesetzten Buchstaben für Vereinigungen entsprechender Mengen, s für Subjekt, p für Prädikat und o für Objekt.

2.1.1.1 Literale und Datentypen

Literale sind Zeichenketten, die einen bestimmten Wert darstellen, und als Objekt in einer Beziehung benutzt werden können. Ohne weitere Angabe ist ein Literal ungetypt und stell somit in manchen Fällen, z. B. Integer-Werte, keine nützlichen Verarbeitungsmöglichkeiten zur Verfügung. Der Typ kann allerdings explizit angegeben werden. Dafür wird der Wert des Literals in Anführungszeichen umschlossen, gefolgt von ^^-Zeichen und einem Datentyp in Form von IRI (z. B. `"51.03"^^http://www.w3.org/2001/XMLSchema#float`). Für allgemein benutzte Datentypen wird die Benutzung von Typen aus dem XML-Schema empfohlen, damit diese einheitlich interpretiert werden können[17].

2.1.1.2 Blank Nodes

In manchen Fällen werden in Beziehungen Hilfsressourcen benutzt, die nur der Strukturierung dienen und nicht global referenziert werden[18]. Diesen Ressourcen wird kein URI zugewiesen. Sie werden als Blank Nodes bezeichnet. Trotzdem erhalten sie in verschiedenen Serialisierungsformaten 2.1.2 einen lokalen Identifikator mit

[14] `http://en.wikipedia.org/wiki/Internationalized_resource_identifier`
[15] 2.1.1.2
[16] `http://de.wikipedia.org/wiki/Literal`
[17] [Hitzler u. a. 2008, S. 56]
[18] [Hitzler u. a. 2008, S. 57]

dem Zweck, den gleichen Knoten in mehreren Beziehungen lokal einzusetzen. Mögliche Einsatzszenarien (z. B. mehrwertige Beziehungen) werden im nächsten Abschnitt vorgestellt.

2.1.2 Syntax der RDF-Dokumente

Da RDF nur die Art und Weise zur Beschreibung von Daten definiert, können diese in verschiedenen Formaten dargestellt werden. Dieser Abschnitt stellt die gängigsten Syntaxen für RDF vor.

2.1.2.1 Tripel-basierte Repräsentation

Bei dieser Herangehensweise wird jede Kante des Graphen, also eine Elementaraussage, in Form eines aus Subjekt, Prädikat und Objekt bestehenden Tripels gespeichert. 1998 hat Tim Berners-Lee die Tripel-basierte Umsetzung namens **Notation 3** (N3) vorgeschlagen, die 2004 sich in eine weniger komplexe Syntax **N-Triples** entwickelt hat[19]. Diese unterstützt allerdings einige nützliche Darstellungsmöglichkeiten (z. B. kompakte URIs) nicht[20]. Die Alternative dazu bietet **Turtle** (Terse RDF Triple Language), das eine Teilmenge des Notation 3 ist[21]. Diese unterstützt eine kompakte und menschenfreundliche Darstellung.

Listing 2.1: Beispiel für Turtle-Syntax

```
1 @prefix ex: <http://www.sts.tu-harburg.de/optique/> .
2 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
3 <http://www.sts.tu-harburg.de/optique/TR/tr1>  ex:engine "HSR-201" ;
4                     ex:contains  [
5     ex:part  ex:tempSensor;
6     ex:numberOf "53"^^xsd:integer
7     ]  .
```

Im Codeabschnitt 2.1 ist ein aus vier Aussagen bestehender RDF-Graph dargestellt. Die ersten beiden Zeilen definieren die Namensräume, die eine kompakte Schreibweise für URIs erlauben (z. B. *ex:engine*). Das Semikolon in Zeile 3 bedeutet, dass diese und folgende Zeile das gleiche Subjekt teilen und dieses nicht mehrmals angegeben werden muss. Bei gleichzeitig gleichem Subjekt und Prädikat kann diese Schreibweise ebenso benutzt werden.
In den Zeilen 4-7 sieht man, dass verschieden kompakte Schreibweisen kombiniert werden können. Das Subject <http://www.test.org/TR/tr1> steht in einer Beziehung ex:contains zum Blank Node. Die in []-Klammern stehenden Zeilen 5-6 stellen

[19] [Hitzler u. a. 2008, S. 40]
[20] http://en.wikipedia.org/wiki/N-Triples
[21] http://de.wikipedia.org/wiki/Turtle_(Syntax)

zwei Aussagen mit dem gleichen Blank Node ohne ID als Subject dar. Die Semikolon-Schreibweise wird damit wiederum innerhalb des Blank Node eingesetzt.

2.1.2.2 RDF/XML-Repräsentation

Wegen der geringen Anzahl von Bibliotheken für Turtle/N3 und einer hohen für XML in verschiedensten Programmiersprachen bleibt die XML-Serialisierung die am häufigsten benutzte Syntax für RDF[22]. Der folgende Code zeigt den RDF-Graphen aus dem vorherigen Abschnitt in RDF/XML-Format.

Listing 2.2: Beispiel für RDF/XML-Syntax

```
1  <rdf:RDF
2      xmlns:ex="http://www.sts.tu-harburg.de/optique/"
3      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4      xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
5      <rdf:Description rdf:about="http://tuhh.de/optique/TR/tr1">
6          <ex:contains rdf:parseType="Resource">
7              <ex:numberOf rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">53</
                   ex:numberOf>
8              <ex:part rdf:resource="http://www.sts.tu-harburg.de/optique/tempSensor"/>
9          </ex:contains>
10         <ex:engine>HSR-201</ex:engine>
11     </rdf:Description>
12 </rdf:RDF>
```

Wurzelelement des RDF/XML-Dokumentes ist rdf:RDF[23]. Der ganze RDF-Graph wird beim Einsatz von XML-Schreibweise nach Subjekten gruppiert. Die Knoten des Graphen (Subjekte und Objekte) sind mit Description-Elementen dargestellt. Die Bezeichnung wird diesen über rdf:about-Eigenschaft zugewiesen. Die Prädikate werden als Kind-Elemente der Subjekte repräsentiert. Das Blank Node wurde über die rdf:parseType="Resource" in Zeile 6 als solches definiert. Die vorgestellte Schreibweise ist nur eine der möglichen. Die ganze Syntaxspezifikation kann in rdfxml gefunden werden.

2.2 Vokabulare und einfache Ontologien mit RDFS

Die Semantik der Ressourcen in RDF-Dokumenten ist nicht ohne weiteren Kontext von verschiedenen Parteien interpretierbar. Die Aussagen beschreiben zwar Beziehungen zwischen konkreten Individuen, beinhalten aber selbst keine Informatio-

[22] [Hitzler u. a. 2008, S. 40]

[23] rdf-Präfix steht für http://www.w3.org/1999/02/22-rdf-syntax-ns# und kann zwar frei gewählt werden. Diese Abkürzung ist aber empfohlen und von meisten benutzt.

nen bezüglich der Struktur oder Bedeutung dieser Beziehungen oder Hierarchie der Individuen-Klassen innerhalb der entsprechenden Wissensdomäne. Das Wissen über die Begriffe und ihre Beziehungen kann in Form von Vokabularen oder Ontologien dargestellt werden.
RDFS ist eine Empfehlung von W3C zur Erstellung von Vokabularen. RDFS ist eine auf RDF basierende Sprache zur Wissensrepräsentation. RDFS stellt selbst ein Vokabular[24] zur Verfügung, das die universellen Bausteine zur Modellierung einer Wissensdomäne beinhaltet.
Die Definition der Vokabulare mit RDFS basiert auf folgenden Konstrukten:

- Klassen, die in einem rdfs:Class-Element spezifiziert werden
- Eigenschaften (als Prädikate benutzbare Ressourcen), die in einem rdf:Property-Element spezifiziert werden

Weitere Details zu RDFS können in rdfs gefunden werden.

2.3 Ontologien mit OWL

Die Aussagekraft von RDFS ist offensichtlich beschränkt. Für die Erstellung von komplexen Ontologien wurde 2004 von W3C die Web Ontology Language (OWL) entwickelt. OWL existiert in drei Varianten **OWL Full**, **OWL DL**[25] und **OWL Lite**. Die OWL Full ist die ausdrucksstärkste Variante. Ihre Komplexität zieht die Unentscheidbarkeit der damit erstellten Ontologien mit[26]. Aus diesem Grund sind die DL- und Lite-Varianten für die Anwendungsentwicklung relevant, wobei die letzte für einfache Ontologien gedacht ist. Alle Varianten der OWL-Sprache schließen RDFS mit unterschiedlichen Einschränkungen ein. Die genauen Unterschiede der Varianten sind im Kapitel 5.2 von Hitzler u. a. [2008] zusammengefasst. **OWL2** ist die Weiterentwicklung von OWL. Die Unterschiede der Versionen sind in dieser Arbeit nicht dargestellt. Da OWL2 rückwärts kompatibel zur OWL ist, sind alle hier vorgestellten Beispiele gültig bezüglich OWL und OWL2.
Es existieren genauso wie für RDF mehrere Syntaxen für OWL, die RDF/XML-Variante gilt aber als die meist benutzte und normative Syntax. Da diese auf XML aufbaut ist eine unterschiedliche und dabei äquivalente Darstellung gleicher Information möglich. Beispiele in diesem Abschnitt werden in der kompakten Schreibweise dieser Syntax dargestellt.

[24] `http://en.wikipedia.org/wiki/Resource_Description_Framework#RDF_vocabulary`
[25] Dl steht für Beschreibungslogik(engl. *description logic*)
[26] [Hitzler u. a. 2008, S. 127]

2.3.1 OWL Syntax

Die Grundbausteine der OWL-Ontologien sind Klassen, Eigenschaften (alternativ Rollen genannt) und Instanzen der Klassen, auch Individuen genannt[27]. Die Konstrukte zur Erstellung einer Ontologie basierend auf diesen werden in diesem Abschnitt vorgestellt. Die komplette Referenz der OWL-Sprache ist unter owl verfügbar.

2.3.1.1 Eigenschaften

OWL im Unterschied zu RDFS unterscheidet zwischen zwei Arten von Eigenschaften: die owl:ObjectProperty und owl:DatatypeProperty die Beziehungen zwischen Individuen oder zwischen Individuen und Datenwerten beschreiben. OWL besitzt im Vergleich zu RDFS einen großen Satz von Attributen und Konstrukten, die eine präzise Definition des Verhaltens und der Beziehungen zu anderen Eigenschaften erlauben. Im folgenden Codeabschnitt sind drei Eigenschaften definiert. Die Angabe von Namensräumen und schon in 2.2 vorgestellten Attributen wurden ausgelassen.

Listing 2.3: Eigenschaften in einer OWL-Ontologie

```
1 <owl:ObjectProperty rdf:ID="hasPart"/>
2 <owl:ObjectProperty rdf:about="isPartOf">
3     <owl:inverseOf rdf:resource="hasPart"/>
4     <rdf:type rdf:resource="&owl;TransitiveProperty"/>
5 </owl:ObjectProperty>
6 <owl:ObjectProperty rdf:ID="isDirectPartOf">
7     <rdfs:subPropertyOf rdf:resource="isPartOf"/>
8     <rdf:type rdf:resource="&owl;FunctionalProperty"/>
9 </owl:ObjectProperty>
```

Eine weitere Eingrenzung der Beziehungstypen in diesem Beispiel wird mithilfe von &owl;TransitiveProperty und &owl;FunctionalProperty angegeben. Die owl:inverseOf deklariert eine Inversion-Beziehung zwischen zwei Eigenschaften. Die Angabe weiterer lokaler Einschränkungen der Eigenschaften zur Definition von Klassen wird in 2.4 vorgestellt.

2.3.1.2 Klassen und Klassenbeziehungen

Da OWL RDFS mit einschließt, sind die schon in 2.2 erwähnten Möglichkeiten zur Definition von Klassen anwendbar. Die Ausdrucksstärke von OWL wird beim Einsatz komplexerer Methoden zu Beschreibung einer Klasse zur Geltung gebracht. OWL

[27] [Hitzler u. a. 2008, S. 129]

stellt einen Satz von logischen Konstruktoren, die auf Klassen anwendbar sind. Eine Klasse kann als Produkt der logischen Operationen und Beziehungen auf/zu anderen Klassen beschrieben werden. Eine weitere Methode ist, eine Klasse "*als die Menge aller Objekte, für die eine bestimmte Rolle immer einen Wert aus einer vorgegebenen Klasse annimmt* "[28], zu definieren.

Listing 2.4: Klassendefinition in einer OWL-Ontologie

```
1  <owl:Class rdf:about="Feature">
2      <owl:disjointWith rdf:resource="SystemElement"/>
3      <rdfs:subClassOf>
4        <owl:Restriction>
5          <owl:onProperty rdf:resource="isFeatureFor"/>
6          <owl:someValuesFrom rdf:resource="Symptom"/>
7        </owl:Restriction>
8      </rdfs:subClassOf>
9      <owl:equivalentClass>
10       <owl:Class>
11         <owl:unionOf rdf:parseType="Collection">
12           <owl:Class rdf:about="BooleanFeature"/>
13           <owl:Class rdf:about="NumericalFeature"/>
14           <owl:Class rdf:about="StringFeature"/>
15         </owl:unionOf>
16       </owl:Class>
17     </owl:equivalentClass>
18  </owl:Class>
```

In 2.4 ist eine Klassendefinition dargestellt, die alle vorher angesprochenen Konstrukte kombiniert. Die Klasse Feature ist disjunkt zur Klasse SystemElement und äquivalent zu einer Vereinigung drei anderer Klassen deklariert. Das owl:Restriction-Element in Zeile 4 definiert eine Menge von Objekten über eine Eigenschaftseinschränkung. Zu dieser steht wiederum die definierende Klasse in einer Unterklassenbeziehung.
Die vorgestellten Eigenschaften der OWL-Sprache machen Inferenz möglich, z. B. aus der Klassendefinition kann festgestellt werden, dass StringFeature kein SystemElement ist. Ein weiterer logischer Schluss wäre, dass ein Symptom zu einer Instanz einer der drei Klassen in der Vereinigung als Objekt in einer hasFeature-Beziehung[29] stehen kann.

[28] [Hitzler u. a. 2008, S. 139]
[29] Es wird angenommen, dass hasFeature eine inverse Eigenschaft zur isFeatureFor darstellt.

2.4 Anfragesprache SPARQL

Ein wichtiger Bestandteil jedes Systems zur Verwaltung und zum Austausch von Daten ist eine Möglichkeit Anfragen zu formulieren um sinnvolle Informationen daraus abzuleiten. SPARQL ist ein Protokoll und eine Anfragesprache für RDF-Daten.

2.4.1 Syntax und Semantik der SPARQL

SPARQL ist eine auf dem Graph-Abgleich (engl. *graph matching*) basierende Anfragesprache. Die SPARQL-Anfragen bestehen aus dem Kopf und dem Körper der Anfrage. Der komplexere Teil ist dabei der Körper. Dies besteht aus einem Graph-Muster. Das Muster wird mittels Disjunktion, Konjunktion der RDF-Tripel, die auch Variablen enthalten können, oder optionalen Untergraphen und Bedingungen definiert. Der Kopf enthält Ausdrücke, die die Darstellung des Resultats der Anfrage definieren. Bei der Ausführung der Anfrage gegen einen RDF-Datensatz wird das Graph-Muster aus dem Körper der Anfrage mit dem Datensatz abgeglichen. Die daraus resultierenden Bindungen der Variablen werden entsprechend der Definition im Kopf der Anfrage mithilfe relationaler Operatoren, z. B. Projektion, dargestellt und zurückgegeben. Die Form des Resultats definiert der Typ der Anfrage. SPARQL stellt folgende Typen zu Verfügung:

- SELECT stellt das Resultat in Form einer Tabelle dar. Diese enthält alle ermittelten Bindungen der Variablen des Graph-Musters
- ASK liefert ja/nein bei Existenz bzw. Nichtexistenz dem Graph-Muster entsprechender Variablenbindungen.
- CONSTRUCT liefert einen neuen RDF-Datensatz, der aus den Werten in den ermittelten Variablenbindungen zusammengestellt wird.
- Mithilfe der DESCRIBE-Anfrage kann die Menge der zu beschreibenden Ressourcen ausgewählt werden. Welche weiteren Merkmale der Ressourcen übertragen werden, bleibt dann der konkreten Implementation überlassen[30].

Modifikation des Resultats Die Darstellung des Resultats der Anfrage kann zusätzlich mithilfe von Modifikatoren angepasst werden. Diese werden zwar teilweise nach dem Körper platziert, gehören dennoch logisch zum Kopf der Anfrage. Dazu gehören Operatoren, die das Sortieren, die Eingrenzung der Anzahl und Gruppierung

[30] [Hitzler u. a. 2008, S. 224]

der resultierenden Variablenbindungen erlauben Dazu gehören vor allem folgende Operatoren: ORDER BY, LIMIT und SORD BY.

2.4.1.1 Syntax der SPARQL-Graph-Muster

Die Syntax zur Definition des Graph-Musters umfasst vier Operatoren: OPTIONAL, weiter bei der Formalisierung als OPT bezeichnet, UNION, FILTER und (.), das einem AND-Operator entspricht. Für diese sind in sparql implizite Rangfolgen und Assoziativitätsregeln definiert. Die Muster können mittels geschweifter Klammern gruppiert werden, um z. B. für die Untergruppe die Bedingungen mit FILTER anzugeben. Ein Graph-Muster-Ausdruck wird folgendermaßen rekursiv definiert[31]:

- Ein elementares Graph-Muster, ein Tripel-Muster (weiter als t bezeichnet), ist ein Tupel $(s, p, o) \in (IB \cup V) \times (I \cup V) \times (IBL \cup V)$
- Angenommen P_1 und P_2 sind Graph-Muster, dann sind die Ausdrücke (P_1 AND P_2), (P_1 OPT P_2) und(P_1 UNION P_2)
- Angenommen P ist ein Graph-Muster und R eine SPARQL-Bedingung, dann ist der Ausdruck (P FILTER R) ebenfalls ein Graph-Muster

Zusätzlich zu oben definierten Bezeichnungen steht V für die unendliche Menge der Variablen, sowie $var(P)$ und $var(R)$ für die Menge der Variablen, die innerhalb des Graph-Musters bzw. der Bedingung vorkommen. Die komplette Liste der zu Verfügung stehenden Bausteine zur Definition der SPARQL-Bedingungen kann unter sparql gefunden werden, dazu gehören vor allem Elemente der $IL \cup V$ Menge, Konstanten, Gleichheits-/Ungleichheitssymbole sowie logische Operatoren.

2.4.1.2 Semantik der SPARQL-Graph-Muster

In diesen Abschnitt stellen wir die Formalisierung der SPARQL-Semantik bezüglich Graph-Muster. Das Resultat der Grundoperation an RDF-Datensätzen, des Muster-Abgleichs, ist eine Abbildung, die als eine partielle Funktion $\mu : V \to IBL$ definiert ist[32]. Die Domäne dieser Funktion, weiter $dom(\mu)$, ist die Untermenge von V, wo μ definiert ist. Das Ersetzen der Variablen im Tripel-Muster t entsprechend der Abbildung μ entstehende Tripel bezeichnen wir als $\mu(t)$. Zwei Mappings μ_1 und μ_2 sind kompatibel, bezeichnet als $\mu_1 \simeq \mu_2$, wenn für alle Variablen[33] $?x \in dom(\mu_1) \cap$

[31] [Pérez u. a. 2009, S. 16:5]
[32] [Le-Phuoc u. a. 2012b, S. 262]
[33] Variablen werden in SPARQL mit einem Fragezeichen gefolgt von einer Zeichenfolge dargestellt.

$dom(\mu_2)$ auch $\mu_1(?x) = \mu_2(?x)$ gilt. Angenommen Ω_1 und Ω_2 sind Mengen der Abbildungen, dann werden folgende Mengenoperationen für diese definiert[34]:

- **Join** $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 und \mu_1 \simeq \mu_3\}$
- **Vereinigung** $\Omega_1 \cap \Omega_2 = \{\mu \mid \mu \in \Omega_1 or \mu \in \Omega_2$
- **Differenz** $\Omega_1 \backslash \Omega_2 = \{\mu \in \Omega_1 \mid \forall \dot{\mu} \in \Omega_2 \mu \neg \simeq \dot{\mu}\}$
- **Linkes Outer Join** $\Omega_1 \mathbb{X} \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \backslash \Omega_2)$

Die Formalisierung der Semantik der Graph-Muster-Ausdrücke basiert auf den oben vorgestellten Definitionen des Graph-Musters und der Operationen auf Mengen der Abbildungen. Die Funktion $[\![P]\!]_D$ gibt die Menge der Abbildungen, die sich aus dem Abgleich des Graph-Musters P mit dem Datensatz D ergeben, zurück. Die Auswertung der Funktion für verschiedene in 2.4.1.1 definierten Graph-Muster ist folgendermaßen definiert[35]:

- Angenommen t ist ein Tripel-Muster $P = t \Rightarrow [\![P]\!]_D = \{\mu \mid dom(\mu) = var(t)$ und $\mu(t) \in D\}$
- $P = (P_1 \text{ AND } P_2) \Rightarrow [\![P]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$
- $P = (P_1 \text{ OPT } P_2) \Rightarrow [\![P]\!]_D = [\![P_1]\!]_D \mathbb{X} [\![P_2]\!]_D$
- $P = (P_1 \text{ UNION } P_2) \Rightarrow [\![P]\!]_D = [\![P_1]\!]_D \cup [\![P_2]\!]_D$
- $[\![(P \text{ FILTER } R)]\!]_D == \{\mu \in [\![P]\!]_D \mid \mu \models R$

Der Ausdruck $\mu \models R$ bedeutet dabei, dass die Abbildung μ der Bedingung R genügt. Die detaillierte Semantik der FILTER-Ausdrücke kann in Pérez u. a. [2009] gefunden werden.

