



GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

ISSN 1612-6793

Master's Thesis

submitted in partial fulfilment of the
requirements for the course "Applied Computer Science"

Linked Open Data and its Evaluation

Martin Heinemann

Institute of Computer Science

Bachelor's and Master's Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

30. April 2019

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000

☎ +49 (551) 39-14403

✉ office@informatik.uni-goettingen.de

🌐 www.informatik.uni-goettingen.de

First Supervisor: Prof. Dr. Wolfgang May

Second Supervisor: Prof. Dr. Carsten Damm

I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Göttingen, 30. April 2019

Abstract

The term Linked Open Data describes data which are published under a open license and contain references to other data sets. In order to become part of this Web of Data and to understand the technology behind it, it was necessary to implement a SPARQL endpoint for the Mondial data set. During this implementation the question came up, how the interconnection of different data sets work and how queries are evaluated, if parts of this queries are executed on federated SPARQL endpoints.

ACKNOWLEDGEMENTS

First of all, I want to thank my thesis advisers Prof. Dr. Wolfgang May and Prof. Dr. Carsten Damm. In addition to hosting my topic, the doors to their offices were always open whenever I struggled with my research or my writing.

Secondly, I want to say thank you to Lars Runge and Sebastian Schrage, for being the best supervisors imaginable. Besides answering almost all my questions, the two of you became close friends of mine, and I hope, this friendship will last very long.

In third place, I want to thank my parents Christiane and Ernst August Heinemann. You allowed me to study as long as I needed and were always there, whenever I needed parental advice.

Last but not least, I want to thank my girlfriend Marthe Frischalowski. Without your encouragement and your advice I would never have been able to complete these studies. Thank you!

Contents

| | | |
|----------|---------------------------------------------------------------|-----------|
| 1 | Introduction | 1 |
| 2 | Basics | 3 |
| 2.1 | Used Knowledge Bases | 3 |
| 2.1.1 | Mondial | 3 |
| 2.1.2 | Wikidata | 4 |
| 2.1.3 | Insee | 4 |
| 2.2 | The Resource Description Framework | 4 |
| 2.3 | RDF Ontologies | 6 |
| 2.3.1 | RDF Schema | 7 |
| 2.3.2 | The Web Ontology Language | 7 |
| 2.4 | The SPARQL Protocol and RDF Query Language | 7 |
| 2.5 | Linked Open Data | 12 |
| 2.6 | Used Tools | 15 |
| 2.6.1 | The Jena-API | 15 |
| 2.6.2 | The SemWebJena-Tool | 15 |
| 2.6.3 | rapper | 15 |
| 2.6.4 | cURL | 16 |
| 3 | The Mondial LOD Service | 17 |
| 3.1 | Use Cases of Web Services | 17 |
| 3.2 | The Mondial LOD Web Service | 18 |
| 3.3 | Integrating the Mondial Data Set into the LOD-Cloud | 21 |
| 3.4 | Technical Issues | 22 |
| 3.5 | The Meta Data Access in Mondial’s SPARQL Endpoint | 23 |
| 3.6 | The Resolving of Blank Nodes | 23 |
| 3.6.1 | The Problem with Blank Node Resolving | 23 |
| 3.6.2 | Wikidata’s Way to Resolve Blank Nodes | 24 |
| 3.6.3 | How does Insee Resolve Blank Nodes? | 24 |

| | | |
|----------|------------------------------------------------------------------------------------------|-----------|
| 3.6.4 | Blank Node Resolving in the Mondial SPARQL Endpoint | 24 |
| 3.7 | Problems with Mondial's URIs | 28 |
| 3.8 | Evaluation of the Mondial LOD Web Service | 29 |
| 4 | Evaluation of Distributed Queries | 31 |
| 4.1 | Analysis of Queries within the Jena-API | 31 |
| 4.2 | Analysis of the <code>SERVICE</code> and the <code>VALUES</code> Keyword | 32 |
| 4.3 | Optimization of the <code>SERVICE</code> Clause Evaluation within the Jena-API | 35 |
| 4.4 | Evaluation of the <code>SERVICE</code> Clause Optimization | 41 |
| 5 | Conclusion | 47 |
| 5.1 | Future Work | 48 |
| | Bibliography | 50 |
| | Glossary | 51 |
| A | How to Integrate the Developed Optimizations into a new Jena-API | 55 |
| B | owl:sameAs Queries | 57 |
| C | Execution Times of Evaluation Queries | 67 |

Chapter 1

Introduction

When people talk about the World Wide Web, in most cases they refer to the so-called Web of Hypertexts, without knowing that the World Wide Web includes much more than only hypertexts. The reason for this might be, that the Web of Hypertexts is automatically accessed, whenever a web browser is opened and a website is called. This website is nothing else but an HTML representation of hypertext files using the Hypertext Markup Language (HTML). On most of these websites, there are references to another website, and from this other website, again there are references to a third website, and so on. Over the last three decades, the amount of websites, and, therefore, the amount of links between those websites on the World Wide Web continuously grew and, thus, formed the Web of Hypertexts.

Another, much younger, part of the World Wide Web, however, is the so-called Web of Data. It is build analogously to the Web of Hypertexts, but instead of linking websites to each other, within the Web of Data different data sets are connected. If these connected data are openly accessible and published under an open license, one speaks of Linked Open Data (LOD) [Ber06]. While for describing such data the Resource Description Framework (RDF) [KCM14] published by the World Wide Web Consortium (W3C) is commonly used, to query them, the SPARQL Protocol and RDF Query Language (SPARQL) [HS13] can be used.

To make RDF data accessible via the Web of Data, it is necessary to set up an LOD endpoint. Therefore, it is possible to implement a Java servlet, which is an application running on a web server. This servlet then awaits SPARQL queries and returns the results in the requested format. To handle RDF data and SPARQL queries within Java code, the Jena-API developed and published by [Fou] can be used.

A database, which already was available in RDF format, but not yet accessible via SPARQL is the Mondial database of the Databases and Information Systems (DBIS) group of the University of Göttingen. This database contains information about geographical and geopolitical entities like rivers and mountains, or countries and organizations. Furthermore, the Mondial data set was not

yet connected to any other data set and, therefore, was part of Open Data, but not of LOD. Creating a working Mondial LOD servlet and integrating the Mondial data set into LOD was one of the major tasks addressed by this thesis. In order to do so, some references between the Mondial data set and, at least, one other data set had to be created.

The second major part of this thesis was highly related to SPARQL's ability to execute queries on a federated SPARQL endpoint, in order to combine data from multiple different data sets within a single query. Therefore, the `SERVICE` keyword of SPARQL together with the Uniform Resource Locator (URL) of the corresponding SPARQL endpoint can be used. Thus, the second part of this thesis was, to analyze the evaluation of such a `SERVICE` clause by the Jena-API and, if possible, to optimize this evaluation.

In Chapter 2, all basics needed for this thesis will be introduced and explained. Chapter 3, on the one hand, covers the analysis of various SPARQL endpoints, in order to find out how the Mondial SPARQL endpoint could be set up, and, on the other hand, describes the creation of the Mondial LOD web service. In Chapter 4, the Jena-API, respectively its ARQ package, is analyzed with focus on query evaluation and if this process can be optimized. Additionally, an attempt to optimize the query execution on federated SPARQL endpoints is described and evaluated. The final chapter of this thesis, Chapter 5, summarizes all important information, and gives an outlook to possible future works, which could be done, based on this thesis.

Chapter 2

Basics

In this chapter, the theoretical basics as well as the tools used during this thesis are explained. In order to integrate the Mondial knowledge base into LOD, two additional knowledge bases, each containing data about geographical and geopolitical resources, were analyzed.

2.1 Used Knowledge Bases

For this thesis three different knowledge bases were used. While the Mondial knowledge base is a major part of this thesis, the Insee knowledge base is used to query data only. The Wikidata knowledge base is for one thing used to query data and to test certain optimizations and for another thing, to create an `owl:sameAs` file. This file is needed in order to create links between databases and, therefore, to provide the "Linked" of the term Linked Open Data. Each of these knowledge bases is explained in detail in the following sections.

2.1.1 Mondial

The Mondial knowledge base contains information about multiple kinds of geographical entities, like mountains or rivers, but also about geopolitical entities, like countries and organizations. It is hosted and maintained by the DBIS group of the Georg-August-Universität Göttingen. The initial setup of Mondial was in 1998 and it is updated on a more or less regular base ever since. Even though Mondial contains a big amount of data, it is not complete, which means, that there are multiple entities which exist in the real world, but are not part of Mondial. This is, because Mondial is primarily used for academical purposes only, and, therefore, is available in multiple different data formats like RDF or as a SQL data base. A list of all formats, as well as a further description of Mondial, can be found at <https://www.dbis.informatik.uni-goettingen.de/Mondial/>.