2.4.1.3 SPARQL 1.1

Die oben definierten Formalisierungen bilden die Basis für die SPARQL-Anfragesprache. In August 2012 wurde die Weiterentwicklung der Anfragesprache unter dem Namen SPARQL 1.1[36] als Empfehlung des W3C vorgestellt. Diese bietet eine große Anzahl an Erweiterungen, die in sparql11 gefunden werden können. Die folgende Liste stellt die wichtigsten vor:

[34] [Pérez u. a. 2009, S. 16:5]
[35] [Pérez u. a. 2009, S. 16:5]
[36] sparql11

- **Aggregationsausdrücke**. Diese umfassen die Operatoren SUM, COUNT, AVG, MIN und MAX sowie die Möglichkeit mittels des GROUP BY-Operators die Variablenbindungen zu gruppieren. Für gruppierte Bindungen können ebenfalls Bedingungen unter Einsatz des HAVING-Operators definiert werden.
- **Unteranfragen**. SPARQL 1.1 erlaubt den Einsatz von Anfragen innerhalb des Körpers der Anfrage.
- **Property Path-Ausdrücke**. Diese erlauben reguläre Ausdrücke an der Stelle des Prädikats in einem Tripel-Muster. Dabei ist z. B. die Angabe des Prädikats als einer unendlich langen Prädikatenkette, die das Subjekt mit dem Objekt verbindet erlaubt. Ein so definierter Tripel-Muster stellt einen komplexen Graph-Muster dar.

Bei der Auswertung der Anfrage, die einen Aggregationsausdruck enthält, werden zu den aus dem Abgleich des Graph-Musters entstehenden Variablenbindungen die neu mittels des Ausdrucks definierten hinzugefügt. Die Formalisierung der Aggerationsoperatoren ist in Barbieri u. a. [2010] beschrieben. Die Formalisierung der Graph-Muster-Ausdrücke kann um den Aggregationsausdruck $[\![A]\!]$ folgendermaßen erweitert werden:

$$[\![(PAGGA)]\!] = [\![P]\!] \bowtie [\![A]\!]$$

Am Beispiel einer SELECT-Anfrage ist in 2.5 der Einsatz einiger der oben erwähnten Konstrukte dargestellt. Andere Anfragetypen werden ähnlich konstruiert.

Listing 2.5: Beispiel einer SELECT-Anfrage

```
1 PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2
3 SELECT ?turbine (COUNT(?part) as ?nOfParts)
4 FROM <http://www.siemens.com/Optique/TurbineStatic.rdf>
5
6 WHERE { ?part usecase:isPartOf ?turbine.}
7 GROUP BY ?turbine
8 HAVING COUNT(?part)>7
```

Im Kopf der Anfrage befinden sich außer den vorgestellten Operatoren die Liste der Präfixe sowie die FROM-Klausel, die den RDF-Datensatz angibt. Die Anfrage gibt alle Turbinen, die mehr als sieben Bauteile besitzen und die Anzahl der Bauteile zurück. Innerhalb der SELECT-Klausel wird die Zuweisung eines Aggregationsausdrucks einer neuen Variablen eingesetzt.

2.4.1.4 SPARQL Folgebeziehungsregimes

SPARQL unterstützt Regimes für Folgebeziehungen (engl. *entailment regimes*). Der Einsatz verschiedener Regimes ermöglicht die Erweiterung der SPARQL, so dass weitere Folgeziehungen aus den Daten, die z. B. RDFS- oder OWL-Axiomen enthalten, gewonnen können. Das Regime definiert, welche RDF-Graphen für das Regime als wohlgeformt gelten und Folgebeziehungsregeln zwischen Untermengen der wohlgeformten Graphen[37]. Das folgende Beispiel illustriert den Einsatz der Folgebeziehungsregimes. Die SPARQL-Anfrage in 2.7 gegen den Datengraph in 2.6 würde im einfachen RDF-Folgebeziehungsregime keine Resultate liefern, allerdings bei Erweiterung auf RDFS-Folgebeziehungen, die Bindung der Variable ?mes an das mess:TC255-Measurement_01_57_46 als Resultat zurückgeben. Weitere Details zu Folgebeziehungsregimes sind sparqlent zu finden.

Listing 2.6: Datengraph für Folgebeziehungsregime-Beispiel

```
1 mess:TC255−Measurement_01_57_46 rdf:type diag:TC255−Measurement
2 diag:TC255−Measurement rdfs:subClassOf usecase:HotGasPathMeasurements
```

Listing 2.7: SPARQL-Anfrage für Folgebeziehungsregime-Beispiel

```
1 mess:TC255−Measurement_01_57_46 rdf:type diag:TC255−Measurement
2 diag:TC255−Measurement rdfs:subClassOf usecase:HotGasPathMeasurements
```

Verschiedene Systeme für semantisches Reasoning, wie z. B. **RacerPro**[38] und **Pellet**[39], unterstützen SPARQL-Anfragen gegen die Ontologien. Die Wissensbasen bestehen dabei aus dem **Tbox**[40] und dem **Abox**[41]. Die Wissensbasis wird in einer Beschreibungslogik definiert, z. B. $SHIQ(D)$[42] in RacerPro, die OWL-DL unterliegt. Generell sind in solchen Systemen nur konjunktive[43] Abox-Anfragen möglich[44][45].

[37] sparqlent
[38] racerpro
[39] pellet
[40] Steht für *terminological box* und enthält Konzepte der Wissensbasis und deren Beziehungen
[41] Steht für *assertional box* und enthält Instanzen der Konzepte der Wissensbasis und deren Beziehungen, z. B. als RDF-Datei dargestellt.
[42] Horrocks u. a. [1999]
[43] `http://en.wikipedia.org/wiki/Conjunctive_query`
[44] racerpro
[45] pellet

3 Semantische Datenströme: Basiskonzepte, Sprachen und Anfragenengines

3.1 Grundlagen der Datenstromverarbeitung

Datenströme werden von verschiedenen Quellen fortdauernd produziert und bestehen aus mit einem Zeitstempel markierten Elementen. Da meistens die Ankunftszeit oder Reihenfolge der Daten in Datenströmen vom diese verarbeitenden System nicht definiert oder vorhergesagt werden können, erfolgt die Verarbeitung sequentiell in der Ankunftsreihenfolge[46].
Die meiste Verbreitung in der Datenstromverarbeitung hat das relationale Modell der Daten[47]. Stromelemente sind dabei relationale Tupel, die einem bestimmten Schema entsprechen. Dieses besteht aus einem Satz benannter Attribute. Es existieren auch weitere Modelle, wie z. B. das hierarchische[48], diese sind aber für die Verarbeitung von semantischen Datenströmen weniger relevant, da die meiste Entwicklung in diesem Bereich sich auf das relationale Modell bezieht[49].

3.1.1 Kontinuierliche Anfragen

Da Datenströme andauernd neue Daten liefern, müssen gegen diese kontinuierliche Anfragen definiert werden. Diese produzieren kontinuierlich entsprechend der Eingabe und der Anfragedefinition ein Resultat. Relationen und Datenströme dienen als Eingabe solcher Anfragen. Die Dateneingabe in Form von Relationen wird dem Operator übergeben und entsprechend der eingesetzten Anfragealgebra verarbeitet. Folgende zwei Algebren für kontinuierliche Anfragen werden verwendet[50]:

- **Strom-zu-Strom-Algebra** In dieser Algebra dienen ein oder mehrere Datenströme und Null oder mehrere Relationen als Eingabe jedes Operators. Das Resultat des Operators ist ein neuer Datenstrom. **StreamingSPARQL**[51] setzt diese Algebra ein.

[46] [Le-Phuoc u. a. 2012b, S. 248]
[47] [Arasu u. a. 2006, S. 5]
[48] `http://en.wikipedia.org/wiki/Hierarchical_database_model`
[49] [Le-Phuoc u. a. 2012b, S. 249]
[50] [Le-Phuoc u. a. 2012b, S. 249]
[51] Bolles u. a. [2008]

- **Gemischte Algebra** Diese verfügt über drei Typen der Operatoren: Strom-zu-Relation, Relation-zu-Relation und Relation-zu-Strom. Der Strom-zu-Strom-Operator wird über diese drei realisiert. Beim Einsatz dieser Algebra konvertiert der Operator die Eingabe in Relationen, verarbeitet diese und übergibt in Form eines Stroms an den nächsten Operator. CSPARQL[52], CQELS[53] und SPARQL$_{stream}$[54] Semantiken basieren auf dieser Algebra.

Da Relationen in der Datenstromverarbeitung aktualisierbar sind, werden diese als eine Abbildung der Zeitdomäne $\mathcal{T}$ auf einen endlichen unbeschränkten Satz von Tupeln, die einem bestimmten Schema entsprechen, definiert und mit dem Begriff Relation bezeichnet[55]. Die Relationen $R(t)$, die einem bestimmten Zeitpunkt $\tau \in \mathcal{T}$ entsprechen oder zu dem berechenbar sind, werden dabei als momentane Relationen (engl. *instantaneous relation*) bezeichnet.

Strom-zu-Strom-Operator Diese Operatoren führen andauernd native statische SQL-Anfragen gegen die Datenströme aus und strömen die Resultate. Diese Operatoren werden in allgemeinen SQL-Strukturen, wie SELECT und FROM, spezifiziert[56].

Strom-zu-Relation-Operator Ein Strom-zu-Relation-Operator produziert aus einem Eingabedatenstrom eine Relation mit dem gleichen Schema[57]. Die über die Datenströme definierten Fenster gehören zu solchen Operatoren, z. B. ein zeitbasiertes gleitendes Zeitfenster ist ein Operator, der aus dem Eingabestrom eine Relation generiert. Diese besteht aus den Elementen, deren Zeitstempel innerhalb der Rahmen des Zeitfensters liegt und ist in CQL für S [Range T], wo S ein Datenstrom und T die Fenstergröße, folgendermaßen definiert:

$$R(t) = \{s| < s, t > \in S \wedge (t' \leq t) \wedge (t' \geq max\{t - T + 1; 0\})\}$$

Definition anderer Fenster-Operatoren in CQL können dem Arasu u. a. [2006] entnommen werden.

[52] Barbieri u. a. [2010]
[53] Le-Phuoc u. a. [2011]
[54] Calbimonte u. a. [2010]
[55] [Arasu u. a. 2006, S. 5]
[56] [Le-Phuoc u. a. 2012b, S. 249]
[57] [Arasu u. a. 2006, S. 9]

Relation-zu-Relation Dieser Typ umfasst Operatoren, die aus einer oder mehreren Relationen eine Relation produzieren. Zu jedem Zeitpunkt t kann die momentane Relation der Ausgaberelation aus den momentanen Relationen der Eingaberelationen berechnet werden. DISTINCT-Operator, das aus der Ausgaberelation die Duplikate eliminiert, ist ein Beispiel für einen Operator dieses Typs.

Relation-zu-Strom Operatoren dieses Typs produzieren aus einer Relation einen dem selben Schema entsprechenden Datenstrom. In CQL sind drei Operatoren dieses Typs spezifiziert:

- ISTREAM produziert aus einer Relation einen Strom, Elemente $< s, t >$ dessen enthalten Tupel s aus R(t)-R(t-1).

$$ISTREAM(R) = \bigcup_{t \geq 0}(R(t) - R(t-1)) \times \{t\})$$

- DSTREAM produziert aus einer Relation einen Strom, Elemente $< s, t >$ dessen enthalten Tupel s aus R(t-1)-R(t).

$$DSTREAM(R) = \bigcup_{t > 0}(R(t-1) - R(t)) \times \{t\})$$

- RSTREAM produziert aus einer Relation einen Strom, Elemente $< s, t >$ dessen enthalten Tupel s, die zum Zeitpunkt t in R sind.

$$RSTREAM(R) = \bigcup_{t \geq 0}(R(t) \times \{t\})$$

3.1.2 Ausführungsstrategien

Bei der Ausführung der Operatoren gegen die Datenströme kann die Neuevaluierung unter Einsatz folgender Strategien erfolgen: eifrige oder periodische[58]. Bei der eifrigen Neuevaluierung wird der Operator jedes Mal neu ausgeführt, wenn ein neues Tupel über den Datenstrom ankommt. Dies kann bei hohen Ankunftsraten der Daten zu Leistungsproblemen führen. Bei der periodischen Ausführung der Operatoren, werden Resultate mit einer bestimmten Frequenz produziert. Wenn die Rate der Ankunft der Stromelemente größer ist als die der Ausführung der Operatoren, führt dies zur Auslassung eines Teils der Eingabedaten und demzufolge der entsprechenden Resultate.
Bei der Ausführung der kontinuierlichen Anfragen müssen die Systeme nicht nur auf Ankunft neuer Elemente reagieren sondern auch auf Verfall derer, die schon im System vorhanden sind[59]. Neue Elemente können zur Löschung der Tupel aus vorherigem Resultat führen, sowie der Verfall eines Tupels kann die Hinzufügung eines

[58] [Golab und Özsu 2003, S. 1]
[59] [Le-Phuoc u. a. 2012b, S. 254]

neuen Tupels bedingen, z. B. beim Einsatz des Operators zur Eliminierung von Duplikaten. Weitere Information zur Implementation verschiedener Operatoren sowie Ausführungsmechanismen können in Le-Phuoc u. a. [2012b], Golab und Özsu [2003] und Arasu u. a. [2006] gefunden werden.

3.2 Verarbeitung semantischer Datenströme

3.2.1 Linked Stream Data

Mit der ansteigenden Produktion von Datenströmen in verschiedenen Systemen, wie Sensornetzwerke und soziale Netze, ist auch eine Darstellung dieser in einer dem semantischen Kontext entsprechenden Form nötig. Basierend auf dem erfolgreichen Modell des **Linked Data**[60] wurde das **Linked Stream Data**[61] für die Darstellung von semantischen Datenströmen vorgeschlagen. Dieses dient vor allem der Integration der Datenströmen und der statische Datenquellen.
Linked Stream Data wird üblicherweise als eine Erweiterung von RDF dargestellt, da dieses auch der meistbenutzte Standard für die Darstellung von Linked Data ist[62]. Die wichtigste Anforderung an Daten, die nach diesem Modell bereitgestellt werden, ist die Identifikation der RDF-Datenströme mithilfe von IRIs. Somit kann auf Datenströme über die IRIs zugegriffen oder sie auch als RDF-Knoten eingesetzt werden. Weitere Anforderungen beziehen sich auf die Identifikation der Systemelemente, die Datenströme produzieren, sowie zeitbezogene Identifikation der Datenströme. Diese sind detailliert in Sequeda u. a. [2009] vorgestellt.

3.2.2 Datenmodell

Semantische Datenströme bestehen aus RDF-Tripeln, versetzt mit zeitlichen Annotationen. Wie in 2.1 schon beschrieben wurde, besteht ein RDF-Tripel aus dem Subjekt, Prädikat und Objekt und ist folgendermaßen definiert $(s, p, o) \in IB \times I \times IBL$. Die Annotationen können Intervall- oder Zeitpunkt-basiert sein. Das Intervall definiert das Zeitfenster, in welchem die Aussage[63], z. B. die Position eines Objekts, gültig ist. Bei Zeitpunkt-basierter Darstellung gibt der angegebene Zeitpunkt diesen der Erstellung oder Ankunft des RDF-Tripels. Trotz ihrer Redundanz wird diese in den meisten Fällen bevorzugt und eingesetzt, da sie eine momentane Reaktion auf die ankommenden Daten erlaubt und kein Puffern dieser bis zum Ablauf des Intervalls erfordert, z. B. die Position eines Objekts verändert sich nicht innerhalb

[60] `http://linkeddata.org/`
[61] Sequeda u. a. [2009]
[62] [Le-Phuoc u. a. 2012b, S. 261]
[63] RDF-Tripeln sind Elementaraussagen

von mehreren Stunden. Diese Darstellung wird in allen im Rahmen dieser Arbeit in Betracht gezogenen Systemen eingesetzt.
Folgende Definitionen bilden das gesamte Datenmodel[64]:

- Ein RDF-Tripel ist $(s, p, o) \in IB \times I \times IBL$
- Ein momentaner RDF-Datensatz, bezeichnet als $D(t)$, ist ein Satz der zum Zeitpunkt t gültigen Tripel. Ein RDF-Datensatz ist somit als nach t sortierte Folge $D = [D(t)], t \in \mathbb{N}$ definiert.
- Ein RDF-Strom S ist als eine Multimenge von Elementen der Form $< (s, p, o) : [t] >$ definiert. Wo t ein Zeitstempel ist und (s, p, o) ein RDF-Tripel. $S^{\leq t}$ bezeichnet eine Multimenge der Elemente von S mit einem Zeitstempel $\leq t$.

Da statische RDF-Datensätze, wie auch Relationen in herkömmlicher Datenstromverarbeitung, sich mit der Zeit verändern können, wird, ähnlich den momentanen Relationen in CQL, ein momentaner Datensatz im Punkt 2 definiert. Wenn D(t)=D(t+1) für alle $t \geq 0$ ist, dann wird D als statisch bezeichnet.

3.2.3 Operatoren

Bei Ausführung der kontinuierlichen Anfrage gegen RDF-Daten (Datenströme und Datensätze) wird in aktuellen Systemen mittels eines Fenster-Operators ein momentaner Datensatz erstellt. Auf diesen werden die relationalen Operatoren angewendet und aus dem Resultat ein Datenstrom produziert. Die kontinuierlichen Anfragen basieren somit auf drei bereits in 3.1.1 vorgestellten Typen von Operatoren.
Die Abgleichoperatoren für Tripel-Muster und Fenster über einen momentanen RDF-Datensatz bzw. einen RDF-Datenstrom werden bezogen auf den Zeitpunkt t definiert. Die Definitionen sind den in 2.4.1.1 für SPARQL vorgestellten ähnlich[65]. Für den Tripel-Muster-Abgleich über den Datensatz D lautet diese:

$$[\![P, t]\!]_D = \{\mu \mid dom(\mu) = var(P) \wedge \mu(P) \in D(t)\}$$

Der Fenster-Abgleich-Operator über den Datenstrom S wird folgendermaßen definiert:

$$[\![P, t]\!]^{\omega}_S = \{\mu \mid dom(\mu) = var(P) \wedge \langle \mu(P) : [\dot{t}] \rangle \in S \ \wedge \dot{t} \in \omega(t)\}$$

[64] [Le-Phuoc u. a. 2011, S. 5]
[65] [Le-Phuoc u. a. 2012b, S. 262]

wo $\omega(t) : N \to 2^N$ eine Funktion, die den Zeitstempel t auf möglicherweise unendliche Menge der Zeitstempel abbildet. Diese Definition gibt die Möglichkeit, verschiedene Fenstertypen zu benutzen, z. B. das gleitende Zeitfenster kann als $\omega_{RANGE}(t) = \{\dot{t} \mid \dot{t} \leq t \wedge \dot{t} \geq \max(0, t - T)\}$ definiert werden. Die Formalisierung des sequenziellen Operators kann in Le-Phuoc u. a. [2012b] gefunden werden, diese kann als Basis auch für die Formalisierung weiterer zeitbasierter Beziehungen benutzt werden.

3.2.4 Anfragesprachen

Definition der Anfragesprachen, die auch Anfragen gegen RDF-Datenströme erlauben, baut auf den in 3.2.3 definierten Basisoperatoren auf. Da das Resultat dieser eine Abbildung, das eine partielle Funktion der Form $\mu : V \to IBL$ ist[66], sind die Definitionen der UNION, AND, OPTIONAL und UNION Kompositionen sowie der Aggregationsausdrücke bezogen auf Operationen auf Abbildungsmengen gleich diesen der SPARQL, zu finden in 2.4.1.2.
Derzeit gibt es keinen Standard für kontinuierlichen Anfragen gegen RDF-Daten. Jedes Entwicklerteam der aktuellen Engines zur Verarbeitung solcher setzt eine eigene Sprache, die eine Erweiterung der SPARQL ist, ein. In folgenden Paragrafen sind die SPARQL-Erweiterung für C-SPARQL, CQELS und SPARQL$_{stream}$ vorgestellt.

C-SPARQL [67] erweitert die Anfragesemantik der SPARQL um die FROM STREAM-Klausel. Diese erlaubt ähnlich der FROM-Klausel die Deklaration eines RDF-Datenstroms als der Datenquelle der Anfrage. Der Datenstrom wird für über die Angabe des IRI, das dieses identifiziert, deklariert.

FromStrClause	→	´FROM´ [´NAMED´] ´STREAM´ *StreamIRI* ´[RANGE´ *Window* ´]´
Window	→	*LogicalWindow* | *PhysicalWindow*
LogicalWindow	→	*Number TimeUnit WindowOverlap*
TimeUnit	→	´ms´ | ´s´ | ´m´ | ´h´ | ´d´
WindowOverlap	→	´STEP´ *Number TimeUnit* | ´TUMBLING´
PhysicalWindow	→	´TRIPLES´ *Number*

Die Syntax erlaubt die Definition des Fensters über jeden Datenstrom, die Definition verschiedener Fenster über den gleichen Strom ist aber nicht möglich.
Des Weiteren wird die Syntax für das Registrieren der Anfragen und Datenströme

[66] Die Bezeichnungen entsprechen diesen in 2.4.1.1 vorgestellten.
[67] csparql

bei dem Engine definiert. Eine CONSTRUCT-Anfrage kann somit als ein neuer Datenstrom registriert werden. Die entsprechende Syntax sieht folgendermaßen aus:

Registration → ‘REGISTER STREAM|QUERY‘ QueryName
[´COMPUTED EVERY´ *Number TimeUnit*] ´AS´ Query

SPARQL$_{stream}$ [68] implementiert als einziges der aktuellen Systeme die aus CQL bekannten Relation-zu-Strom-Operatoren: ISTREAM, DSTREAM und RSTREAM. Dies ermöglicht Anfrageszenarien, die mit anderen SPARQL-Erweiterungen nur über eine Kombination mehrerer Operatoren oder gar nicht möglich sind. Ebenfalls wie C-SPARQL wird die Semantik von SPARQL um die FROM STREAM-Klausel erweitert. Im Unterschied zum C-SPARQL wird das Fenster über den Datenstrom definiert als ´FROM *start* TO *end* [STEP *int unit*]´, wo *start* und *end* der Form ‘NOW‘ oder ‘NOW-*Literal*‘ sind[69].