Much more information than in the Mondial knowledge base is stored, for example, within the Wikidata knowledge base.

2.1.2 Wikidata

Wikidata (<https://www.wikidata.org>) is a huge knowledge base containing more than 55 million items (March 2019). It is completely open, which means, that all data within Wikidata are accessible and editable by anyone. As its name indicates, Wikidata is part of the Wikimedia family and, therefore, a sibling project to the famous Wikipedia (<https://www.wikipedia.org/>). In contrast to the Mondial data set, Wikidata contains information about all kinds of entities, for example, planets, books, or significant historic events, like the Defenestrations of Prague. This is one major aspect that sets Wikidata apart from other, more focused knowledge bases like Insee.

2.1.3 Insee

The knowledge base of the Insitut National de la Statistique et des Etudes Economiques (Insee) is hosted at <https://insee.fr/>, and contains information about the French economy and society. Additionally, the Insee knowledge base is used to analyze and spread these information.

To store information in a knowledge base, a special data format, the Resource Description Framework (RDF), can be used.

2.2 The Resource Description Framework

On October 2, 1997 the World Wide Web Consortium (W3C), an organization which develops standards for the Internet, released the first working draft for a new standard for data interchange on the World Wide Web. This standard is called the Resource Description Framework (RDF) and is available in version 1.1, since February 15, 2014 [KCM14].

RDF describes the world in triples, where each triple is called an RDF statement and consists of the three parts *Subject*, *Predicate*, and *Object*. All three parts of an RDF statement can be entities represented by a Uniform Resource Identifier (URI), which is a worldwide unique identifier for any kind of resource, similar to the Uniform Resource Locator (URL) used for websites. Additionally, the subject of an RDF statement can be a blank node, meaning that it is a reified statement, while the object also can be a blank node or a literal. A blank node in object position of an RDF statement means, that it is a resource not meant to have an identifier. If the object is a literal, it can have an absolute value of some data type, like "Integer", "Float", or "String", defined by [BM12]. Instead of defining a literal as a String, it is also possible to assign a language tag like "@en" or "@de" to the literal. Each literal with language tag is automatically assumed to be a String.

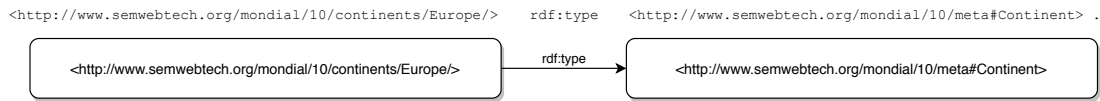


Figure 2.1: A simple RDF graph. It shows the graphical interpretation of Line 6 of Listing 2.1 where the continent "Europe" is defined as a Mondial-Continent.

Because it can be easily translated into a directed graph, a set of RDF statements is also called an RDF graph. For this transformation, the subjects and objects transfer into the graph's nodes, while the predicates become directed, labeled edges. Even blank nodes can be represented in an RDF graph, by a node without label. One of the most simplest RDF graphs is shown in Figure 2.1, defining that the URI belonging to the Mondial-Continent "Europe" describes a Mondial-Continent.

In June 2001, the first draft of N-Triples was released by the RDF Core Working Group of the W3C [RDF01]. Until August 2005, N-Triples was called N3, but since then, the name N3 is meant to be an abbreviation for the Notation3 Logic. N-Triples' syntax is human readable, but for [BB08] still too lengthy and unreadable. This is the reason that the Terse RDF Triple Language (Turtle) was invented and released as a W3C team submission on January 14, 2008. In both cases, RDF statements are written as triples, each ending with a "." (full stop). The major difference between N-Triples (Listing 2.1 Line 4) and Turtle (Listing 2.1 Line 6) is, that Turtle allows the usage of prefixes (Listing 2.1 Lines 1 and 2) and, therefore, abbreviations for repeatedly used patterns. Another way to shorten RDF data is, to make use of the , (comma) and ; (semicolon) operators (Listing 2.1 Line 10). The ; operator can be used to concatenate multiple RDF statements referring to the same subject, but different predicates and objects. The , operator, however, indicates, that the subject and the predicate stay the same, and only the object changes.