CQELS [70] erweitert nicht die FROM-Klausel, sondern den Graph-Muster-Ausdruck um einen STREAM-Operator. Genauer wird das *GraphPatternNotTriples*-Muster[71] der SPARQL um den *StreamGraphPattern* erweitert[72]. In EBNF Notation[73] lautet dies:

GraphPatternNotTriples → *GroupOrUnionGraphPattern* | *OptionalGraphPattern*
| *MinusGraphPattern* | *GraphGraphPattern* | ***StreamGraphPattern***
| *ServiceGraphPattern* | *Filter* | *Bind*
Window → *Range* | *Triple* | ´NOW´ | ´ALL´
Range → ´RANGE´ *Duration*(SLIDE *Duration*)?
Triple → ´TRIPLES´ INTEGER
Duration → (INTEGER ´d´ | ´j´ | ´h´ | ´m´ | ´s´ | ´ms´ |´ns´)$^+$

Die Semantik von CQELS erlaubt den Einsatz des gleichen Datenstroms mit unterschiedlich definierten Fenstern. Dies stellt eine Möglichkeit dar, Szenarien, die CQL ähnliche Relation-zu-Window Operatoren benötigen, darzustellen.

[68] sparqlstr
[69] [Calbimonte u. a. 2010, S. 4]
[70] cqels
[71] Das Muster beschreibt alle Blocke des Graph-Musters, die keine Tripel-Muster sind.
[72] [Le-Phuoc u. a. 2011, S. 9]
[73] `http://en.wikipedia.org/wiki/Extended_Backus-Naur_Form`

3.2.5 Architekturen der Anfrageengines

Es existieren zwei generelle Typen der Architektur, die von verschiedenen Engines umgesetzt werden: **whitebox** und **blackbox**. Die Whitebox-Architektur sieht eine Entwicklung aller benötigten Bausteine des Systems vor und umfasst eine Implementierung der im Rahmen der Semantik formalisierten Operatoren. Weitere Details zur Implementierung von Operatoren können in Le-Phuoc u. a. [2012b] gefunden werden. In 3.1 sind die wichtigsten Komponenten der Whitebox-Architektur dargestellt.

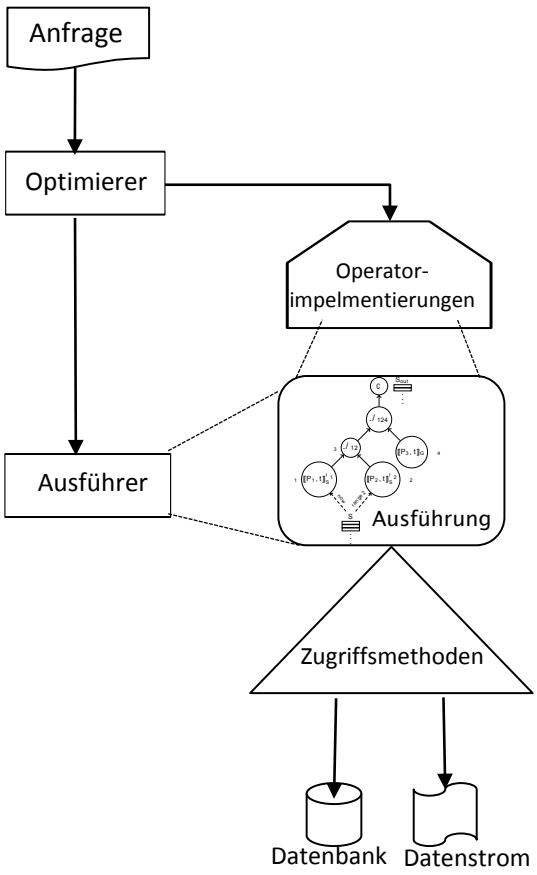

Abbildung 3.1: Whitebox-Architektur
Le-Phuoc u. a. [2012b]

Die Anfragen werden von der Optimierer-Komponente in einen logischen Anfragenplan übersetzt und basierend auf den Operator Implementierungen wird ein Ausführungsplan erstellt. Dieses wird der Ausführer-Komponente übergeben, die eine Ausführung nach diesem Plan durchführt. Die Whitebox-Architektur erlaubt somit eine ständige den Daten- oder Umgebungsveränderungen entsprechende Reoptimierung des Ausführungsplan[74].
Bei der Umsetzung der Blackbox-Architektur werden die verschiedenen Aufgaben

[74] [Le-Phuoc u. a. 2012b, S. 269]

anderen bereits existierenden Komponenten übergeben. Demzufolge bedarf es einer Erweiterung der vorgestellten Whitebox-Architektur um eine Verwaltungskomponente (**Orchestrator**), die verschiedene Unteraufgaben des Ausführungsprozesses an die dafür zuständigen Komponenten weiterleitet. Da die Komponenten auch verschieden Datenformate einsetzten, muss eine Komponente zur Datentransformation implementiert werden. Des Weiteren müssen die in einer SPARQL ähnlichen Sprache definierten Anfragen übersetzt werden. Die Zusammenarbeit der Komponenten in einer Blackbox-Architektur ist in 3.2 dargestellt.

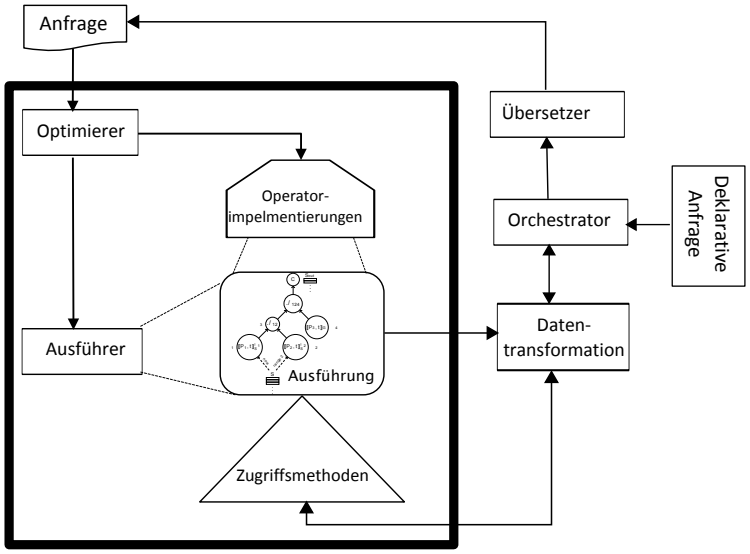

Abbildung 3.2: Blackbox-Architektur
Le-Phuoc u. a. [2011]

C-SPARQL System[75] implementiert die Blackbox-Architektur. Das Orchestrator übernimmt im C-SPARQL die Rolle des Übersetzers und transformiert C-SPARQL Anfragen in einen statischen (in SPARQL definiert) und dynamischen (in CQL definiert) Teil[76]. Diese entsprechen den Teilen der Anfrage, die sich auf einen RDF-Datensatz bzw. RDF-Datenstrom beziehen. Der statische Teil wird von dem **Sesame**[77] Anfrageengine und der dynamische von dem **STREAM**[78] Datenstrommanagementsystem verarbeitet. Die eigentliche Ausführung der Anfragen erfolgt in C-SPARQL periodisch. Weitere Details über die Implementierung des C-SPARQL Engines sind in Barbieri u. a. [2010] zu finden.

[75] csparql
[76] [Barbieri u. a. 2010, S. 7]
[77] `http://www.openrdf.org/`
[78] `http://infolab.stanford.edu/stream/`

SPARQL$_{stream}$ [79] unterstützt Anfragen nur gegen RDF-Datenströme und implementiert ebenfalls die Blackbox-Architektur. Die SPARQL$_{stream}$ Anfragen werden vom System in **SNEEql**[80] Sprache übersetzt. Da eine SPARQL$_{stream}$ Anfrage bezogen auf eine Ontologie definiert ist und in eine auf Datenquellen bezogene übersetzt werden muss, bedarf die Übersetzung einen Satz von vordefinierten Abbildungen[81]. Diese werden in **S$_2$O**[82] Abbildungssprache definiert. Die Ausführung der Anfragen erfolgt periodisch. Weitere Details über die Implementierung des SPARQL$_{stream}$ Engines sind in Calbimonte u. a. [2010] zu finden.

CQELS [83] ist das einzige Engine, das nach einer adaptiven Whitebox-Architektur entwickelt wurde. Dies ermöglicht eine Optimierung des Ausführungsplans der Anfrage zu ihrer Laufzeit. Das Elementarobjekt des CQELS-Systems ist ein Tripel[84]. Dies ermöglicht den nativen Zugriffsmethoden auf RDF-Daten einen hohen Datendurchsatz zu erzielen. Die Ausführung der Anfragen mit CQELS erfolgt nach der Strategie der eifrigen Ausführung[85]. Weitere Details über die Implementierung des CQELS Engines sind in Le-Phuoc u. a. [2011] und Le-Phuoc u. a. [2012b] zu finden.

[79] sparqlstr
[80] `http://www.semsorgrid4env.eu/index.php/snee`
[81] [Le-Phuoc u. a. 2012b, S. 274]
[82] [Calbimonte u. a. 2010, S. 5]
[83] cqels
[84] [Le-Phuoc u. a. 2012b, S. 275]
[85] Die Anfrage wird immer nach Erhalt neuer Daten erneuert ausgeführt.

4 Verwandte Arbeiten

In diesem Kapitel werden verschiedene Entwurfsvorschläge zur Entwicklung und bereits existierende Benchmarks vorgestellt, die ähnliche oder gleiche Testziele verfolgen.

4.1 Linear Road Benchmark

Das **Linear Road Benchmark** wurde von den Entwicklern der **Aurora** und **STREAM** Datenstrommanagementsysteme zur Auswertung solcher und ähnlicher Systeme entworfen. Eines der Ziele war auch die Darstellung der Vorteile von DSMS gegenüber den für Verarbeitung von Datenströmen angepassten Datenbankmanagementsystemen. Die Ergebnisse zeigten den Mindestleistungsvorteil mit dem Faktor 5.
Das Linear Road ist das einzige existierende Benchmark, das in Rahmen dieser Arbeit gefunden wurde, das sich mit der Verarbeitung von nichtsemantischen Datenströmen auseinandersetzt. Das Linear Road eignet sich nicht für Auswertung von Verwaltungssystemen und Engines für semantische Datenströme, da es auf dem relationalen Datenmodel aufbaut und Aspekte wie z. B. Reasoning nicht berücksichtigt[86]. Aus dem Benchmark können aber die allgemeinen Vergleichsrichtlinien für Datenstromverarbeitung abgeleitet werden.

Werkzeuge Um die Implementation zu ermöglichen, stellt Linear Road einen Satz von Werkzeugen zu Verfügung. Dieses umfasst Generatoren für statische historische und dynamische Verkehrsdaten sowie einen zur Weitergabe der generierten Daten an das System und Validierung der Anfragenresultate.

Anfragen Linear Road ist angesichts aller Anforderungen ein hochkomplexes Benchmark. Es fasst einen Satz komplexer Anfragen, z. B. gleitendes Zeitfenster, Aggregation und verschiedene Arten von Joins. Zusätzlich zu den kontinuierlichen Anfragen bezüglich Kontostand des Fahrzeugs und geschätzter Reisezeit muss das System auch historische Anfragen bezüglich z. B. Gebührenhistorie ausführen können[87]. Da

[86] [Zhang u. a. 2012, S. 2]
[87] [Zhang u. a. 2012, S. 14]

es keine einheitliche Anfragesprache für die Datenströme gab, sind alle Anfragen in Prädikatenlogik definiert.

4.1.1 Usecase

Der Anwendungsfall repräsentiert ein komplexes System zur Verwaltung und Anpassung von Straßenbenutzungsgebühren. Die Umgebung stellt ein Autobahnnetz mit sämtlichen sich auf ihm bewegenden Fahrzeugen dar. Diese übermitteln ihre Position in fest definierten Zeitabständen. Je nach daraus ermittelter Straßensituation werden Gebühren für jedes Segment der Autobahn errechnet und an das Fahrzeug zurückgegeben. Linear Road beinhaltet eine eindeutige Beschreibung der Datenstruktur, die in einem das Benchmark implementierenden System verarbeitet und ausgegeben wird. Diese sind in Form verschiedener Datentupeln dargestellt.

4.1.2 Kriterien und Auswertung

Die Auswertung mit Linear Road Benchmark basiert auf drei Kriterien[88]:

- **Antwortzeit**. Mittelwert oder Maximum der Zeitfenstern zwischen dem Einfügen eines Datentupels in das System und der Generierung des darauf basierenden Outputs.
- **Unterstützte Anfragenlast** bezeichnet die Menge der Eingabedaten, die das System verarbeiten kann unter Erfüllung bestimmter Einschränkungen bezüglich Antwortzeit und Korrektheit.
- **Korrektheit**. Aufgrund der Abhängigkeit des Resultats von mehreren Parameter, z. B. Ankunftsreihenfolge der Datentupeln, werden für jede Anfragen mehrere Antworten als korrekt definiert.

Auf der Basis dieser Kriterien ist das sogenannte L-Rating als Anzahl der Autobahnen, dessen Daten das testende System unter Einhaltung von erwähnten Einschränkungen verarbeiten können, definiert.

4.2 LUBM

LUBM ist ein Benchmark für die Auswertung von wissensbasierten Systemen des Semantischen Web in Bezug auf den Einsatz in komplexen OWL-Anwendungen[89]. Im Rahmen des Benchmarks werden Kriterien für die Definition der Testanfragen

[88] [Arasu u. a. 2004, S. 481]
[89] [Guo u. a. 2005, S. 4]

und Metriken zur Auswertung der Leistungsfähigkeit der Systeme vorgestellt. Diese können als Basis für die Ausarbeitung dieser in Benchmarks für unterschiedliche Typen der Anfragenengines des Semantischen Web dienen. Andere Aspekte, mit denen sich das Benchmark auseinandersetzt, die Auswertung der Systeme sowie die praktische Umsetzung des Benchmarks sind für diese Arbeit weniger relevant und sind im Detail in Guo u. a. [2005] zu finden. Im Rahmen dieser Arbeit werden nur die angesprochenen Kriterien vorgestellt.

4.2.1 Kriterien zur Definition von Testanfragen

LUBM beinhaltet vierzehn Anfragen die in SPARQL formuliert sind. Folgende Faktoren wurden bei der Definition in Betracht gezogen[90]:

- **Große der Eingabe**. Prozentualer Anteil der Klasseninstanzen des ganzen Datensatzes, die für die Anfragen relevant sind. Als groß wird die Eingabe bezeichnet, die mehr als 5 Prozent des Datensatzes umfasst.
- **Selektivität**. Prozentualer Anteil der Klasseninstanzen der Eingabedaten im Resultat der Anfrage. Als hoch selektiv werden die Anfragen bezeichnet, deren erwartetes Resultat weniger als 10 Prozent der Eingabedaten umfasst.
- **Komplexität**. Sie wird als Anzahl der in die Anfrage involvierten Klassen und Eigenschaften definiert.
- **Vorausgesetztes Hierarchiewissen**. Dieses beschreibt, ob Kenntnisse über die Hierarchie der Klassen und Eigenschaften für die Ausführung nötig sind, damit diese ein vollständiges Resultat produziert.
- **Vorausgesetzte logische Inferenz**. Dieses beschreibt, ob für die Ausführung der Anfrage Inferenz aus Klassen- und Eigenschaftsbeziehungen, z. B. owl:TransitiveProperty, erforderlich ist, damit diese ein vollständiges Resultat produziert.

Das Resultat wird als vollständig definiert, wenn dieses alle aus der Wissensbasis erschließbaren Daten, die den Anfragenkriterien entsprechen, enthält.

4.2.2 Leistungsmetriken

Außer den bereits aus anderen Benchmarks bekannten Kriterien, wie Antwortzeit, werden manche davon im Rahmen des Benchmarks neu definiert oder erst eingeführt. Die Definition dieser bezieht sich auf statische Daten und einmalige Anfragen und kann nicht in der Form für kontinuierliche Anfrage übernommen werden.

[90] [Guo u. a. 2005, S. 7]

Vollständigkeit Das System wird vollständig genannt, wenn dieses alle aus der Wissensbasis erschließbaren Antworten, die den Anfragenkriterien entsprechen, produziert. Dabei sind die einzelnen Antworten Bindungen der Variablen an entsprechende Instanzen aus dem Datensatz[91]. Als Maß der Vollständigkeit wird der von dem System produzierte prozentuale Anteil des erwarteten vollständigen Resultats definiert.

Kombinierte Metriken LUBM führt drei kombinierte Metriken ein. Die Metriken erlauben eine absolute Einstufung der einzelnen Systeme, sind aber abstrahiert von eventuellen Details und sind somit nicht in jedem Fall aussagekräftig[92].

4.3 NEXMark

Das **NEXMark** Benchmark war als eine Erweiterung in Bezug auf Verarbeitung von Datenströmen des **XMark**[93] Benchmarks geplant, wurde aber nach unserem besten Wissen nicht veröffentlicht. Der Entwurf des Benchmarks stellt den Anwendungsfall, einen Satz von Testanfragen und die Vorschläge für Metriken zur Auswertung der Leistungsfähigkeit der Systemen vor.

Anfragen Die Anfragen des Benchmarks bilden zwar die Anforderungen an DSMS ab, sind aber in einer freien Interpretation der CQL-Sprache definiert und in einer kleinen Anzahl verfügbar.

4.3.1 Anwendungsfall

Der Anwendungsfall repräsentiert die Abläufe in einem Auktionshaus. Das Model umfasst folgende Business-Objekte, die verschiedene Datenströme erzeugen:

- Personen, die einen Datenstrom mit Registrierungsdaten erzeugen.
- Verkäufer, die Daten des Gegenstands, wie Beschreibung und Endzeit der Auktion, erzeugen.
- Käufer, die einen Datenstrom von Geboten erzeugen.

Zusätzlich werden in einem statischen Datensatz die sich selten verändernden Daten, wie Kategorien der Gegenstände, gespeichert.

[91] [Guo u. a. 2005, S. 9]
[92] [Guo u. a. 2005, S. 9]
[93] xmark

4.3.2 Leistungsmetriken

Für die Auswertung der Systeme wurden im Rahmen des Entwurfs drei Metriken vorgeschlagen: Rate des Eingangsdatenstroms, Übereinstimmung der Ausgabe und Antwortzeit. Bei der Definition des Maßes für die Übereinstimmung des Resultats wird die Korrektheit angenommen und nur die Verzögerung für die Verarbeitung der Daten in Betracht gezogen. Das Maß wird als Produkt der Verzögerung und der Differenz des erwarten Wertes und des vom System aktuell dargestellten definiert. In 4.1 wird das Maß für eine Beispielanfrage, die den aktuellen Preis eines Gegenstands zurückgibt, dargestellt.

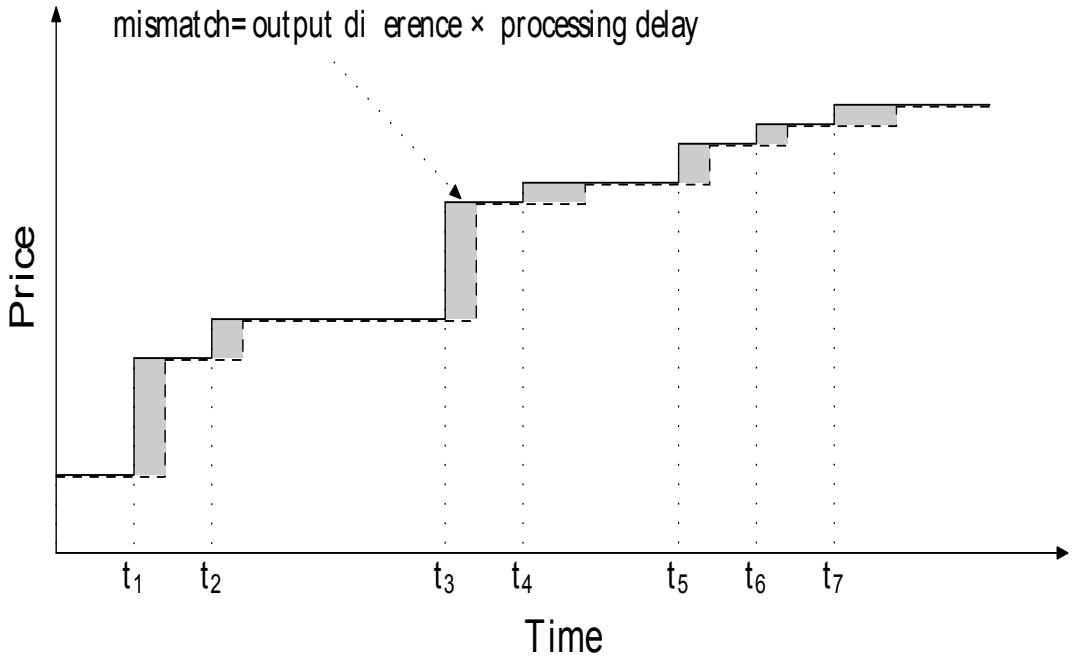

Abbildung 4.1: Nichtübereinstimmung des Resultats für eine Beispielanfrage
Tucker und Maier [2002]

4.4 SRBench

SRBench ist das erste öffentliche Benchmark für strRS Engines, das einen Satz fest definierter Anfragen bereitstellt und auf einem großen Satz reeller Daten aufbaut. Das Ziel der Entwickler[94] war die erste Vorstellung den funktionalen Möglichkeiten der existierenden strRS Engines und ein Vergleich dieser[95]. Es wurden C-SPARQL, SPARQL$_{Stream}$ und CQELS Engines zum Vergleich gezogen. In Zusammenarbeit[96]

[94] Das Entwicklerteam besteht unter allem aus den Entwicklern des SPARQL$_{Stream}$ Engines
[95] [Zhang u. a. 2012, S. 3]
[96] [Zhang u. a. 2012, S. 15]

mit den Entwicklern dieser Engines konnten in Rahmen dieses Benchmarks alle Engines getestet und eine Auswertung durchgeführt werden. Resultate dieser Tests sind in 4.4.3 zu finden.