Listing 2.1: Example for the RDF syntaxes N-Triples and Turtle. In Line 4 the definition of the country "Germany" as Mondial-Country is shown in N-Triples syntax. Lines 1, 2, and 6 belong to the Turtle syntax and define, that the URI of the Mondial-Continent "Europe" describes a Mondial-Continent. Line 8 shows the declaration of a Mondial-PopulationCount for "Germany", which is defined as an object blank node. In the last line (Line 10) the definition of a subject blank node is shown.

This listing is meant to be an example and is created in such a way. Even though Mondial URIs are used, these lines do not appear in the original Mondial.n3 file.

```

1 @PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#> .
2 @PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3
4 <http://www.semwebtech.org/mondial/10/countries/D>
   ↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   ↪ <http://www.semwebtech.org/mondial/10/meta#Country> .
5
6 <http://www.semwebtech.org/mondial/10/continents/Europe> rdf:type mon:Continent .
7
8 <http://www.semwebtech.org/mondial/10/countries/D> mon:hadPopulation [ rdf:type
9   ↪ mon:PopulationCount; mon:year 2011; mon:value 80219695] .
10 [ rdf:type mon:Border ; mon:length 456; mon:isBorderOf
   ↪ <http://www.semwebtech.org/mondial/10/country/D/> ,
   ↪ <http://www.semwebtech.org/mondial/10/country/PL/>] .

```

The given example shows, that it is easily possible to express information using RDF. Besides this, another main reason why RDF is used is, that databases can be transformed into N-Triples syntax very easily. This frequent use of RDF made it clear, that certain definitions in this world will be used repeatedly and are worth to be defined permanently. Therefore, the W3C released two ontologies named RDF Schema (RDFS) and the Web Ontology Language (OWL).

2.3 RDF Ontologies

In computer science, especially in context of the Web of Data, the term ontology can be seen as a vocabulary. Ontologies, for example, define classes and relationships and can be created by anyone. The ontology of the Mondial knowledge base can be found at <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial-meta.n3>. Such ontologies can be used to define relations between classes and properties. By convention, there are some prefixes which are reserved for the ontologies defined by the W3C. These are, for example, the `owl` prefix for the OWL ontology and the `rdfs` prefix for the RDFS ontology.

2.3.1 RDF Schema

The first recommendation of the RDF Schema (RDFS) ontology was published by the W3C on March 3, 1999. Its newest version 1.1 was released on February 25, 2014 by [BG14]. Besides many basic things, like `rdfs:Class`, RDFS introduces a hierarchy of classes by defining properties like `rdfs:subClassOf`. It also defines the most basic class in RDF called `rdfs:Resource`. This means, that everything in every ontology is a descendant of this class. One of the most frequently used properties of RDFS is the `rdfs:label`, which assigns a human-readable label to any resource it is used for. To define relationships between different classes, the OWL ontology can be used.

2.3.2 The Web Ontology Language

The Web Ontology Language (OWL) was released as working draft on July 29, 2002 by [Mv02] and is nowadays available in version 2 which was lastly updated on the December 11, 2012 by [OWL12]. It defines, for example, relationships between classes like `owl:disjointWith` and introduces cardinalities, which can be used to express that a specific property has to be assigned to a class, for example, at most once or at least three times. Another very important definition by OWL is the `owl:sameAs` property. It describes that the resources connected by this predicate refer to the same real world object.

To query RDF data and to make use of such ontologies, a special query language, called SPARQL, has been defined.

2.4 The SPARQL Protocol and RDF Query Language

The SPARQL Protocol and RDF Query Language (SPARQL) is a language to query RDF data. Its first draft was released on October 12, 2004 by [PS04] and is available in version 1.1 [HS13] since March 21, 2013.

The syntax of SPARQL is highly related to the syntax of the common query language SQL for relational databases. SPARQL also uses the `SELECT-FROM-WHERE`-syntax but in opposite to SQL the `FROM` clause of SPARQL can be left out, if the queried data set is specified elsewhere. The `WHERE` clause of a SPARQL query has to be surrounded by `{` and `}` (curly brackets).

The most simple queries in SPARQL consist of triple patterns only and are called Basic Graph Patterns (BGPs) (Listing 2.2 Lines 4 and 5). Within a BGP, the RDF statements are evaluated in a conjunctive way, which means that they are connected by a logical `AND`. Each of the BGPs values can be a variable, which in SPARQL either start with a `?` like `?country` or with a `$` like `$country`, with both notations describing the same variable. Like RDF, SPARQL can use a Turtle-style syntax which allows the definition and usage of prefixes at the beginning of each SPARQL query.