4.4.1 Daten

SRBench stellt keinen konkret definierten Anwendungsfall vor sondern baut auf den Datensätzen aus dem **Linked Open Data** Cloud[97]. Damit die Auswertung aussagekräftig ist und die Erfüllung der Anforderung, wie Reasoning, an strRS Engines getestet werden können, bedarf es mehrerer Datensätze, die miteinander verknüpft sind. Folgende Auflistung beschreibt die Datensätze sowie ihre Beziehungen, diese sind ebenfalls in 4.2 dargestellt:

- **GeoNames**-Ontologie ist eine freie Datenbank, die Daten der über 8 Millionen Orte der Welt beinhaltet.
- **LinkedSensorData** besteht aus zwei Teilsätzen, die von **Kno.e.sis**[98] in Form von LinkedStreamData transformiert wurden und Definitionen entsprechend der O&M-OWL Ontologie[99] enthalten:
  - **LinkedSendorMetada**, das Daten über rund 20000 Messstationen umfasst. Jede Messstation verfügt über mehrere Sensoren, die Messungen der Temperatur, Windstärke usw. liefern. Zusätzlich werden Position der Station und eine Verlinkung dieser mit einer im GeoNames Datensatz erfasst.
  - **LinkedObservationData**, das während verschiedener massiver Wirbelstürme in den Vereinigten Staaten von den Wetterstationen aus dem *LinkedSensorMetada*-Satz aufgenommene Messdaten beinhaltet.
- **DBpedia** ist eine Wissensbasis, die aus gemeinschaftlich extrahierten strukturierten Daten aus Wikipedia besteht. DBpedia enthält über 85000 Verlinkungen mit dem GeoNames-Datensatz.

Werkzeuge Da SRBench auf mehreren großen reellen Datensätzen aufbaut, braucht die Implementation des Benchmarks keine Datensimulation.

[97] loddia
[98] knoesis
[99] lsd

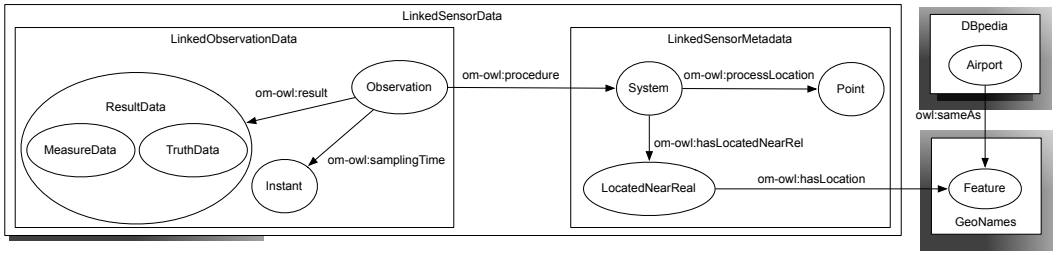

Abbildung 4.2: SRBench-Datensätze und deren Beziehungen
Zhang u. a. [2012]

4.4.2 Anfragen

SRBench bietet einen umfangreichen Satz von Testanfragen in natürlicher Sprache sowie Umsetzung dieser in drei mit den Engines zusammenhängenden Erweiterung von SPARQL an. Die Anfragen stellen sinnvolle reelle Fragen dar, z. B. ob es derzeit regnet, und umfassen den Einsatz nahezu allen Funktionalitäten und Operatoren von SPARQL 1.0 und 1.1, die für die Verarbeitung von semantischen Datenströmen relevant sind.
Zusätzlich wurden Anfragen zum Testen von Reasoning-Fähigkeiten der Engines definiert. Da Engines auf den jeweiligen Erweiterungen von SPARQL basieren und SPARQL keine Reasoning-Sprache ist, wurden die Anfragen, die Reasoning bezüglich der in RDF codierten Ontologien für die Ausführung benötigen, mittels zusätzlicher Graph-Muster unter Einsatz von PropertyPath-Ausdrücken definiert[100]. Einige Anfragen setzen auch den Einsatz von aus CQL bekannten Operatoren Istream und Dstream voraus.
Der Satz von Anfragen enthält Anfragen gegen gleichzeitig mehrere statische Datensätze und jeweils nur einen Datenstrom, der Einsatz von mehreren Datenströmen ist aber in keiner Anfrage berücksichtigt.

4.4.3 Auswertung

Die Resultate der Auswertung können in drei Kategorien aufgeteilt werden: Unterstützung der SPARQL Funktionen, unterschiedlicher reeller Anfrageszenarien und des Reasonings.

SPARQL 1.0 und 1.1 Alle Engines unterstützten die meisten Basis-Anfragen, außer der ASK-Anfragen, zeigten aber nahezu keine Unterstützung von SPARQL 1.1-Features. Mit dem SPARQL$_{Stream}$ Engine konnten wegen fehlender Unterstützung

[100] [Zhang u. a. 2012, S. 7]

des GROUP BY-Operators und Aggregation nur drei aus insgesamt siebzehn Anfragen ausgeführt werden. CSPARQL und CQELS konnten jeweils sieben Anfragen verarbeiten. Die meisten anderen Anfragen fielen wegen fehlender Unterstützung der Property Path-Ausdrücke aus.

Abdeckung verschiedener Szenarien SPARQL$_{Stream}$ bietet als einziges Engine die Implementation der Istream und Dstream Operatoren. Außer diesen Operatoren fehlt CSPARQL die Möglichkeit den gleichen Strom zwei mal in einer Anfrage einzusetzen. Aus diesem Grund konnten manche Anfragen in CSPARQL gar nicht definiert werden.

Reasoning Von allen drei Engines unterstützt nur CSPARQL einfaches Reasoning auf der Basis von einfachen RDF-Folgebeziehungen[101].

Die Auswertung, wie schon oben angesprochen wurden, bezog sich nur auf die funktionalen Möglichkeiten der Engines. Diese befinden sich allerdings noch in anfänglicher Entwicklung, somit wurden von den Entwicklern des Benchmarks die Resultate als zu erwartende bezeichnet[102]. Die Auswertung anderer Aspekte, wie Korrektheit und Datendurchsatz , wurde als weitere Arbeit angesprochen aber, wahrscheinlich wegen des kleines Umfangs der unterstützten Funktionalitäten, nicht vorgenommen.

4.5 LSBench

LSBench wurde von dem Entwicklerteam des CQELS Engines vorgestellt. Das Benchmark umfasst drei Arten von Tests: Funktionalität, Korrektheit und Leistung, und es wurden drei Engines zum Vergleich gezogen: CSPARQL, CQELS und ETALIS. Im Vergleich zu SRBench wurde mehr Wert auf Leistungsmerkmale der Engines gesetzt und die funktionalen Tests umfassen nur die Basis-Muster und Operatoren. Vor allem wurden bei den Tests folgende Charakteristiken der Engines in Bezug genommen[103]:

- **Semantik**. Auf Grund der fehlenden Standardanfragesprache für Datenströme weisen die den Engines entsprechenden Erweiterungen von SPARQL deutliche Unterschiede auf, die bei der Definition von Anfragen zur Geltung kommen.

[101] [Zhang u. a. 2012, S. 13]
[102] [Zhang u. a. 2012, S. 12]
[103] [Le-Phuoc u. a. 2012a, S. 2]

- **Ausführungsmechanismen der Engines**. Engines benutzen verschiedene Strategien bei der Ausführung von Anfragen, dies hat auch unterschiedliche Resultate zur Folge. CSPARQL setzt die periodische Ausführung von Anfragen ein, wobei CQELS und ETALIS dem Prinzip der eifrigen Ausführung (eager execution) folgen, i.e. die Anfrage bei Ankunft neuer Daten immer neu ausgeführt wird.
- **Ausführungsumgebung**. Der Einfluss der Ausführungsumgebung und verschiedener Testparameter wird berücksichtigt.

4.5.1 Anwendungsfall

Der Anwendungsfall repräsentiert einen Teil des Prozesses in einem sozialen Netzwerk. Der repräsentierte Prozess umfasst die statischen Metadaten der Benutzer, wie Profile und Beziehungen zu anderen Benutzern, und mehrere Datenströme. Diese bilden verschiedene Aktivitäten der Benutzer ab, wie Erstellung eines Eintrags, Übermittlung der Position über GPS, Hinzufügen von Bildern und Reaktion anderer Benutzer. Der Aufbau des Anwendungsfalls ist in 4.3 dargestellt.

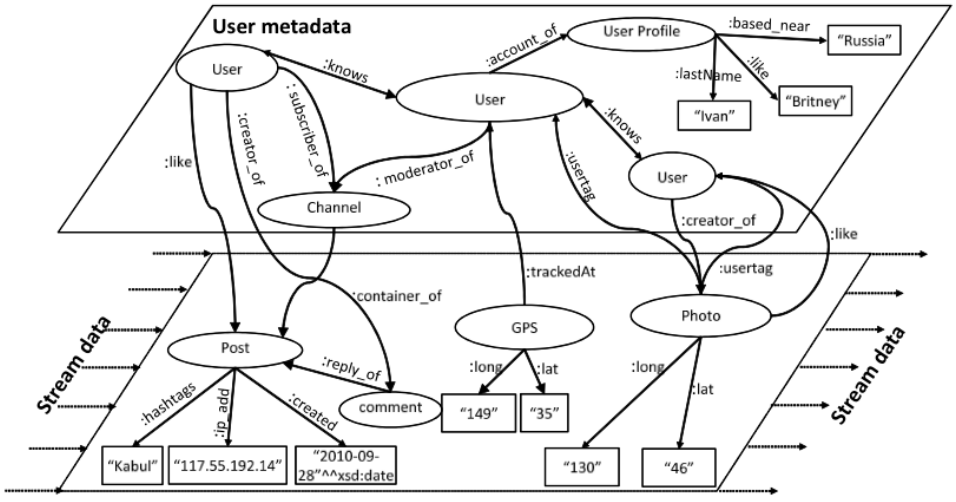

Abbildung 4.3: LSBench Datensätze und deren Beziehungen
Le-Phuoc u. a. [2012a]

Werkzeuge Das Benchmark setzt künstliche Daten, die mittels des sogenannten **S2Gen** (Stream Social network data Generator) generiert werden. Die Stromdaten werden nach dem Prinzip des gleitenden Fensters[104] generiert. Dabei wird ein Fenster, das eine bestimmte Anzahl von Benutzern umfasst, über den ganzen Satz der

[104] [Le-Phuoc u. a. 2012a, S. 4]

Benutzer bewegt. Für die Benutzer, die sich in dem Fenster befinden werden dabei sämtliche Aktivitäten generiert. Die unterschiedlichen Aktivitäten werden dann nach Zeitstempeln sortiert und als Datenströme an das Engine übergeben. Bei der Generierung von Aktivitäten wurden verschiedene Verteilungen eingesetzt, um realitätsnahe Daten zu erstellen[105]. Der Datengenerator steht unter lsbench zu Verfügung.

4.5.2 Anfragen

Im Vergleich zur SRBench decken die insgesamt zwölf Anfragen des LSBench einen wesentlich kleineren Umfang an möglichen zu testenden Funktionalitäten der Engines. Die Definition dieser erfolgte in natürlicher Sprache ist aber wegen des fehlenden Kontexts nicht eindeutig. Die Formulierung der Anfragen in entsprechenden SPARQL-Erweiterungen, auf denen die im nächsten Abschnitt vorgestellte Auswertung basiert, umfasst wiederum nur sechs aus zwölf Anfragen. Diese sind die von allen Engines unterstützten Anfragen. Die Hinweise auf die Zugehörigkeit zur Definition in natürlicher Sprache fehlt ebenfalls.
Zwei weitere wesentliche Unterschiede zum SRBench bestehen in:

- LSBench setzt Anfragen gegen gleichzeitig mehrere Datenströme verschiedener Struktur und statischer Datensätze
- Keine Anfragen beanspruchen die Reasoning-Fähigkeiten der Engines.

4.5.3 Auswertung

Auswertung und Vergleich der drei Systeme (wobei ETALIS ein Event-Processing System ist) erfolgte, wie schon erwähnt wurde, bezüglich drei verschiedener Aspekte. Die funktionale Auswertung folglich dem kleinen Umfang der Abdeckung der SPARQL-Features und der Reasoning-Fähigkeiten ist nur bedingt aussagekräftig und bestätigt einen Teil der SRBench Auswertung.

4.5.3.1 Korrektheit

Die Auswertung der Engines in Bezug auf Korrektheit der Resultate basiert in Rahmen des LSBench auf einem innerhalb der Arbeit definierten Maß der Nichtübereinstimmung dieser. Das Maß ist als prozentuale Abdeckung des Resultats eines Engines von diesem des anderen bei der Ausführung einer Anfrage definiert[106]. Die

[105] [Le-Phuoc u. a. 2012a, S. 4]
[106] [Le-Phuoc u. a. 2012a, S. 7]

genaue Definition, die auch Unterschiede in Ausführungsmechanismen in Betracht zieht, ist in Le-Phuoc u. a. [2012a] zu finden. Trotz der manchmal gravierenden(bis 87,66%)[107] Nichtübereinstimmung der Resultate auch bei gleicher Anzahl dieser, wird bei dem Vergleich von der Korrektheit dieser ausgegangen.
Bei den einfacheren Anfragen war die gegenseitige Abdeckung der Resultate aller Engines zwischen 98 und 99 Prozent. Das Resultat von CSPARQL beinhaltet dabei wegen des periodischen Ausführungsmechanismus des Engines mehrfach die gleichen Tripel. Die Resultate der Auswertung komplexerer Abfragen haben gezeigt, dass nur CQELS nahezu alle Resultate der anderen Engines abdeckt, aber nicht umgekehrt.

4.5.3.2 Leistung

Bei der Auswertung der Leistungsfähigkeit der Engines wurde der Datendurchsatz und ihre Skalierbarkeit gemessen. Als Maß des Datendurchsatzes wurde der Quotient der Eingabegröße und der Zeit, die für die Verarbeitung vom Engine verbraucht wurde, definiert. ETALIS und CQELS haben einen Vorsprung von mehreren Größenordnungen im Vergleich zu CSPARQL gezeigt.
Die Skalierbarkeitstests haben gezeigt, dass CQELS als einziges Engine unter wachsender Größe der statischen Eingabedaten kleinere Leistungsverluste zeigt. CSPARQL zeigte schon bei weniger komplexen Anfragen gegen statische Datensätze mit über 1 Million Tripel einen deutlichen Leistungsverlust. Bei der gleichzeitigen Ausführung mehrerer Anfragen zeigten alle Engines einen Leistungsverlust. Die Resultate der Auswertung in Bezug auf den maximalen Datendurchsatz in Abhängigkeit von der Anzahl der Tripel im statischen Datensatz bzw. der Anzahl der Anfragen sind in 4.4 bzw. 4.5 vorgestellt. Die Werte des maximalen Durchsatzes sind in logarithmischer Darstellung aufgetragen.

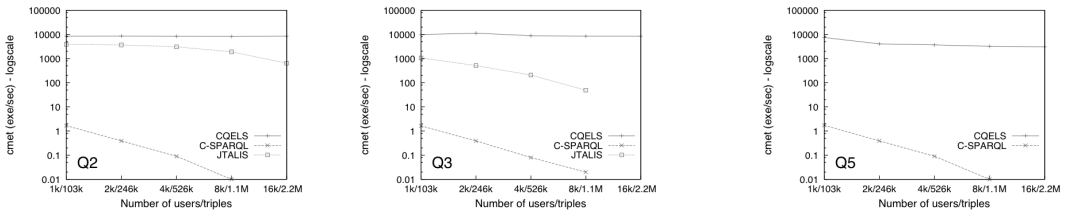

Abbildung 4.4: Vergleichbarer maximaler Datendurchsatz abhängig von der Größe des statischen Datensatzes
Le-Phuoc u. a. [2012a]

Zusammenfassung Keine der vorgestellten Benchmarks bietet die Möglichkeit einer vielseitiger Auswertung der strRS Engines zur Verfügung. LSBench und SR-

[107] [Le-Phuoc u. a. 2012a, S. 9]

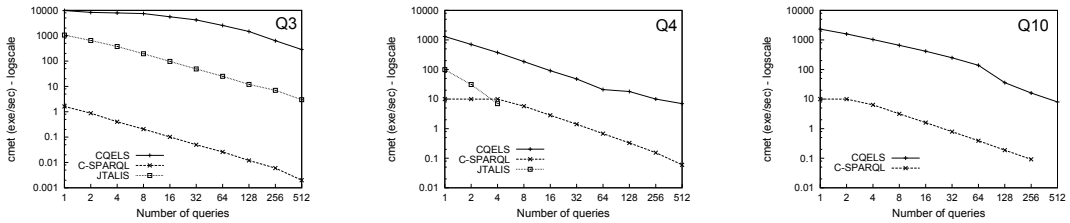

Abbildung 4.5: Vergleichbarer maximaler Datendurchsatz bei der Ausführung mehrerer Anfragen
Le-Phuoc u. a. [2012a]

Bench stellen zwar nachvollziehbare Anwendungsfälle zur Verfügung, decken aber nur einen funktionalen bzw. leistungsbezogenen Teilbereich der Auswertung ab. Dabei sind auch die Anfragen von LSBench nicht komplett nachvollziehbar. Das LUBM Benchmark ist für statische RDF-Daten konzipiert und zieht die für die Datenstromverarbeitung relevanten Aspekte nicht in Betracht. Das Linear Road Benchmark stellt zwar einen komplexen Anwendungsfall dar, mit umfangreichen Möglichkeiten DSMS zu testen, ist allerdings für DSMS, die Anfragen gegen relationale Datenströme verarbeiten, entwickelt und für graphbasierte Daten ungeeignet.

5 Benchmark: Entwurf und Implementierung

Dieses Kapitel widmet sich der Entwicklung der Bausteine, die die Basis für ein Benchmark bilden. Das Benchmark soll die Auswertung von RDF/SPARQL Engines zur Verarbeitung von Datenströmen ermöglichen. Diese umfasst mehrere Daten-, Anfragen- und Datenstrom-Generatoren, einen Satz von Anfragen zum Testen der Fähigkeit des Engines verschiedene Anfragenmuster zu bearbeiten und eine **Flex/BlazeDS**-Webanwendung zur Darstellung der Anfrageresultate. Des Weiteren wird der Anwendungsfall vorgestellt, auf dem die Testumgebung aufbaut.

5.1 Anwendungsfall: Siemens Gasturbinen

Der Anwendungsfall bezieht sich auf den Arbeitsablauf einer Gasturbine und basiert auf den von der Siemens AG[108] innerhalb des OPTIQUE-Projekts[109] bereitgestellten Daten. Diese umfassen zwei Ontologien, die einen Teil der Gasturbinen-Wissensdomäne beinhalten. Die optique-pattern.owl-Ontologie fasst das Basiswissen, z. B. NumericalFeature und SystemElement Klassen, zusammen. Die Diagnosis.owl-Ontologie importiert die Basisontologie und beschreibt konkret die Turbinenwissensdomäne, z. B. TemperatureSensor-Klasse.
Ebenfalls wurde ein Satz von Beispieldaten, die von einer Gasturbine innerhalb eines Zeitraumes erzeugt werden, bereitgestellt. Basierend auf diesen Daten wurde das Verhalten der Brennkammer einer Gasturbine und der zusammenhängenden Komponenten simuliert. Die dafür relevanten Daten wurden in zwei Schichten unterteilt: statische Daten und dynamische Datenströme.

5.1.1 Anpassung der bereitgestellten Daten

Um die vorhandenen Daten für das Testszenario zu benutzen, müssen sie angepasst werden. Im Rahmen dieser Anpassung wurde eine der Ontologien um zwei weitere Eigenschaften erweitert und die Messdaten wurden interpoliert.

[108] http://www.siemens.com/entry/cc/en/
[109] optique

5.1.1.1 Erweiterung der Ontologie

Die gegebenen Ontologien beschreiben zwar eine Beziehung zwischen einer Messung und ihrem Wert, es fehlten aber die Eigenschaften, die eine Beziehung zwischen der Messung und der messenden Komponente der Turbine darstellen. Aus diesem Grund wurde die Basisontologie um diese Eigenschaften erweitert. Diese wurden, wie in 5.3 zu sehen ist, als zwei zu einander inverse funktionale Eigenschaften definiert. Als Beispiel für die Definition wurde die Umsetzung logisch ähnlicher Beziehungen in **O&M-OWL** (**SemSOS**)[110] benutzt. Die angepassten Ontologien sind im Appendix unter 8.2 zu finden.

Listing 5.1: Erweiterung der Optique-Ontologie

```
<owl:ObjectProperty rdf:about="#generatedFeature">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    Inverse of element.
  </rdfs:comment>
    <rdfs:range rdf:resource="#Feature"/>
    <rdfs:domain rdf:resource="#SystemElement"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#element">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    element
  </rdfs:label>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    This property is the description of an System Element used to generate it.
  </rdfs:comment>
    <rdfs:domain rdf:resource="#Feature"/>
    <rdfs:range rdf:resource="#SystemElement"/>
    <owl:inverseOf rdf:resource="#generatedFeature"/>
</owl:ObjectProperty>
```

5.1.1.2 Vorbereitung der Messdaten

Für den Anwendungsfall wurden die Messungen der Temperatursensoren in den Brennkammern ausgewählt. Der Datensatz umfasst Messwerte der sechs Temperatursensoren innerhalb eines ganzen Tages. Der Zeitabstand zwischen verschiedenen Messwerten beträgt eine Minute. Diese sind aber laut Aussage des Siemens Mitarbeiters die Mittelwerte, die den Operatoren dargestellt werden. Die reellen Sensor-Messungen werden 10 Mal pro Sekunde durchgeführt. Diese Messwerte müssten aus

[110] lsd

den vorhandenen interpoliert werden, die Methode, die bei der Berechnung benutzt wird, ist aber zum Zeitpunkt der Erstellung dieser Arbeit unbekannt. Da nur die Mittelwerte vorhanden sind, lassen sich keine aussagekräftigen Rückschlüsse über die Verteilung der zu Grunde liegenden Daten treffen, besonders ob diese normal verteilt sind oder nicht. Aus diesem Grund wurden die Zwischenwerte aus den vorhandenen mittels linearer Interpolation ermittelt.
Die lineare Interpolation der Daten wurde in einer Java-Klasse implementiert. Der originale Datensatz beinhaltet Messwerte in Form einer Excel-Tabelle. Um diese abzulesen und in eine neue Excel-Datei zu schreiben wurde die **JExcelApi**-Bibliothek[111] benutzt.

5.1.2 Statische Daten

Statische Daten stellen den Zusammenbau der Gasturbinen anhand hierarchischer Beziehungen zwischen den Basiskomponenten dar. Der Aufbau der Brennkammern der Turbinen wurde detaillierter mit dazugehörigen Temperatursensoren definiert, da diese für die Anfragen gegen die zu simulierenden Stromdaten relevant sind. Auf der Basis der angepassten Ontologien wurde ein Datengenerator implementiert. Dieser ermöglicht eine automatische Erzeugung von statischen RDF-Daten in RDF/XML-Format für eine beliebige Anzahl von Gasturbinen. Unten ist ein Abschnitt aus der damit erzeugten Datei dargestellt, die Definition entsprechender Namensräume wurde in diesem Beispiel ausgelassen.

Listing 5.2: Beispiel erzeugter statischer RDF-Daten

```
1  <rdf:Description rdf:about="http://www.siemens.com/Optique/turbines/GastTurbine2103_1">
2      <rdf:type rdf:resource="&diag-siem;GastTurbine2103"/>
3  </rdf:Description>
4
5  <rdf:Description rdf:about="http://www.siemens.com/Optique/elements/Burner_1">
6    <rdf:type rdf:resource="&diag-siem;Burner"/>
7    <usecase:isPartOf rdf:resource="http://www.siemens.com/Optique/turbines/
         GastTurbine2103_1">
8  </rdf:Description>
9
10 <rdf:Description rdf:about="http://www.siemens.com/Optique/sensors/TC255_1">
11   <rdf:type rdf:resource="&diag-siem;TC255"/>
12   <usecase:isPartOf rdf:resource="http://www.siemens.com/Optique/elements/Burner_1">
13 </rdf:Description>
```

In diesem Beispiel wird eine Instanz der GasTurbine2103-Klasse deklariert, sowie die dazugehörige Brennkammer mit einem Temperatursensor vom Typ TC255. Die

[111] http://sourceforge.net/projects/jexcelapi/

komplette Datei enthält weitere Komponenten der Turbinen, wie ControlUnit, und jeweils sechs Temperatursensoren verschiedener Typen pro Brennkammer.

5.1.3 Stromdaten

Im Anwendungsfall wurden zwei Typen von Datenströmen verwendet:

- Temperaturmessungen, die von verschiedenen Temperatursensoren der Brennkammer produziert werden, wobei die Messungen über einen gemeinsamen oder verschiedene Datenströme geliefert werden können. Diese werden mit eine Rate von 10 Messungen/Minute geströmt.
- An der Turbine auftretende Ereignisse mit einer mittleren Rate von 1 Ereignis/Minute

Die Struktur der zu simulierenden Datenströme ist aus 5.3 erkennbar. Diese beinhalten pro Messung bzw. Ereignis drei Tripel. Der erste definiert eine Messung bzw. ein Ereignis als eine Instanz einer bestimmten Klasse. Instanzen werden dabei immer nach folgendem Muster benannt <Klassenname>_yyyy_MM_dd_HH_mm_ss. Die beiden weiteren Tripel definieren ihren Wert und das erzeugende Systemelement.

Listing 5.3: Beispiel erzeugter Stromdaten

```
1  http://www.siemens.com/Optique/measurements/TC255-
       Measurement_2013_01_22_01_57_46
2      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
3          http://www.siemens.com/Optique/Diagnosis#TC255-Measurement
4
5  http://www.siemens.com/Optique/measurements/TC255-
       Measurement_2013_01_22_01_57_46
6      http://www.siemens.com/Optique/UseCase#hasNumericalValue
7          78.099^^http://www.w3.org/2001/XMLSchema#double
8
9  http://www.siemens.com/Optique/measurements/TC255-
       Measurement_2013_01_22_01_57_46
10     http://www.siemens.com/Optique/UseCase#element
11         http://www.siemens.com/Optique/sensors/TC255_1
12
13
14
15 http://www.siemens.com/Optique/events/event_2013_01_29_00_28_08
16     http://www.w3.org/1999/02/22-rdf-syntax-ns#type
17         http://www.siemens.com/Optique/Diagnosis#Events
18
19 http://www.siemens.com/Optique/events/event_2013_01_29_00_28_08
20     http://www.siemens.com/Optique/UseCase#hasStringValue
21         Start Initiated^^http://www.w3.org/2001/XMLSchema#string
```

```
22
23 http://www.siemens.com/Optique/events/event_2013_01_29_00_28_08
24     http://www.siemens.com/Optique/UseCase#element
25         http://www.siemens.com/Optique/turbines/GastTurbine2103_1
```

5.2 Anfragen

Einer der wichtigsten Teile der Testumgebung für einen Anfrageengine ist ein Satz von Test-Anfragen. Im Rahmen dieser Arbeit wurden verschiedene Anfragen zum Testen der Engines bezüglich Funktionalität, Leistungsfähigkeit und Skalierbarkeit erstellt. Die funktionalen Testanfragen basieren auf den Daten der Siemens-Gasturbinen und die generierbaren leistungsorientierten Anfragen auf den Zufallsdaten. Die Beschreibung entsprechender Generatoren ist unter 5.3.1.1 und 5.3.1.2 zu finden.
Damit die Durchführung der Tests verschiedene Aspekte der möglichen Anfragen abdeckt und aussagekräftige Ergebnisse liefert, wurde ein Satz von allgemeinen Kriterien, die eine Anfrage definieren, zusammengefasst. Die Kriterien wurden teilweise aus den für das **LUBM**-Benchmark[112] definierten abgeleitet und für den Einsatz dynamischer Datenquellen angepasst. Der Satz der Testanfragen, funktionalen sowie leistungsorientierten, setzt folgende Kriterien praktisch um:

- **Eingabegröße** ist die Anzahl der Aussagen (Tripel), die zu jedem Ausführungszeitpunkt angefragt werden. Dieses Kriterium ist besonders für Leistungsanfragen relevant und wird im Abschnitt 5.2.2 näher besprochen.
- **Selektivität** bezeichnet den abgeschätzten proportionalen Anteil aller Klasseninstanzen der Anfrageeingabe, die den Anfragekriterien entsprechen[113].
- **Komplexität** ist durch den Einsatz komplexer Operatoren, z. B. UNION oder OPTIONAL, Resultat-Modifikatoren und die Komplexität der Anfrageeingabe bestimmt. Die Komplexität der Eingabedaten ist als Anzahl der eingesetzten Klassen und Eigenschaften zu verstehen[114].
- **Erwartetes Reasoning.** Manche Anfragen können nur dann gültige Resultate liefern, wenn das Reasoning bezüglich Klassen- und Eigenschaftenhierarchie sowie logische Inferenz im getesteten System stattfinden[115].

[112] lubm
[113] [Guo u. a. 2005, S.7]
[114] [Guo u. a. 2005, S.7]
[115] [Guo u. a. 2005, S.7]

5.2.1 Anfragen für funktionale Tests

Die Basissprache für Anfragen gegen semantische Daten ist SPARQL. Auf der Basis des Anwendungsfalls wurde ein Satz von Anfragen erstellt. Dieser ist so aufgebaut, dass alle für die Anfragen gegen Stromdaten relevante Konstrukte der SPARQL-Sprache abgedeckt sind. Diese umfassen zusätzlich zu den vorher definierten allgemeinen Kriterien die Möglichkeiten des Aufbaus der drei generellen Bausteine der Anfragen[116]:

- Graphen-Muster, dies beinhaltet alle Operatoren: AND(als "."dargestellt), FILTER, UNION und OPTIONAL, sowie den Einsatz von Unteranfragen.
- Resultat Modifikation mithilfe von Aggregation, Projektion, Zuweisung und Einsatz von Funktionen
- Verschieden Anfragentypen: ASK, CONSTRUCT und SELECT

Die für die Datenströme weniger relevanten Resultatmodifikatoren wie ORDER BY, OFFSET, LIMIT und REDUCED wurden, sowie der DESCRIBE-Anfragentyp aus dem Grund seiner starken Implementationsabhängigkeit, ausgelassen[117].
Die Anfragen wurden erst in natürlicher Sprache definiert und anschließend in C-SPARQL (diese Definition ist aus Platzgründen im Appendix 8 zu finden). Bei der Erstellung von Anfragen wurde ebenfalls der gleichzeitige Einsatz von statischen und dynamischen Datenquellen, sowie das Testen der Reasoning-Fähigkeiten des Engines berücksichtigt. Der Einsatz verschiedener Anfragen-Bausteine und Eingabedaten ist in der Tabelle 5.1 dargestellt.

Anfrage 1 *Liefere alle Werte der TC255-Messungen und die Systemelemente (Temperatursensoren), die diese produziert haben in einem Zeitintervall von 10 Minuten.*
Einfache Anfrage zum Testen der Basisfunktionalität.

Anfrage 2 *Liefere die Werte der TC255-Messungen und die Systemelemente (Temperatursensoren), die diese produziert haben, in einem Zeitfenster von 10 Minuten 6 Mal pro Sekunde.*
Variante der Anfrage 1 mit frequentem Update der Resultate.

[116] srbench
[117] srbench

Anfrage 3 *Liefere alle 6 Sekunden alle TC255-Messungen, deren Wert größer als 70 ist, und die Systemelemente (Temperatursensoren), die diese produziert haben in einem Zeitintervall von 10 Minuten.*
Einfache Anfrage zum Testen der Basisfunktionalität mit Einsatz des FILTER-Operators.

Anfrage 4 *Liefere alle 6 Sekunden Systemelemente, für die der Mittelwert der TC255-Messungen größer als 70 ist, und den ermittelten Mittelwert. Zeitintervall von 10 Minuten.*
Einfache Anfrage zum Testen der Basisfunktionalität mit Einsatz der Aggregation.

Anfrage 5 *Liefere alle Systemelemente, die Messungen mit numerischen Werten produzieren, mit der produzierten Messung und dem Wert dieser in einem Zeitfenster von 30 Minuten.*
Etwas komplexere als die erste Anfrage gegen gleichzeitig zwei Ströme unterschiedlicher Struktur.

Anfrage 6 *Liefere alle Turbinen und Messungen die zu einem Fehler geführt haben. Der Fehler ist als eine Senkung um 5 Prozent der TC255-Messung im Vergleich zum Mittelwert in letzten 3 Stunden bei gleichzeitigem 2-fachen Anstieg des Wertes der TC256-Messung, die von den Temperatursensoren der Brennkammer der Turbine ermittelt wurden.*
Stark selektive Anfrage gegen statische und dynamische Daten unter Einsatz von FILTER-Modifikator und mathematischen Funktionen.

Anfrage 7 *Ist eine Variante der Anfrage 5, die anstatt einen heterogenen Datenstrom gegen mehrere homogene[118] Datenströme ausgeführt wird und zusätzlich optional vorhandene Ereignisse der Fehler produzierenden Turbine liefert.*
Eingabe: zwei homogene Datenströme der Messungen vom Typ TC255 und TC256, Datenstrom der Ereignisse und statischer Datensatz. Die Komplexität dieser Anfrage besteht im Einsatz des OPTIONAL-Operators und gleichzeitigem Einsatz zweier Datenströme und eines statischen Datensatzes.

[118] Unter einem homogenen bzw. heterogenen Datenstrom verstehen wir einen Datenstrom, der Messungen nur eines bestimmten bzw. mehreren Typen beinhaltet.

Anfrage 8 *Liefere die Mittelwerte der Messungen TC255 und TC266 an allen Komponenten innerhalb eines Zeitintervalls von 30 Minuten einmal in 1 Minute.*
Eine Anfrage mittlerer Komplexität zum Testen der Zuweisungsfunktionalität innerhalb einer SELECT-Anfrage.

Anfrage 9 *Generiere einen neuen Strom mit Systemelementen, die ein Symptom des Typs Symptom_1 haben und zum Auftreten des Symptoms führenden Messungen.*
Eine CONSTRUCT-Anfrage mit einer SELECT-Unteranfrage testet die Möglichkeit des Engines neues dynamisches Wissen aus den vorhandenen statischen und dynamischen Daten zu generieren.

Anfrage 10 *Frag einmal in 5 Minuten ab, ob es ein Systemelement gibt, das für eine TC255-Messung einen Wert größer als 5 (Symptom_1 aus der Diagnosis.owl) innerhalb eines Zeitintervalls von 30 Minuten produziert hat.*
Eine einfache ASK-Anfrage gegen einen Datenstrom.

Anfrage 11 *Liefere alle Turbinen für die eine oder beide der folgenden Bedingungen gelten:*

- *Wert der TC255-Messung ist unter der Grenze 3/4 des Mittelwertes der letzten 3 Stunden gefallen*
- *Wert der TC256-Messung ist größer als 700*

Eine komplexe SELECT-Anfrage mit Einsatz des UNION-Operators und Modifikation des Resultats.

Anfrage 12 *Liefere alle Messungen eines bestimmten Typs und die Systemelemente, die diese produziert haben, unter der Bedingung, dass die Turbine, zu der das Systemelement gehört, in dem gleichen Zeitintervall von einer Minute keine Events produzierte.*
Stark selektive Anfrage gegen mehrere Datenströme und statische Daten unter Einsatz der FILTER- und NOT EXISTS-Operatoren.

Die Anfragen 13-15 wurden zum Testen der Möglichkeit, implizites Wissen mithilfe von Reasoning bezugnehmend auf Klassen- und Eigenschaften-Beziehungen abzuleiten konzipiert. Die gleichen Resultate können teilweise mittels verschiedener zusätzlicher Graph-Muster oder ***Property Path****-Ausdrücke von SPARQL 1.1*[119] *erreicht werden.*

Anfrage 13 *Ist eine Variante der Anfrage 5 mit Reasoning bezugnehmend auf die Transitivität der isPartOf-Eigenschaft.*
Es wird nicht explizit auf die Struktur der Turbine eingegangen (Sensor - isPartOf - Burner - isPartOf - GasTurbine), sondern mittels Inferenz Beziehung zwischen dem Sensor und der entsprechenden Turbine abgeleitet.

Anfrage 14 *Liefere alle Features, die numerische Werte produzieren und den Wert dieser im Zeitfenster von 30 Minuten.*
Da der Strom nur die Instanzen verschiedener Messungsklassen beinhaltet, bedarf es des Reasonings bezugnehmend auf subClassOf-Beziehung aller Messungstypen zur Feature-Klasse.

Anfrage 15 *Liefere alle Brennkammern(CombustionChamber-Instanzen) deren Sensoren aktiv Messungen produzieren in einem Zeitfenster von 30 Minuten.*
Die erfolgreiche Ausführung bedarf des Reasonings bezugnehmend auf owl:equivalentClass-Beziehung zwischen Burner und CombustionChamber-Klassen.

Anfrage 16 *Liefere die Zustände aller Systemelemente, die BurnerTipThermocouple-Messungen produzieren. Der Zustand hängt folgendermaßen von dem Wert der Messung ab:*

- *?value<5 => ?state=Offline*
- *5<?value<700 => ?state=Online*
- *?value>700 => ?state=Overheat alarm*

Die erfolgreiche Ausführung bedarf des Reasonings bezugnehmend auf owl:equivalentClass-Beziehung zwischen Burner und CombustionChamber-Klassen.

[119] sparql

	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**	**Q11**	**Q12**	**Q13**	**Q14**	**Q15**	**Q16**
1.Graphenmuster	A	A	A,F	A	A	A	A,O	A	A,F	A,F	A,U	A,F	A	A	A	A
2.Modifikatoren	P	P	P	P,D	P,D	P,D	P,D	P,D	P		P	P	P,D	P,D	P,D	P
3.Anfragetyp	S	S	S	S	S	S	S	S	S,C	A	S	S	S	S	S	S
4.SPARQL 1.1				A,F,Z		A,F	A,F	F,Z	U,Z		F	N	A,F			I
5.Reasoning													T	S	C	
6.Window		S	S	S						S						
7.Input	D	D	D	D	2D	S,D	S,3D	S,D	2S,D	D	S,D	S,2D	S,D	D	D	S,D

Tabelle 5.1: In funktionalen Anfragen eingesetzte Merkmale.

Jedes Merkmal wird mit jeweils einem Buchstaben abgekürzt in der Tabelle vorgestellt. Die Struktur[120] der Darstellung wurde aus einer ähnlichen Tabelle der SR-Bench übernommen und angepasst. Entsprechend der Gruppierung in der Tabelle 5.1 sind folgende Abkürzungen anzunehmen:

- **1.** **A**nd, **F**ilter, **U**nion, **O**ptional
- **2.** **P**rojektion, **D**ISTINCT
- **3.** **S**elect, **A**sk, **C**onstruct
- **4.** **A**ggregation, **F**unktionen, **U**nteranfrage, **N**egation, **I**F, **Z**uweisung, Ausdrücke in **S**elect
- **5.** subClassOf, owl:equivalent**C**lass, **T**ransitivität der Eigenschaft
- **6.** **G**leitendes Zeitfenster
- **7.** **S**tatischer Datensatz, **D**atenströme (gegebenenfalls wird die Anzahl der Datenquellen angegeben)

5.2.2 Anfragen für Skalierbarkeits- und Leistungstests

Da dynamische Umgebungen, wie eine Gasturbine, große Mengen an Daten generieren, ist die Fähigkeit, diese Mengen korrekt und zeitnah zu verarbeiten, eine wichtige Charakteristik eines Anfrageengines. Das erste wichtige Kriterium ist die Eingabegröße, die wir bereits als Anzahl der angefragten Aussagen definiert haben. Für Anfragen gegen mehrere Datenströme bezeichnet die Eingabegröße somit die Menge der Daten, die sich in dem für jeden Datenstrom definierten Zeitfenster zum Ausführungszeitpunkt befinden. Die Menge dieser Daten hängt von der Anzahl der

[120] [Zhang u. a. 2012, S.7]

Ströme, der Struktur dieser und der Raten ab, mit denen die Daten erzeugt werden. Die genaue Anzahl der zu verarbeitenden Tripeln pro Anfrage gegen mehrere Datenströme kann wie folgt berechnet werden:

$$\sum_{i=1}^{n} \frac{T_i \cdot S_i}{R_i}$$

Dabei bezeichnen T_i die Anzahl der Tripel, die zu jedem Zeitpunkt generiert werden, S_i die Größe des Zeitfensters und R_i die Generierungsrate des entsprechenden Datenstroms. Da im Vordergrund die gleichzeitige Verarbeitung einer bestimmter Menge an Daten steht, haben wir die gleichen Parameter für jeden Strom angenommen. Somit hat sich die Formel in folgende Form umgewandelt, wobei N die Anzahl der Ströme bezeichnet.

$$\frac{N \cdot T \cdot S}{R}$$

Weiter ist bei der Verarbeitung die Selektivität der Anfrage von Bedeutung. Diese bestimmt die Menge der Daten, die nach dem Matching der Eingangsdaten und des Graph-Muster durch Modifikation und Projektion ausgegeben wird.
Die Anfragen für Skalierungstests werden für verschiedene Permutationen der obengenannten Kriterien von dem in 5.3.1.1 beschriebenen Generator erstellt. In 5.4 ist die Struktur der generierten Anfrage für C-SPARQL gegen drei Datenströme mit dem Zeitfenster von 10 Minuten und einer mittleren Selektivität zu sehen.

Listing 5.4: Beispiel einer generierten Anfrage für C-SPARQL

```
1  SELECT ?o1 ?o2 ?o3
2
3  FROM STREAM <http://www.test.org/stream1> [RANGE 10m TUMBLING]
4  FROM STREAM <http://www.test.org/stream2> [RANGE 10m TUMBLING]
5  FROM STREAM <http://www.test.org/stream3> [RANGE 10m TUMBLING]
6
7  WHERE {
8    <http://www.test.org/stream1/S> ?p1 ?o1 .
9    <http://www.test.org/stream2/S> ?p1 ?o2 .
10   <http://www.test.org/stream3/S> ?p1 ?o3 .
11   FILTER ( ?o1>AVG(?o1) && ?o2>AVG(?o2) && ?o3>AVG(?o3) )
12   }
```

Für die Anfragen benötigte Tripel werden mit dem Tripelgenerator aus 5.3.1.1 erstellt und können in Form eines C-SPARQL-RDFStream an das Engine mit unterschiedlichen Raten weitergegeben werden. Dies ist im Detail in 5.3.3.2 beschrieben.

Für die Ausführung der Anfragen ist auch eine Registrierung entsprechender Anzahl der Datenströme bei dem Engine notwendig. Somit bilden diese Anfragen zusammen

mit der Möglichkeit, Daten mit unterschiedlichen Raten zu strömen, eine Basis für das Testen von Leistungsmerkmalen, z. B. Throughput. Skalierbarkeit kann dabei abhängig von der Anzahl der Ströme in einer Anfrage oder der Anzahl gleichzeitig angemeldeter Anfragen überprüft werden.

5.3 Implementierung

In diesem Abschnitt werden die Entscheidungsfaktoren bei der Entwicklung der Bausteine der Testumgebung, die die Ausführung der oben beschriebenen Testszenarien ermöglichen, zusammengefasst. Folgend werden die Details der auf diesen Faktoren basierenden Implementierung der Bausteine vorgestellt.

5.3.1 Tripelerzeugung

Die Daten werden an die Engines in Form von RDF-Tripeln übergeben. Dafür werden diese beinhaltende RDF-Ströme bei dem entsprechenden Engine angemeldet. Innerhalb der Testumgebung wurde ein Paket von Java-Klassen entwickelt, das die Funktionalität zur Generierung verschiedener RDF-Tripel für funktionale und Leistungstests bereitstellt. Die Basisklasse des Pakets ist die Triple-Klasse, deren Instanzen von zwei implementierten Tripelgeneratoren zurückgegeben werden.

5.3.1.1 Zufallsdaten

Für die generischen, nicht an den Anwendungsfall gebundenen, Leistungstests wurde ein Tripelgenerator implementiert, der verschiedene Strukturen der Datenströme berücksichtigt und ermöglicht diese zu simulieren. Die Tripel in einem RDF-Strom bestehen aus dem Subjekt, Prädikat und Objekt. Somit ist die Struktur des Stroms von diesen definiert. Folgende Kriterien für die Bausteine des Tripels haben sich ergeben:

- Heterogenität oder Homogenität der Subjekte (Subjekte aller Tripeln sind gleich bzw. unterschiedlich)
- Heterogenität oder Homogenität der Prädikate (Prädikate aller Tripeln sind gleich bzw. unterschiedlich)
- Objekte aller Tripel sind getypte oder ungetypte Literale
- Heterogenität oder Homogenität der Objekte für den Fall, dass diese ungetypte Literale darstellen

- Mittelwert und Abweichung für zufällige numerisch getypte Literale

Permutation dieser Kriterien unter Ausschluss sinnloser Varianten hat acht verschiedene Typen von Tripel ergeben. Anhand dieser wurden entsprechende Erweiterungen der Triple-Klasse implementiert. Für die RandomTripleGenerator-Klasse sind mehrere Konstruktoren definiert, die abhängig von Eingangsparametern, wie Anzahl der generierenden Tripel oder Basis-URI der Subjekte, Tripel des entsprechenden Typs generieren.

5.3.1.2 Gasturbinendaten

Im Rahmen dieser Arbeit wurden die Daten der Temperaturmessungen und Ereignisse in einer Excel-Datei in der ersten bzw. zweiten Arbeitsmappe abgelegt. Jede Reihe der Arbeitsmappe stellt einen bestimmten Zeitpunkt dar. Für die SiemensTripleGenerator-Klasse wurden Methoden zur Umwandlung der Daten für einen bestimmten Zeitpunkt aus der Excel-Arbeitsmappe in entsprechende RDF-Tripeln definiert. Die entstehende Datenstruktur wurde bereits in 5.1.3 vorgestellt.

Die Modularität der Generatoren-Implementierung äußert sich in Unabhängigkeit von verschiedenen Implementierung der RDF-Ströme in Anfrageengines. Dies bietet die Möglichkeit diese zum Testen von allen auch später entwickelten Engines einzusetzen. Des Weiteren bietet die Implementierung des Zufallsdaten-Generators die Möglichkeit verschiedene Typen der Tripel in einem RDF-Strom zusammenzufügen und somit eine große Anzahl an Strukturen zu simulieren.

5.3.2 Anfragengenerator

Für die Durchführung von Leistungstests, besonders Skalierbarkeitstests in Bezug auf die Anzahl der Datenströme die angefragt werden, wurde ein Anfragengenerator in Java-Programmiersprache implementiert. Die Implementierung basiert auf den in 5.2.2 definierten Kriterien. Die Kriterien wurden folgendermaßen umgesetzt:

- Es wurden vier verschiedene Selektivitätsstufen definiert. Die Stufen fassen verschiedene Projektions-Muster innerhalb der SELECT-Klausel sowie den Einsatz von Operatoren und Aggregation-Funktionen, wie AVG und FILTER, zusammen.
- Eine beliebige Anzahl der Ströme kann festgelegt werden
- Die Größe des Zeitfensters für die Ströme kann festgelegt werden

Außer der definierten Permutation verschiedener Kriterien kann die Basis-URI der verwendeten Ströme, sowie die Anfragensprache, für die die Anfrage generiert wird, definiert werden. Momentan ist nur eine Generierung von C-SPARQL-Anfragen möglich, da Tests gegen andere Engines wegen fehlender Dokumentation nicht möglich sind[121]. Der Anfragengenerator kann aber mit geringem Aufwand auf weitere Sprachen erweitert werden. Die Struktur der Anfragen wurde bereits in 5.4 vorgestellt.

5.3.3 Webanwendung

Für die Darstellung der Anfragenresultate wurde eine Webanwendung mit Flex als Frontend-Technologie und einem BlazeDS-Java-Backend[122] implementiert. Diese Technologien wurden aus dem Grund ausgewählt, dass die Messaging-Komponente von BlazeDS zusammen mit Adobe Flex eine Implementierung einer auf Stromdaten basierten Anwendung mit geringem Aufwand[123] ermöglicht. Zusätzlich werden die Daten vom Messaging-Dienst als binärer Datenstrom im **Action-Message-Format**[124] übetragen. AMF wurde von Adobe als offenes Protokoll für das Serialisieren bzw. Deserialisieren von komplexen Objekten zwischen Klient und Serverseite entwickelt und bietet viele Leistungsvorteile[125][126]. Diese kommen besonders bei Anfragen mit einem erwarteten umfangreichen Resultset, dass mit hohen Raten erzeugt wird, zur Geltung. Bei der Entwicklung wurden folgende Tools und Technologien eingesetzt:

- **Eclipse IDE for Java Developers Ingigo Package** als Entwicklungsumgebung
- **Flex Builder Plugin for Eclipse 4.6** für die Entwicklung von Flex-Anwendungen in Eclipse
- **Adobe Flex 4.6 SDK**[127]
- **BlazeDS 4.0 turnkey**[128] beinhaltet zusätzlich einen integrierten Tomcat-Server
- **CSPARQL-ReadyToGoPack-0.7.4**, das einen Teil der Klassen, die den Einsatz des C-SPARQL-Engines erlauben, beinhaltet.
- **CQELS** jar-Paket mit allen nötigen Klassen für den Einsatz des Engines

[121] 5.3.3.3
[122] blazeds
[123] gsblazeds
[124] [Widjaja 2010, S.137]
[125] [Khristianov 2012, S. 38]
[126] census
[127] flexsdk
[128] blazeds

Zusätzlich wurde der Tomcat-Server in der Eclipse-Entwicklungsumgebung nach einer Anleitung[129] eingestellt.

5.3.3.1 BlazeDS-Backend

Das Backend wurde als ein auf dem Tomcat-Server laufender Dienst entwickelt. Die Implementierung erfolgte auf der Basis von zwei Anleitungen[130][131] und umfasst eine Erweiterung der ServiceAdapter-Klasse sowie eine Temperature-Klasse, die Temperatur-Messungen darstellt. Die Instanzen der letzten Klasse werden an das Frontend geschickt.
Die Adapter-Klasse integriert das CPARQL-Engine und sendet die Anfrageresultate an die Clients nach der Push-Technologie[132]. Des Weiteren kommen die schon vorgestellten Tripel-Generatoren 5.3.1.2 und 5.3.1.1 sowie in 5.3.3.2 beschriebene Bausteine des CSPARQL-Engines zum Einsatz.
Das Verhalten des BlazeDS-Backends wurde in entsprechenden Konfigurationsdateien definiert. Es wurde die Csparql-Destination deklariert und mit dem oben beschriebenen Adapter verknüpft. Diese Destination benutzt den zugewiesenen Adapter zur Weiterleitung von Nachrichten und kann als Instanz des vom Adapter bereitgestellten Dienstes angesehen werden[133]. Bei dieser Destination können sich die Clients anmelden, die die weitergeleiteten Nachrichten empfangen wollen. Für den Adapter wurde ebenfalls das Verhalten des Standard-AMF-Kanals namens my-streaming-amf definiert. Die beschriebenen Definitionen befinden sich in den Dateien messaging-config.xml und service-config.xml auf der dieser Arbeit beiliegenden Disk mit dem Code der implementierten Webanwendung.

5.3.3.2 C-SPARQL

C-SPARQL-Engine wird in Form eines oben genannten Pakets zu Verfügung gestellt. Folgende Liste beinhaltet die wichtigsten Klassen des Pakets und deren Zusammenhänge:

- CsparqlEngine ist die Basisklasse. Instanz dieser ist die Engine selbst. Die Klasse stellt Methoden zum Anmelden der RDF-Datenströme und Anfragen bereit.
- RdfQuadruple beschreibt eine RDF-Tripel mit dem entsprechenden Zeitstempel.

[129] setser
[130] gsblazeds
[131] cusser
[132] `http://en.wikipedia.org/wiki/Push_technology`
[133] [Pisa 2009, S.304]

- RDFStream beschreibt einen RDF-Datenstrom. Die Klasse wird über die put-Methode dem Strom RDF-Quadrupeln hinzugefügt.
- CsparqlQueryResultProxy stellt Funktionalität zur Verwaltung des Anfragenresultats bereit.
- ResultFormatter dient zur Formatierung des Anfragen-Resultats. Instanz dieser Klasse wird über die addObserver-Methode an die Instanz der CsparqlQueryResultProxy-Klasse übergeben.
- CsparqlQueryResultProxy stellt Funktionalität zur Verwaltung des Anfragenresultats bereit.

RDF-Stromgenerator Im Rahmen der Arbeit wurden zwei RDF-Stromgeneratoren implementiert. Diese wurden als Erweiterungen der RDFStream-Klasse definiert, die gleichzeitig die Runnable-Schnittstelle implementieren. Diese werden dann als Java-Threads gestartet werden.

- Der TurbineStreamer-Generator stellt die vom Tripelgenerator für Siemens Daten 5.3.1.2 gelieferten RDF-Daten in Form eines RDF-Stroms bereit. Die Default-Generierungsrate des Generators ist auf zehn Messungen pro Sekunde gelegt, was der im Anwendungsfall 5.1 festgelegten Rate der Temperaturmessungen entspricht. Der Typ des gelieferten Stroms kann auf Messungsstrom bzw. Ereignisstrom gesetzt werden.
- Der GenericStreamGenerator-Generator stellt die vom generischen Tripelgenerator gelieferten RDF-Daten in Form eines RDF-Stroms bereit.

Resultatformatierung CSPARQL sieht eine Formatierung des Anfrageresultats vor. Eine MeasurementValueFormatter-Klasse wurde als Erweiterung der abstrakten ResultFormatter-Klasse implementiert. Die update-Methode der abstrakten Oberklasse wurde überschrieben, um die Werte der Temperaturmessungen an den BlazeDS-Service 5.3.3.1 weiterzuleiten.

5.3.3.3 CQELS

Für die Implementation auf der Basis des CQELS-Engines wurde das letzte bereitgestellte jar-Paket (von August 2011) benutzt. Die Implementation umfasste zwei Klassenimplementierungen: einer Engine-Klasse und eines Stromgenerators. Diese

basierte auf dem Beispiel auf der Seite des CQELS-Projekts[134] und erlaubte das Ausführen einiger simplen Anfragen. Allerdings ist das vorgestellte Beispiel recht simpel und es existiert keine weitere Dokumentation. Die versuchte Kommunikation mit dem Entwickler brachte keine weiteren Erkenntnisse. Aus diesen Gründen fand eine weitere Integration des CQELS-Engines in die Webanwendung nicht statt.

5.3.3.4 Flex-Frontend

Das Frontend der Webanwendung stellt einen Navigator dar, der zwischen Darstellungen in Form eines Diagramms oder einer Tabelle umschalten lässt. Dieses wurde in MXML-Sprache implementiert in einer einzigen Datei. Das Hauptelement der Anwendung wird als Client des BlazeDS Messaging-Dienstes angemeldet indem einer Instanz der Consumer-Klasse die auf der Serverseite definierte Destination zugewiesen wird. In diesem Fall die auf dem BlazeDS-Backend deklarierte Csparql-Destination.
Nach Empfang einer Nachricht wird die messageHandler-Methode ausgeführt und die Temperaturdaten in einem ArrayCollection gespeichert. Dieses dient dann als gleichzeitiger Datenprovider für die Diagramm- und Tabellen-Komponenten. Da bei jeder Veränderung des Resultats der Anfrage das Resultat an die Csparql-Destination als Nachricht verschickt wird, wird der Datenprovider ebenfalls aktualisiert und somit auch die Darstellung der Daten innerhalb der Anwendung.

[134] cqels

6 Ergebnisse und Auswertung

In diesem Kapitel werden die Ergebnisse der Auswertung vorgestellt, die im Rahmen dieser Arbeit durchgeführt wurde. Dieser konnte nur CSPARQL unterzogen werden. Die Auswertung anderer Engines ist wegen der fehlenden Dokumentation oder öffentlich zugänglicher Implementierung nicht möglich gewesen.
Das implementierte Benchmark wurde auf dem folgenden System ausgeführt unter Einsatz der CSPARQL-Implementierung aus dem CSPARQL-ReadyToGoPack-0.7.4[135] und den CSPARQL-Formulierungen der Anfragen aus dem in 5.2.1 definierten Satz (die Formulierungen sind dem Appendix 8.1 zu entnehmen):

Betriebssystem	Windows 7 Ultimate 32-Bit Service Pack 1[136]
Prozessor	Intel®Core™I3 M370 @2,40Ghz
Arbeitsspeicher	2,92 GB

Tabelle 6.1: Ausführungsumgebung für das Benchmark

6.1 Funktionalität

Aus sechzehn Testanfragen für funktionale Tests konnten mit CSPARQL sechs erfolgreich ausgeführt werden. Der häufigste Grund für die gescheiterte Ausführung war ein Bug in der Implementierung des Engines. Dies hatte zur Folge, dass für alle Anfragen die gegen mehrere Datenquellen ausgeführt wurden, ein leeres Resultat produziert wurde. Ein anderer Grund für den Wegfall der Anfragen waren die nicht vorhandene Implementierung der mit SPARQL 1.1 eingeführten Operatoren, wie IF und NOT EXISTS.
Die sechs ausführbaren Anfragen (Nummern 1,2,3,4,8,9) haben gezeigt, das CSPARQL die wesentlichen Anfragemuster unterschiedlicher Komplexität unterstützt. Mit CSPARQL konnten SELECT und CONSTRUCT Anfragentypen ausgeführt und somit auch neue Datenströme erzeugt werden, allerdings fehlt die Implementation der ASK-Anfragen. Die Umgebung, die Anfragen gegen erzeugte Datenströme erlaubt,

[135] csparql
[136] http://windows.microsoft.com

befindet sich laut Aussage des Entwicklers in Entwicklung. CSPARQL unterstützte folgende Konstrukte: Projektion, Zuweisung, Ausdrücke und Funktionen innerhalb der SELECT-Klausel, Aggregation, Modifikation des Resultats und Einsatz der UNION und AND Mengenoperatoren in Graph-Muster.
Obwohl laut anderen Benchmarks[137] und der Entwicklerseite[138] CSPARQL als einziges der aktuellen strRS Engines einfaches Reasoning auf Basis der RDF-Folgebeziehungen unterstützt, konnte dieses auf Grund des vorhandenen Bugs nicht getestet werden.

6.2 Leistungsfähigkeit und Skalierbarkeit

Aus dem Grund, das CSPARQL sowie andere Engines sich noch in der Entwicklungsphase befindet und den vorhandenen Bugs, sind ausführliche Tests des Engines bezüglich Leistungsfähigkeit nicht möglich. Für das Testen dieser wären folgende Kriterien wichtig:

- Große des statischen Datensatzes
- Anzahl von angefragten Datenströmen
- Komplexität der Anfragen
- Anzahl an gleichzeitig ausführenden Anfragen
- Hohe Datenankunftsraten

Trotz fehlender Möglichkeit mehrere Datenquellen einzusetzen wurden im Rahmen dieser Arbeit die ersten Skalierungs- und Leistungstests durchgeführt. Aus dem gleichen Grund konnten die generierbaren Anfragen aus 5.2.2 nicht benutzt werden. Die Tests basierten auf vier Anfragen des festen Anfragensatzes (Nummern 1,2,4,8) unterschiedlicher Komplexität. Dabei wurde der Einfluss verschiedener Aspekte auf die Ausführungszeit der Anfragen in folgenden Szenarien getestet:

- Anfragen wurden gegen Datenströme mit verschiedenen Raten und unveränderter Größe des Zeitfensters ausgeführt. Damit haben sich folgende drei Größen der sich im Zeitfenster befindenden Daten ergeben: 65, 130 und 260 Tausend Tupeln mit den Ankunftsraten von 6,12 und 24 Tupel pro Sekunde.
- Es wurde die gleiche Anfrage mehrmals beim Engine bei konstanter Rate der ankommenden Daten von 6 Tupel pro Sekunde angemeldet. Jede der vier Testanfragen wurde jeweils 1, 10, 100 und 300 Mal gleichzeitig ausgeführt.

[137] [Zhang u. a. 2012, S. 13]
[138] csparql

Die Resultate der Tests sind in 6.1 und 6.2 vorgestellt und stellen den Mittelwert der Ausführungszeit der Anfragen in Abhängigkeit von den angesprochenen Kriterien dar. Da das über den Datenstrom innerhalb der Anfragen definierte Zeitfenster 10 Minuten beträgt und beim Start der Umgebung die Daten erst in das System geströmt werden müssen, fanden die Messung erst statt, nachdem dem Zeitfenster entsprechende Daten im System verfügbar waren.

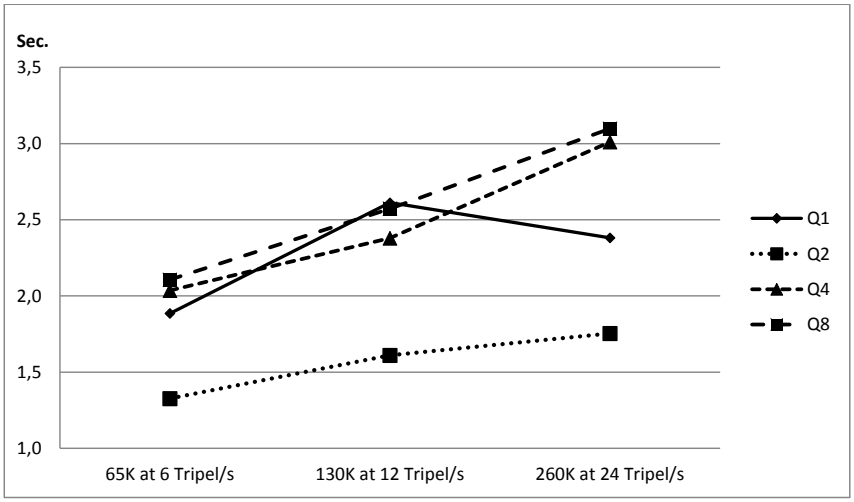

Abbildung 6.1: Mittlere Ausführungszeit bei unterschiedlicher Datenankunftsrate

Im ersten Szenario wurden jeweils die letzten fünf Ausführungszeiten der Anfragen gemessen und ein Mittelwert daraus gebildet. Die Erhöhung der Ankunftsrate hatte als Folge erwartungsgemäß auch eine Erhöhung der Ausführungszeit, diese zeigte aber keine drastische Einbuße. Bei der Erhöhung der Ankunftsrate und somit der Menge der angefragten Daten um den Faktor 4 erhöhte sich die Ausführungszeit abhängig von der Komplexität der Anfrage um den Faktor von 1,26 bis 1,5.

Für das zweite Szenario wurde der Mittelwert der fünf letzten Ausführungszeiten aller gleichzeitig ausgeführten Anfragen gebildet, z. B. bei 100 angemeldeten Anfragen wurden für jede Anfrage 5 Messungen vorgenommen und somit insgesamt 500 Ausführungszeiten gemessen. Die gleichzeitig mehrfache Ausführung der gleichen Anfrage hatte sogar eine Senkung der mittleren Ausführungszeiten zur Folge. Eine mögliche Erklärung dafür könnte eine in CSPARQL implementierte Optimierung bei Ausführung von Anfragen sein, die den gleichen Graph-Muster besitzen[139].
Ein weiteres Ergebnis der beiden Szenarien war, dass die Anfrage 1 trotz kleinster Komplexität im Vergleich zu anderen eine unerwartet hohe Ausführungszeit hatte. Was diese Anfrage von anderen unterscheidet ist, dass für diese kein gleitendes son-

[139] [Golab und Zsu 2010, S. 6]

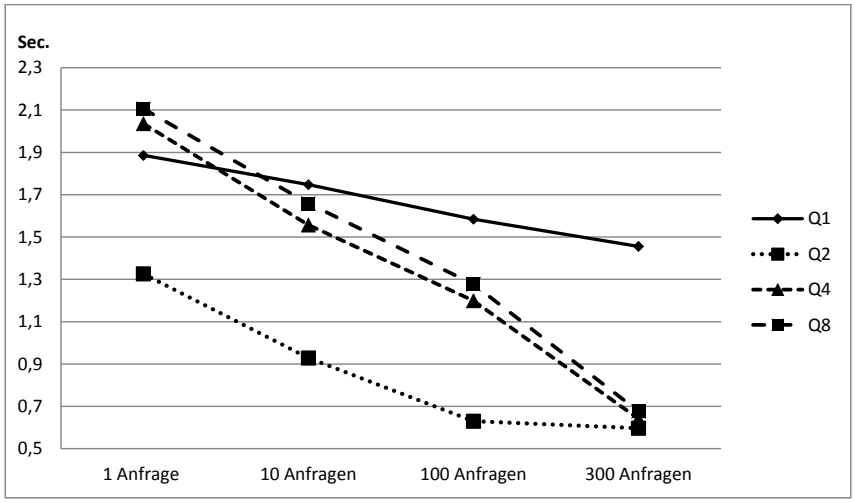

Abbildung 6.2: Mittlere Ausführungszeit bei gleichzeitiger Ausführung mehrerer Anfragen

dern ein festes Zeitfenster über den Datenstrom definiert wurde. Als Folge dessen erfolgt die Ausführung immer über eine Menge der Daten, die sich bei letzter Ausführung im Zeitfenster noch nicht befanden. Das deutet auf eine Optimierung der Ausführung von Anfragen gegen Daten, die sich nur teilweise verändert haben.
Da im Rahmen dieser Arbeit kein Vergleich verschiedener Engines bezüglich Leistungsfähigkeit möglich ist, haben wir die Ausführungszeiten der Anfragen der in Le-Phuoc u. a. [2012b] durchgeführten Auswertung in Betracht gezogen. Daraus und aus unserer Auswertung ist ersichtlich, dass CSPARQL auch bei Ausführung simpler Anfragen deutlich hohe Ausführungszeiten vorweist, z. B. war in dieser Auswertung eine Ausführungszeit für eine simple Anfrage für CQELS in Höhe von 0,47 Millisekunden angegeben, wobei die gleiche Anfrage von CSPARQL in 332,46 Millisekunden verarbeitet wurde.

7 Fazit und Ausblick

Im Rahmen dieser Arbeit wurden wichtige Aspekte der Verarbeitung von Datenströmen und besonders RDF-Datenströme untersucht. Diese Untersuchung ergab relevante Kriterien und Anforderungen an Engines Anfragen gegen semantische Stromdaten. Um eine Auswertung von solchen Engines zu ermöglichen, wurde in Bezug auf diese Kriterien ein Satz von Werkzeugen, die eine Basis für ein derartiges Benchmark bilden, vorgestellt und implementiert. Der Anwendungsfall für das Benchmark ist im neu angelaufenen EU-Projekt Optique[140] beschrieben und basiert auf den Daten der Siemens-Gasturbinen.
Folgende Bausteine des Benchmarks wurden entwickelt und implementiert:

- Struktur der statischen und dynamischen Daten basierend auf den Daten der Siemens Gasturbinen
- RDF-Datengenerator für statische Daten basierend auf dem Anwendungsfall
- RDF-Tripel-Generator für Stromdaten basierend auf dem Anwendungsfall
- Generischer RDF-Tripel-Generator für die Erstellung von Testtripeln verschiedener Struktur
- Ein Satz von Anfragen für funktionale Tests
- Ein auf mehreren Kriterien basierender Anfragen-Generator für Leistungs- und Skalierungstests
- Eine Webanwendung zur Darstellung der Resultate der Anfragen, basierend auf CSPARQL-Engine, Adobe Flex und BlazeDS

Es wurden die vorgeschlagenen Anfragegrammatiken der CQELS, CSPARQL und SPARQLstream Engines untersucht, praktisch getestet werden konnte nur CSPARQL. Beide andere Engines konnten aus dem Grund fehlender Dokumentation und mangelnden Supports durch die Entwickler nicht getestet werden. Für CQELS wurden sehr simple Anfragen ausgeführt allerdings traten beim Einsatz minimal komplexer Anfragen Fehler auf, die aus dem vorher genannten Grund leider nicht analysiert werden konnten.

[140] optique

Die auf dem Satz von Testanfragen basierenden funktionalen Tests zeigten eine Unterstützung grundlegender Anfragemuster, aber mangelnde der SPARQL 1.1 Operatoren. Auf Grund von Bugs lieferten mehrere Anfragen leere Resultate und konnten deswegen nicht ausgewertet werden. Die Fehler entstanden vor allem beim Einsatz mehrerer Datenströme sowie statischer Datensätze.
Die Skalierungstests deuten auf verschiedene Optimierungen bezüglich z. B. gleicher Graph-Muster in Anfragen und Einsatz von Datenquellen mit unterschiedlichen realistischen Datenankunftsraten. Der Vergleich mit anderen Engines bezüglich der Leistungsfähigkeit und Skalierbarkeit fand aus benannten Gründen nicht statt.
Resultate der im Rahmen dieser Arbeit bezüglich des Funktionalitätsumfangs durchgeführter Auswertung am Beispiel von CSPARQL sind diesen des SRBench[141] Benchmarks ähnlich. Die Resultate der Auswertung des CSPARQL-Engines bezüglich Skalierbarkeit im Rahmen des LSBench[142] Benchmarks konnten nicht nachvollzogen werden. Da aber keine Angaben über die eingesetzte Version des Engines vorhanden sind und manche Details der Testausführung unklar sind, kann kein aussagekräftiger Vergleich durchgeführt werden.

7.1 Ausblick

Die Auswertungen im Rahmen dieser Arbeit sowie anderer aktuellen Benchmarks zeigten zu erwartenden Resultate, da alle Systeme für die Verarbeitung von semantischen Datenströmen sich in der Entwicklungsphase befinden. Um diese Entwicklung voranzutreiben, bedarf es einer ständigen Auswertung dieser Systeme. Eine der Voraussetzungen, die dafür erfüllt sein müssen ist die Existenz einsetzbarer Implementierungen und entsprechender Dokumentation. Dieses wird derzeit von nahezu keinem der Systeme erfüllt.
Die Definition der Leistungsmetriken und eine Entwicklung und Implementation der Validierung der Anfrageresultate wären die wichtigen Schritte zur Weiterentwicklung dieses Benchmarks. Die Definition der Metriken kann eventuell von den in LUBM[143] und Linear Road[144] Benchmarks definierten abgeleitet oder angepasst werden. Dies setzt aber eine schon existierende Validierung der Resultate voraus, was angesichts derzeit unterschiedlicher Interpretation gleicher Konstrukte einer weiteren Untersuchung bedarf.

[141] srbench
[142] lsbench
[143] lubm
[144] linroad

8 Appendix

8.1 Anfragenformulierungen in CSPARQL

Listing 8.1: Anfrage 1

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4
5  SELECT ?measurement ?value
6  FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 10m
       TUMBLING]
7  WHERE {
8    ?measurement rdf:type dia-siem:TC255-Measurement ;
9                  usecase:hasNumericalValue ?value
10   }
```

Listing 8.2: Anfrage 2

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4
5  SELECT ?measurement ?value
6  FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 10m STEP 166
       ms]
7  WHERE {
8      ?measurement rdf:type dia-siem:TC255-Measurement ;
9                    usecase:hasNumericalValue ?value
10   }
```

Listing 8.3: Anfrage 3

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4
5  SELECT ?sensor ?measurement
6  FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 10m STEP 6s]
7  WHERE {
8    ?measurement usecase:element ?sensor ;
9            rdf:type  dia-siem:TC255-Measurement ;
```

```
10                  usecase:hasNumericalValue ?value .
11  FILTER (?value>70)
12  }
```

Listing 8.4: Anfrage 4

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4
5  SELECT DISTINCT ?sensor (AVG(?value) as ?avg)
6  FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 10m 6s]
7  WHERE {
8    ?measurement usecase:element ?sensor ;
9           rdf:type  dia-siem:TC255-Measurement ;
10                  usecase:hasNumericalValue ?value
11   }
12   GROUP BY ?sensor
13   HAVING (AVG(?value) > "70"^^xsd:float)
```

Listing 8.5: Anfrage 5

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3
4  SELECT DISTINCT ?element ?measurement ?value
5  FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 30m
       TUMBLING]
6  FROM STREAM <http://www.siemens.com/Optique/events/> [RANGE 30m TUMBLING]
7
8  WHERE {
9    ?measurement usecase:element ?element ;
10          usecase:hasNumericalValue ?value.
11   }
```

Listing 8.6: Anfrage 6

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4
5  SELECT DISTINCT ?turbine ?valueK ?valueT
6  FROM   <http://localhost:8400/TurbineStatic.rdf>
7  FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 3h TUMBLING
       ]
8
9  WHERE {
10   ?measurementK usecase:element ?elementK ;
11                 rdf:type  dia-siem:TC255-Measurement ;
12          usecase:hasNumericalValue ?valueK.
```

```
13   ?measurementT usecase:element ?elementT ;
14                 rdf:type  dia-siem:TC256-Measurement ;
15           usecase:hasNumericalValue ?valueT.
16
17   ?elementK usecase:isPartOf ?syselementK.
18   ?elementT usecase:isPartOf ?syselementT.
19
20   ?syselementK usecase:isPartOf ?turbine.
21   ?syselementT usecase:isPartOf ?turbine.
22   }
23   GROUP BY ?turbine
24   HAVING (?measurementK<0,95*AVG(?measurementK)&&?measurementT>2*AVG(?
         measurementT))
```

Listing 8.7: Anfrage 7

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4
5  SELECT DISTINCT ?turbine ?valueK ?valueT ?event
6  <http://localhost:8400/TurbineStatic.rdf>
7  FROM STREAM <http://www.siemens.com/Optique/measurements/TC255-Measurements> [
       RANGE 3h TUMBLING]
8  FROM STREAM <http://www.siemens.com/Optique/measurements/TC256-Measurements> [
       RANGE 3h TUMBLING]
9  FROM STREAM <http://www.siemens.com/Optique/events/> [RANGE 3h TUMBLING]
10 WHERE {
11   ?measurementK usecase:element ?elementK ;
12                 rdf:type  dia-siem:TC255-Measurement ;
13           usecase:hasNumericalValue ?valueK.
14   ?measurementT usecase:element ?elementT ;
15                 rdf:type  dia-siem:TC256-Measurement ;
16           usecase:hasNumericalValue ?valueT.
17
18   ?elementK usecase:isPartOf ?syselementK.
19   ?elementT usecase:isPartOf ?syselementT.
20
21   ?syselementK usecase:isPartOf ?turbine.
22   ?syselementT usecase:isPartOf ?turbine.
23
24
25   OPTIONAL{
26     ?event usecase:element ?turbine
27   }
28   }
29   GROUP BY ?turbine
30   HAVING (?measurementK<0,95*AVG(?measurementK)&&?measurementT>2*AVG(?
         measurementT))
```

Listing 8.8: Anfrage 8

```
PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT (AVG(?value255) AS ?avg255) (AVG(?value256) AS ?avg256)

FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 30m STEP 1m]
WHERE {
  {?measurement255 rdf:type dia-siem:TC255-Measurement ;
        usecase:hasNumericalValue ?value255.}
  UNION
  {?measurement256 rdf:type dia-siem:TC256-Measurement ;
        usecase:hasNumericalValue ?value256.}
  }
```

Listing 8.9: Anfrage 9

```
PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {?sensor usecase:hasSymptom [a dia-siem:Symptom_1; usecase:hasFeature ?
    measurement] }
FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 30m STEP 5m]
WHERE {
  SELECT DISTINCT ?sensor ?measurement
  WHERE {
    ?measurement usecase:element ?sensor ;
                rdf:type  dia-siem:TC255-Measurement ;
        usecase:hasNumericalValue ?value.
    FILTER (?value>5)
    }
  }
```

Listing 8.10: Anfrage 10

```

PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

ASK
FROM STREAM <http://www.siemens.com/Optique/UseCase/measurements> [RANGE 30m
    STEP 5m]
WHERE {
  ?measurement usecase:element ?sensor ;
                rdf:type  dia-siem:TC255-Measurement ;
        usecase:hasNumericalValue ?value.
```

```
12  FILTER (?value>5)
13  }
```

Listing 8.11: Anfrage 11

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4
5  SELECT ?turbine ?valueK ?valueT
6  FROM   <http://localhost:8400/TurbineStatic.rdf>
7  FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 3h TUMBLING
       ]
8  WHERE {
9    {?measurementK usecase:element ?elementK ;
10                  rdf:type  dia-siem:TC255-Measurement ;
11           usecase:hasNumericalValue ?valueK.
12   ?elementK usecase:isPartOf ?syselementK.
13   ?syselementK usecase:isPartOf ?turbine.
14   FILTER (?measurementK<0,75*AVG(?measurementK))
15   }
16   UNION
17   {
18   ?measurementT usecase:element ?elementT ;
19                  rdf:type  dia-siem:TC256-Measurement ;
20           usecase:hasNumericalValue ?valueT.
21   ?elementT usecase:isPartOf ?syselementT.
22   ?syselementT usecase:isPartOf ?turbine.
23   FILTER (?measurementT>700)
24   }
25   }
```

Listing 8.12: Anfrage 12

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4
5  SELECT ?sensor ?value
6  FROM   <http://localhost:8400/TurbineStatic.rdf>
7  FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 1m
       TUMBLING]
8  FROM STREAM <http://www.siemens.com/Optique/events/> [RANGE 1m TUMBLING]
9
10 WHERE {
11   ?measurement usecase:element ?sensor ;
12           rdf:type  dia-siem:TC255-Measurement ;
13                   usecase:hasNumericalValue ?value
14   ?elementK usecase:isPartOf ?syselementK.
15   ?syselementK usecase:isPartOf ?turbine.
```

```
16
17   FILTER NOT EXISTS { ?event usecase:element ?turbine ;
18                       a dia-siem:Events}
19   }
```

Listing 8.13: Anfrage 13

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4
5  SELECT DISTINCT ?turbine ?valueK ?valueT
6  FROM   <http://localhost:8400/TurbineStatic.rdf>
7  FROM   <http://localhost:8400/optique-pattern.owl>
8  FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 3h TUMBLING
       ]
9  WHERE {
10   ?measurementK usecase:element ?elementK ;
11                 rdf:type  dia-siem:TC255-Measurement ;
12           usecase:hasNumericalValue ?valueK.
13   ?measurementT usecase:element ?elementT ;
14                 rdf:type  dia-siem:TC256-Measurement ;
15           usecase:hasNumericalValue ?valueT.
16
17   ?elementK usecase:isPartOf ?turbine.
18   ?elementT usecase:isPartOf ?turbine.
19   }
20   GROUP BY ?turbine
21   HAVING (?measurementK<0,95*AVG(?measurementK)&&?measurementT>2*AVG(?
         measurementT))
```

Listing 8.14: Anfrage 14

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3
4  SELECT ?measurement ?value
5  FROM   <http://localhost:8400/Diagnosis.owl>
6  FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 30m
       TUMBLING]
7  WHERE {
8    ?measurement a usecase:Feature ;
9           usecase:hasNumericalValue ?value.
10   }
```

Listing 8.15: Anfrage 15

```
1  PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
2  PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
3
```

```
SELECT DISTINCT ?combuster
FROM   <http://localhost:8400/Diagnosis.owl>
FROM   <http://localhost:8400/TurbineStatic.rdf>
FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 30m
     TUMBLING]

WHERE {
  ?sensor    a dia-siem:Sensor ;
         usecase:isPartOf ?combuster .
  ?combuster  a dia-siem:CombustionChamber .
  }
```

Listing 8.16: Anfrage 16

```
PREFIX usecase: <http://www.siemens.com/Optique/UseCase#>
PREFIX dia-siem: <http://www.siemens.com/Optique/Diagnosis#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?sensor (IF(?value < 5, "Offline",
           IF(?value) < 700,  "Online",  "Overheat alarm") as ?state)
FROM   <http://localhost:8400/TurbineStatic.rdf>
FROM STREAM <http://www.siemens.com/Optique/measurements> [RANGE 30m
     TUMBLING]
WHERE {
  ?measurement usecase:element ?sensor ;
             rdf:type  dia-siem:BurnerTipThermocouple ;
                  usecase:hasNumericalValue ?value
  }
```

8.2 Angepasste Ontologien

Listing 8.17: Ontologie der Gasturbinen-Wissensdomäne

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY UseCase "http://www.siemens.com/Optique/usecase#" >
    <!ENTITY dia-siem "http://www.siemens.com/Optique/Diagnosis#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
]>

<rdf:RDF
    xmlns="http://www.siemens.com/Optique/Diagnosis#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
15      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
16      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
17      xmlns:owl="http://www.w3.org/2002/07/owl#"
18      xmlns:UseCase="http://www.siemens.com/Optique/UseCase#"
19      xmlns:j.0="http://www.w3.org/2006/12/owl11#"
20    xml:base="http://www.siemens.com/Optique/Diagnosis">
21
22    <owl:Ontology rdf:about="">
23      <owl:imports rdf:resource="http://www.siemens.com/Optique/UseCase"/>
24    </owl:Ontology>
25
26    <owl:Class rdf:about="#Events">
27      <rdfs:subClassOf rdf:resource="&UseCase;StringFeature"/>
28    </owl:Class>
29    <owl:Class rdf:about="#BearingMeasurements">
30      <rdfs:subClassOf rdf:resource="&UseCase;Feature"/>
31    </owl:Class>
32    <owl:Class rdf:about="#LubricatingOilSystemMeasurements">
33      <rdfs:subClassOf rdf:resource="&UseCase;Feature"/>
34    </owl:Class>
35    <owl:Class rdf:about="FuelAndSpeedMeasurements">
36      <rdfs:subClassOf rdf:resource="&UseCase;Feature"/>
37    </owl:Class>
38    <owl:Class rdf:about="#HotGasPathMeasurements">
39      <rdfs:subClassOf rdf:resource="&UseCase;Feature"/>
40    </owl:Class>
41    <owl:Class rdf:about="#BurnerTipThermocouple">
42      <rdfs:subClassOf rdf:resource="&UseCase;NumericalFeature"/>
43      <rdfs:subClassOf rdf:resource="#HotGasPathMeasurements"/>
44    </owl:Class>
45    <owl:Class rdf:about="#TC255-Measurement">
46      <rdfs:subClassOf rdf:resource="#BurnerTipThermocouple"/>
47      <owl:equivalentClass>
48        <owl:Restriction>
49          <owl:someValuesFrom rdf:resource="#TC255"/>
50          <owl:onProperty rdf:resource="&UseCase;element"/>
51        </owl:Restriction>
52      </owl:equivalentClass>
53    </owl:Class>
54    <owl:Class rdf:about="#TC256-Measurement">
55      <rdfs:subClassOf rdf:resource="#BurnerTipThermocouple"/>
56      <owl:equivalentClass>
57        <owl:Restriction>
58          <owl:someValuesFrom rdf:resource="#TC256"/>
59          <owl:onProperty rdf:resource="&UseCase;element"/>
60        </owl:Restriction>
61      </owl:equivalentClass>
62    </owl:Class>
63    <owl:Class rdf:about="#TC257-Measurement">
```

```
64      <rdfs:subClassOf rdf:resource="#BurnerTipThermocouple"/>
65      <owl:equivalentClass>
66        <owl:Restriction>
67          <owl:someValuesFrom rdf:resource="#TC257"/>
68          <owl:onProperty rdf:resource="&UseCase;element"/>
69        </owl:Restriction>
70      </owl:equivalentClass>
71    </owl:Class>
72    <owl:Class rdf:about="#TC258-Measurement">
73      <rdfs:subClassOf rdf:resource="#BurnerTipThermocouple"/>
74      <owl:equivalentClass>
75      <owl:Restriction>
76        <owl:someValuesFrom rdf:resource="#TC258"/>
77        <owl:onProperty rdf:resource="&UseCase;element"/>
78        </owl:Restriction>
79    </owl:equivalentClass>
80    </owl:Class>
81    <owl:Class rdf:about="#TC259-Measurement">
82      <rdfs:subClassOf rdf:resource="#BurnerTipThermocouple"/>
83      <owl:equivalentClass>
84        <owl:Restriction>
85          <owl:someValuesFrom rdf:resource="#TC259"/>
86          <owl:onProperty rdf:resource="&UseCase;element"/>
87        </owl:Restriction>
88      </owl:equivalentClass>
89    </owl:Class>
90    <owl:Class rdf:about="#TC260-Measurement">
91      <rdfs:subClassOf rdf:resource="#BurnerTipThermocouple"/>
92      <owl:equivalentClass>
93        <owl:Restriction>
94          <owl:someValuesFrom rdf:resource="#TC260"/>
95          <owl:onProperty rdf:resource="&UseCase;element"/>
96        </owl:Restriction>
97      </owl:equivalentClass>
98    </owl:Class>
99    <owl:Class rdf:about="#Turbine">
100     <rdfs:subClassOf rdf:resource="&UseCase;SystemElement"/>
101   </owl:Class>
102   <owl:Class rdf:about="#SteamTurbine">
103     <rdfs:subClassOf rdf:resource="#Turbine"/>
104   </owl:Class>
105   <owl:Class rdf:about="#GasTurbine">
106     <rdfs:subClassOf rdf:resource="#Turbine"/>
107   </owl:Class>
108   <owl:Class rdf:about="#GastTurbine2103">
109     <rdfs:subClassOf rdf:resource="#GasTurbine"/>
110   </owl:Class>
111   <owl:Class rdf:about="#CombustionChamber">
112     <rdfs:subClassOf rdf:resource="&UseCase;SystemElement"/>
```

```
113   </owl:Class>
114   <owl:Class rdf:about="#Burner">
115     <rdfs:subClassOf rdf:resource="&UseCase;SystemElement"/>
116     <owl:equivalentClass rdf:resource="#CombustionChamber"/>
117   </owl:Class>
118   <owl:Class rdf:about="#PowerTurbine">
119     <rdfs:subClassOf rdf:resource="&UseCase;SystemElement"/>
120   </owl:Class>
121   <owl:Class rdf:about="#ControlUnit">
122     <rdfs:subClassOf rdf:resource="&UseCase;SystemElement"/>
123   </owl:Class>
124   <owl:Class rdf:about="#DriveTurbine">
125     <rdfs:subClassOf rdf:resource="&UseCase;SystemElement"/>
126   </owl:Class>
127   <owl:Class rdf:about="#Compressor">
128     <rdfs:subClassOf rdf:resource="&UseCase;SystemElement"/>
129   </owl:Class>
130   <owl:Class rdf:about="#Sensor">
131     <rdfs:subClassOf rdf:resource="&UseCase;SystemElement"/>
132   </owl:Class>
133   <owl:Class rdf:about="#TemperatureSensor">
134     <rdfs:subClassOf rdf:resource="#Sensor"/>
135   </owl:Class>
136   <owl:Class rdf:about="#TC255">
137     <rdfs:subClassOf rdf:resource="#TemperatureSensor"/>
138   </owl:Class>
139   <owl:Class rdf:about="#TC256">
140     <rdfs:subClassOf rdf:resource="#TemperatureSensor"/>
141   </owl:Class>
142   <owl:Class rdf:about="#TC257">
143     <rdfs:subClassOf rdf:resource="#TemperatureSensor"/>
144   </owl:Class>
145   <owl:Class rdf:about="#TC258">
146     <rdfs:subClassOf rdf:resource="#TemperatureSensor"/>
147   </owl:Class>
148   <owl:Class rdf:about="#TC259">
149     <rdfs:subClassOf rdf:resource="#TemperatureSensor"/>
150   </owl:Class>
151   <owl:Class rdf:about="#TC260">
152     <rdfs:subClassOf rdf:resource="#TemperatureSensor"/>
153   </owl:Class>
154
155
156   <owl:Class rdf:about="Symptom_2">
157     <owl:equivalentClass>
158       <owl:Restriction>
159         <owl:someValuesFrom>
160           <owl:Class>
161             <owl:intersectionOf rdf:parseType="Collection">
```

```
162                 <owl:Class rdf:about="TC255-Measurement"/>
163                 <owl:Restriction>
164                   <owl:allValuesFrom>
165                     <owl:DataRange>
166                       <j.0:minExclusive rdf:datatype="&xsd;double"
167                       >5</j.0:minExclusive>
168                       <j.0:onDataRange rdf:resource="&xsd;double"/>
169                     </owl:DataRange>
170                   </owl:allValuesFrom>
171                   <owl:onProperty rdf:resource="&UseCase;hasNumericalValue"/>
172                 </owl:Restriction>
173               </owl:intersectionOf>
174             </owl:Class>
175           </owl:someValuesFrom>
176           <owl:onProperty rdf:resource="&UseCase;hasFeature"/>
177         </owl:Restriction>
178       </owl:equivalentClass>
179       <rdfs:subClassOf rdf:resource="&UseCase;Symptom"/>
180     </owl:Class>
181
182     <owl:Class rdf:about="Symptom_1">
183       <owl:equivalentClass>
184         <owl:Restriction>
185           <owl:someValuesFrom>
186             <owl:Class>
187               <owl:intersectionOf rdf:parseType="Collection">
188                 <owl:Class rdf:about="#Events"/>
189                 <owl:Restriction>
190                   <owl:hasValue rdf:datatype="&xsd;string"
191                   >Can Flame Failure Detected</owl:hasValue>
192                   <owl:onProperty rdf:resource="&UseCase;hasStringValue"/>
193                 </owl:Restriction>
194               </owl:intersectionOf>
195             </owl:Class>
196           </owl:someValuesFrom>
197           <owl:onProperty rdf:resource="&UseCase;hasFeature"/>
198         </owl:Restriction>
199       </owl:equivalentClass>
200       <rdfs:subClassOf rdf:resource="&UseCase;Symptom"/>
201     </owl:Class>
202
203     <owl:Class rdf:about="Diagnosis_StartFailure">
204       <owl:equivalentClass>
205         <owl:Restriction>
206           <owl:someValuesFrom>
207             <owl:Class>
208               <owl:intersectionOf rdf:parseType="Collection">
209                 <owl:Restriction>
210                   <owl:someValuesFrom rdf:resource="#CombustionChamber"/>
```

```
211                 <owl:onProperty rdf:resource="&UseCase;hasPart"/>
212               </owl:Restriction>
213               <owl:Class>
214                 <owl:unionOf rdf:parseType="Collection">
215                   <owl:Restriction>
216                     <owl:someValuesFrom rdf:resource="#Symptom_1"/>
217                     <owl:onProperty rdf:resource="&UseCase;hasSymptom"/>
218                   </owl:Restriction>
219                   <owl:Restriction>
220                     <owl:someValuesFrom rdf:resource="#Symptom_2"/>
221                     <owl:onProperty rdf:resource="&UseCase;hasSymptom"/>
222                   </owl:Restriction>
223                 </owl:unionOf>
224               </owl:Class>
225             </owl:intersectionOf>
226           </owl:Class>
227         </owl:someValuesFrom>
228         <owl:onProperty rdf:resource="&UseCase;isDiagnosisFor"/>
229       </owl:Restriction>
230     </owl:equivalentClass>
231     <rdfs:subClassOf rdf:resource="&UseCase;Diagnosis"/>
232   </owl:Class>
233
234 </rdf:RDF>
```

Listing 8.18: Basisontologie

```
1  <?xml version="1.0"?>
2  <rdf:RDF
3      xmlns="http://www.siemens.com/SyD/SyD-pattern#"
4      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
6      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
7      xmlns:owl="http://www.w3.org/2002/07/owl#"
8      xmlns:owl11="http://www.w3.org/2006/12/owl11#"
9    xml:base="http://www.siemens.com/Optique/UseCase">
10   <owl:Ontology rdf:about=""/>
11   <rdfs:Datatype rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
12   <rdfs:Datatype rdf:about="http://www.w3.org/2001/XMLSchema#double"/>
13   <rdfs:Datatype rdf:about="http://www.w3.org/2001/XMLSchema#boolean"/>
14
15   <owl:Class rdf:ID="NumericalFeature">
16     <owl:disjointWith>
17       <owl:Class rdf:ID="StringFeature"/>
18     </owl:disjointWith>
19     <owl:disjointWith>
20       <owl:Class rdf:ID="BooleanFeature"/>
21     </owl:disjointWith>
22     <rdfs:subClassOf>
23       <owl:Restriction>
```

```
24          <owl:someValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
25          <owl:onProperty>
26            <owl:DatatypeProperty rdf:ID="hasNumericalValue"/>
27          </owl:onProperty>
28        </owl:Restriction>
29      </rdfs:subClassOf>
30      <rdfs:subClassOf>
31        <owl:Class rdf:ID="Feature"/>
32      </rdfs:subClassOf>
33    </owl:Class>
34    <owl:Class rdf:about="#Feature">
35      <owl:disjointWith>
36        <owl:Class rdf:ID="Symptom"/>
37      </owl:disjointWith>
38      <owl:disjointWith>
39        <owl:Class rdf:ID="Diagnosis"/>
40      </owl:disjointWith>
41      <owl:disjointWith>
42        <owl:Class rdf:ID="SystemElement"/>
43      </owl:disjointWith>
44      <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
45      <rdfs:subClassOf>
46        <owl:Restriction>
47          <owl:onProperty>
48            <owl:ObjectProperty rdf:ID="isFeatureFor"/>
49          </owl:onProperty>
50          <owl:someValuesFrom>
51            <owl:Class rdf:about="#Symptom"/>
52          </owl:someValuesFrom>
53        </owl:Restriction>
54      </rdfs:subClassOf>
55      <owl:equivalentClass>
56        <owl:Class>
57          <owl:unionOf rdf:parseType="Collection">
58            <owl:Class rdf:about="#BooleanFeature"/>
59            <owl:Class rdf:about="#NumericalFeature"/>
60            <owl:Class rdf:about="#StringFeature"/>
61          </owl:unionOf>
62        </owl:Class>
63      </owl:equivalentClass>
64    </owl:Class>
65    <owl:Class rdf:about="#Diagnosis">
66      <owl:disjointWith>
67        <owl:Class rdf:about="#SystemElement"/>
68      </owl:disjointWith>
69      <owl:disjointWith>
70        <owl:Class rdf:about="#Symptom"/>
71      </owl:disjointWith>
72      <owl:disjointWith rdf:resource="#Feature"/>
```

```
73      <rdfs:subClassOf>
74        <owl:Restriction>
75          <owl:someValuesFrom>
76            <owl:Class rdf:about="#SystemElement"/>
77          </owl:someValuesFrom>
78          <owl:onProperty>
79            <owl:ObjectProperty rdf:ID="isDiagnosisFor"/>
80          </owl:onProperty>
81        </owl:Restriction>
82      </rdfs:subClassOf>
83      <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
84    </owl:Class>
85    <owl:Class rdf:about="#StringFeature">
86      <owl:disjointWith rdf:resource="#NumericalFeature"/>
87      <owl:disjointWith>
88        <owl:Class rdf:about="#BooleanFeature"/>
89      </owl:disjointWith>
90      <rdfs:subClassOf>
91        <owl:Restriction>
92          <owl:someValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
93          <owl:onProperty>
94            <owl:DatatypeProperty rdf:ID="hasStringValue"/>
95          </owl:onProperty>
96        </owl:Restriction>
97      </rdfs:subClassOf>
98      <rdfs:subClassOf rdf:resource="#Feature"/>
99    </owl:Class>
100   <owl:Class rdf:about="#Symptom">
101     <rdfs:subClassOf>
102       <owl:Restriction>
103         <owl:someValuesFrom>
104           <owl:Class rdf:about="#SystemElement"/>
105         </owl:someValuesFrom>
106         <owl:onProperty>
107           <owl:ObjectProperty rdf:ID="isSymptomFor"/>
108         </owl:onProperty>
109       </owl:Restriction>
110     </rdfs:subClassOf>
111     <owl:disjointWith rdf:resource="#Feature"/>
112     <owl:disjointWith rdf:resource="#Diagnosis"/>
113     <owl:disjointWith>
114       <owl:Class rdf:about="#SystemElement"/>
115     </owl:disjointWith>
116     <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
117   </owl:Class>
118   <owl:Class rdf:about="#BooleanFeature">
119     <owl:disjointWith rdf:resource="#StringFeature"/>
120     <owl:disjointWith rdf:resource="#NumericalFeature"/>
121     <rdfs:subClassOf>
```

```
122         <owl:Restriction>
123           <owl:someValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
124           <owl:onProperty>
125             <owl:DatatypeProperty rdf:ID="hasBooleanValue"/>
126           </owl:onProperty>
127         </owl:Restriction>
128       </rdfs:subClassOf>
129       <rdfs:subClassOf rdf:resource="#Feature"/>
130     </owl:Class>
131     <owl:Class rdf:about="#SystemElement">
132       <owl:disjointWith rdf:resource="#Symptom"/>
133       <owl:disjointWith rdf:resource="#Feature"/>
134       <owl:disjointWith rdf:resource="#Diagnosis"/>
135       <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
136     </owl:Class>
137     <owl:ObjectProperty rdf:ID="isDirectPartOf">
138       <rdfs:range rdf:resource="#SystemElement"/>
139       <rdfs:domain rdf:resource="#SystemElement"/>
140       <rdfs:subPropertyOf>
141         <owl:TransitiveProperty rdf:ID="isPartOf"/>
142       </rdfs:subPropertyOf>
143       <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
144     </owl:ObjectProperty>
145     <owl:ObjectProperty rdf:ID="hasDiagnosis">
146       <owl:inverseOf rdf:resource="#isDiagnosisFor"/>
147       <rdfs:range rdf:resource="#Diagnosis"/>
148       <rdfs:domain rdf:resource="#SystemElement"/>
149       <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
150     </owl:ObjectProperty>
151     <owl:ObjectProperty rdf:ID="hasPart"/>
152     <owl:ObjectProperty rdf:about="#isFeatureFor">
153       <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
154     </owl:ObjectProperty>
155     <owl:DatatypeProperty rdf:about="#hasBooleanValue">
156       <rdfs:subPropertyOf>
157         <owl:FunctionalProperty rdf:ID="hasValue"/>
158       </rdfs:subPropertyOf>
159       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
160       <rdfs:domain rdf:resource="#BooleanFeature"/>
161     </owl:DatatypeProperty>
162     <owl:DatatypeProperty rdf:about="#hasStringValue">
163       <rdfs:subPropertyOf>
164         <owl:FunctionalProperty rdf:about="#hasValue"/>
165       </rdfs:subPropertyOf>
166       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
167       <rdfs:domain rdf:resource="#StringFeature"/>
168     </owl:DatatypeProperty>
169     <owl:DatatypeProperty rdf:about="#hasNumericalValue">
170       <rdfs:subPropertyOf>
```

```
171         <owl:FunctionalProperty rdf:about="#hasValue"/>
172       </rdfs:subPropertyOf>
173       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
174       <rdfs:domain rdf:resource="#NumericalFeature"/>
175     </owl:DatatypeProperty>
176     <owl:TransitiveProperty rdf:about="#isPartOf">
177       <rdfs:range rdf:resource="#SystemElement"/>
178       <rdfs:domain rdf:resource="#SystemElement"/>
179       <owl:inverseOf rdf:resource="#hasPart"/>
180       <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
181     </owl:TransitiveProperty>
182     <owl:FunctionalProperty rdf:about="#hasValue">
183       <rdfs:domain rdf:resource="#Feature"/>
184       <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
185     </owl:FunctionalProperty>
186     <owl:FunctionalProperty rdf:ID="hasName">
187       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
188       <rdfs:domain rdf:resource="#Feature"/>
189       <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
190     </owl:FunctionalProperty>
191     <owl:InverseFunctionalProperty rdf:ID="hasFeature">
192       <rdfs:range rdf:resource="#Feature"/>
193       <rdfs:domain rdf:resource="#Symptom"/>
194       <owl:inverseOf rdf:resource="#isFeatureFor"/>
195     </owl:InverseFunctionalProperty>
196     <owl:InverseFunctionalProperty rdf:ID="hasSymptom">
197       <owl:inverseOf rdf:resource="#isSymptomFor"/>
198       <rdfs:range rdf:resource="#Symptom"/>
199       <rdfs:domain rdf:resource="#SystemElement"/>
200     </owl:InverseFunctionalProperty>
201     <owl:AnnotationProperty rdf:about="http://www.w3.org/2002/07/owl#versionInfo"/>
202
203     <!-- Added by Nikita Khristianov -->
204
205     <owl:ObjectProperty rdf:about="#generatedFeature">
206           <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/
                >
207           <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
208               >Inverse of element.</rdfs:comment>
209           <rdfs:range rdf:resource="#Feature"/>
210           <rdfs:domain rdf:resource="#SystemElement"/>
211       </owl:ObjectProperty>
212
213     <owl:ObjectProperty rdf:about="#element">
214           <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
215           <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">element</
                rdfs:label>
216           <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
217             >This property is the description of an System Element used to generate it.</
                    rdfs:comment>
218         <rdfs:domain rdf:resource="#Feature"/>
219         <rdfs:range rdf:resource="#SystemElement"/>
220         <owl:inverseOf rdf:resource="#generatedFeature"/>
221     </owl:ObjectProperty>
222
223 </rdf:RDF>
```

Literaturverzeichnis

Arasu u. a. 2006

Arasu, Arvind ; Babu, Shivnath ; Widom, Jennifer: The CQL continuous query language: semantic foundations and query execution. In: *The VLDB Journal* (2006), S. 121–142. http://dx.doi.org/10.1007/s00778-004-0147-z. – DOI 10.1007/s00778–004–0147–z. – ISSN 1066–8888

Arasu u. a. 2004

Arasu, Arvind ; Cherniack, Mitch ; Galvez, Eduardo F. ; Maier, David ; Maskey, Anurag ; Ryvkina, Esther ; Stonebraker, Michael ; Tibbetts, Richard: Linear Road: A Stream Data Management Benchmark. In: *VLDB*, Morgan Kaufmann, 2004. – ISBN 0–12–088469–0, S. 480–491

Barbieri u. a. 2010

Barbieri, Davide F. ; Braga, Daniele ; Ceri, Stefano ; Grossniklaus, Michael: An execution environment for C-SPARQL queries. In: *Proceedings of the 13th International Conference on Extending Database Technology*, ACM, 2010 (EDBT '10). – ISBN 978–1–60558–945–9, S. 441–452

blazeds

Adobe Open Source BlazeDS. http://opensource.adobe.com/wiki/display/blazeds/BlazeDS,

Bolles u. a. 2008

Bolles, Andre ; Grawunder, Marco ; Jacobi, Jonas: Streaming SPARQL - Extending SPARQL to Process Data Streams. Version: 2008. http://dx.doi.org/10.1007/978-3-540-68234-9-34. In: *The Semantic Web: Research and Applications*. Springer Berlin Heidelberg, 2008. – DOI 10.1007/978–3–540–68234–9–34. – ISBN 978–3–540–68233–2, S. 448–462

Calbimonte u. a. 2010

Calbimonte, Jean-Paul ; Corcho, Oscar ; Gray, Alasdair J. G.: Enabling ontology-based access to streaming data sources. In: *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I*, Springer-Verlag, 2010 (ISWC'10). – ISBN 3–642–17745–X, 978–3–642–17745–3, S. 96–111

census

Data rendering and loading performance comparison for RIA technolohies. http://www.jamesward.com/census2/,

cqels

CQELS-Projekt website. http://code.google.com/p/cqels/,

csparql

Continuous SPARQL (C-SPARQL) download. http://streamreasoning.org/download,

cusser

Datapush using custom java adapter to Flex apps. http://flexatom.com/?p=320,

flexsdk

Using Flex 4. http://help.adobe.com/de_DE/Flex/4.0/UsingSDK/index.html,

Golab und Zsu 2010

Golab, L. ; Zsu, T.: *Data Stream Management*. Morgan & Claypool, 2010 (Synthesis Lectures on Data Management Series). – ISBN 9781608452729

Golab und Özsu 2003

Golab, Lukasz ; Özsu, M T.: Processing sliding window multi-joins in continuous queries over data streams. In: *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB Endowment, 2003 (VLDB '03). – ISBN 0–12–722442–4, S. 500–511

gsblazeds

Getting started with BlazeDS. http://www.adobe.com/devnet/livecycle/articles/blazeds_gettingstarted.html,

Guo u. a. 2005

Guo, Yuanbo ; Pan, Zhengxiang ; Heflin, Jeff: LUBM: A benchmark for OWL knowledge base systems. In: *Web Semantics: Science, Services and Agents on the World Wide Web* (2005), S. 158–182. http://dx.doi.org/10.1016/j.websem.2005.06.005. – DOI 10.1016/j.websem.2005.06.005. – ISSN 15708268

Hitzler u. a. 2008

Hitzler, P. ; Krötzsch, M. ; Rudolph, S. ; Sure, Y.: *Semantic Web: Grundlagen*. Springer London, Limited, 2008. – ISBN 9783540339946

Horrocks u. a. 1999

Horrocks, Ian ; Sattler, Ulrike ; Tobies, Stephan: Practical Reasoning for

Expressive Description Logics. In: *Proceedings of the 6th International Conference on Logic Programming and Automated Reasoning*, Springer-Verlag, 1999. – ISBN 3–540–66492–0, S. 161–180

Khristianov 2012

Khristianov, Nikita: *Vergleich der RIA-Technologien zur Implementierung einer Webanwendung zur Vereinheitlichung logistischer VorgÃ¤nge*, TU Hamburg-Harburg, Studienarbeit, Februar 2012

knoesis

The Ohio Center of Excellence in Knowledge-enabled Computing. `http://knoesis.org/aboutus`,

Le-Phuoc u. a. 2011

Le-Phuoc, Danh ; Dao-Tran, Minh ; Parreira, Josiane X. ; Hauswirth, Manfred: A native and adaptive approach for unified processing of linked streams and linked data. In: *Proceedings of the 10th international conference on The semantic web - Volume Part I*, Springer-Verlag, 2011 (ISWC'11). – ISBN 978–3–642–25072–9, S. 370–388

Le-Phuoc u. a. 2012a

Le-Phuoc, Danh ; Dao-Tran, Minh ; Pham, Minh-Duc ; Boncz, Peter ; Eiter, Thomas ; Fink, Michael: Linked stream data processing engines: facts and figures. In: *Proceedings of the 11th international conference on The Semantic Web - Volume Part II*, Springer-Verlag, 2012 (ISWC'12). – ISBN 978–3–642–35172–3, S. 300–312

Le-Phuoc u. a. 2012b

Le-Phuoc, Danh ; Xavier Parreira, Josiane ; Hauswirth, Manfred: Linked Stream Data Processing. In: *Reasoning Web. Semantic Technologies for Advanced Query Answering.* 2012. – ISBN 978–3–642–33157–2, S. 245–289

linroad

Linear Road: A Stream Data Management Benchmark. `http://www.cs.brandeis.edu/~linearroad/`,

loddia

The Linking Open Data cloud diagram. `http://lod-cloud.net/`,

lsbench

A benchmark for Linked Stream Data processing engines. `https://code.google.com/p/lsbench/`,

lsd

LinkedSensorData. http://wiki.knoesis.org/index.php/LinkedSensorData,

lubm

SWAT Projects - the Lehigh University Benchmark (LUBM). http://swat.cse.lehigh.edu/projects/lubm/,

optique

Optique project website. http://www.optique-project.eu/,

owl

OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/,

pellet

Pellet: OWL 2 Reasoner for Java. http://clarkparsia.com/pellet,

Pérez u. a. 2009

Pérez, Jorge ; Arenas, Marcelo ; Gutierrez, Claudio: Semantics and complexity of SPARQL. In: *ACM Trans. Database Syst.* (2009), S. 16:1–16:45. http://dx.doi.org/10.1145/1567274.1567278. – DOI 10.1145/1567274.1567278. – ISSN 0362–5915

Pisa 2009

Pisa, F.D.: *Beginning Java and Flex: Migrating Java, Spring, Hibernate and Maven Developers to Adobe Flex*. Apress, 2009 (Apress Series). – ISBN 9781430223856

racerpro

RacerPro 2.0 : A system overview. http://www.racer-systems.com/products/racerpro/preview/overview.phtml,

rdfs

RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/,

rdfxml

RDF/XML Syntax Specification. http://www.w3.org/TR/rdf-syntax-grammar,

Sequeda u. a. 2009

Sequeda, Juan ; Corcho, Oscar ; Taylor, Kerry ; Ayyagari, Arun ; Roure, David D.: Linked Stream Data: A Position Paper. In: *Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN09) at ISWC 2009*, 2009, S. 148–157

setser

Setting Up a Basic BlazeDS Server with Debugging. http://flexbandit.com/archives/55,

sparql

SPARQL: Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/,

sparql11

SPARQL 1.1: Query Language for RDF. http://www.w3.org/TR/sparql11-query/,

sparqlent

SPARQL 1.1 Entailment Regimes. http://www.w3.org/TR/2013/REC-sparql11-entailment-20130321,

sparqlstr

SPARQLStream. https://code.google.com/p/semanticstreams/wiki/SPARQLStream,

srbench

Streaming RDF/SPARQL Benchmark. http://www.w3.org/wiki/SRBench,

swwiki

Semantic Web wikipedia article. http://en.wikipedia.org/wiki/Semantic_Web,

Tucker und Maier 2002

Tucker, Tufte K. Papadimos V. P. A. A. P. A. ; Maier, D.: NEXMark – A benchmark for queries over data streams. (2002)

Valle u. a. 2009

Valle, Emanuele D. ; Ceri, Stefano ; Harmelen, Frank v. ; Fensel, Dieter: It's a Streaming World! Reasoning upon Rapidly Changing Information. In: *IEEE Intelligent Systems* (2009), S. 83–89. http://dx.doi.org/10.1109/MIS.2009.125. – DOI 10.1109/MIS.2009.125. – ISSN 1541–1672

Widjaja 2010

Widjaja, S.: *Adobe Flex 4*. Hanser Fachbuchverlag, 2010. – ISBN 9783446422681

xmark

XMark: An XML Benchmark Project. http://www.xml-benchmark.org/,

Zhang u. a. 2012

Zhang, Ying ; Duc, Pham M. ; Corcho, Oscar ; Calbimonte, Jean-Paul: SRBench: a streaming RDF/SPARQL benchmark. In: *Proceedings of the 11th international conference on The Semantic Web - Volume Part I*, Springer, 2012 (ISWC'12). – ISBN 978–3–642–35175–4, S. 641–657