Listing 2.2: Simple SPARQL query to ask for all Mondial-Volcanos and their elevation. Lines 4 and 5 show an example of a SPARQL BGP, where Line 4 additionally introduces different sorts of SPARQL abbreviations. It can be seen, that the aforementioned `,` and `;` operators of the Turtle syntax can be used within SPARQL queries, too.

```
1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT ?volcano ?name ?elevation
3 WHERE {
4   ?volcano a mon:Mountain, mon:Volcano; :name ?name.
5   ?volcano mon:elevation $elevation.
6 }
```

SPARQL uses many different keywords to state its queries. The most important one is the `a` (abbreviation of: is a) keyword, which is also a part of the Turtle syntax. It refers to the `http://www.w3.org/1999/02/22-rdf-syntax-ns#type` property and is the most frequently used predicate in RDF and SPARQL. Therefore, the `a` keyword has to be used without prefix and can be used without defining a prefix. When used with a prefix, like `mon:a`, it never refers to the `http://www.w3.org/1999/02/22-rdf-syntax-ns#type` property.

Other keywords used in SPARQL distinguish between four different forms of queries. The `SELECT` keyword initiates a query that returns all matching bindings for a subset of all variables used within the query (Listing 2.2). A binding is a key-value-pair, where the key is the variable, and the value the value bound to this variable. `SELECT` queries are the most frequently used queries in SPARQL.

Another form of query is the `CONSTRUCT` query (Listing 2.3). This form is similar to the `SELECT` form, but, instead of a list of bindings, it returns an RDF graph representing the results.

The third SPARQL query form is the `ASK` query, as shown in Listing 2.4. It returns a Boolean value, indicating if the stated query has any result at all.

The last query form is the `DESCRIBE` form. It returns an RDF graph containing information about resources, but neither [HS13] nor [PS04] define what this RDF graph contains. They leave the responsibility to decide what is the content of this RDF graph with the query processor and, therefore, with the maintainer of the queried data set.

Listing 2.3: An example query using the CONSTRUCT query form. The result of this query will be an RDF graph containing all Mondial-Countries, their name, and their area.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX demo: <http://www.this.is/an/example/>
3 CONSTRUCT {
4     ?X demo:name ?name.
5     ?X demo:area ?area.
6 }
7 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8 WHERE{
9     ?X a mon:Country.
10    ?X mon:name ?name.
11    ?X mon:area ?area.
12 }

```

Listing 2.4: An example query using the ASK query form. This query will return true, indicating that there is at least one Mondial-Country in the Mondial data set.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 ASK
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE{
5     ?country a mon:Country.
6 }

```

With the OPTIONAL keyword (Listing 2.5), SPARQL delivers a way to express the algebraic operation LEFT OUTER JOIN. This means, that if an RDF statement returns no result for the query within the OPTIONAL clause, its binding for the query part outside the OPTIONAL clause is still returned to the user. Only the variable entries for the variables within the OPTIONAL are left empty.

Listing 2.5: In this query the OPTIONAL keyword is demonstrated. The query asks for all countries in Mondial and if they have a city, whose name equals the name of some country, this city is returned as well.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT DISTINCT ?country ?countryName ?city
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5     ?country a mon:Country.
6     ?country mon:name ?countryName.
7     ?country mon:hasCity ?X .
8     OPTIONAL {
9         ?city a mon:City.
10        ?city mon:name ?countryName.
11    }
12 }

```

To explicitly omit unwanted results from a result set, the logical difference of two RDF graphs can be built. SPARQL can do this in two different ways, either by using the `MINUS` keyword (Listing 2.6) or by using the `FILTER NOT EXISTS` keywords (Listing 2.7). Even though both ways are often assumed to be equivalent, there is a slight difference in these expressions and they describe different algebraic operations. The `MINUS` keyword searches for all bindings in the result set matching the triples within the `MINUS` clause and removes them from the original result set. The `FILTER NOT EXISTS` clause, however, searches for all bindings that exist within its clause, negates this result set and filters the original result set of all matching patterns. This difference is critical in cases where variables are involved. The expression in Listing 2.6 will return all Mondial-Organizations, because neither of these variables bound within the `MINUS` clause is used outside of it. The `FILTER NOT EXISTS` clause in Listing 2.7, however, will return an empty result set, because the triple `?a ?b ?c` matches any triple, so it evaluates to true.

Listing 2.6: This query shows the syntax of the `MINUS` clause. The result set will contain all organizations stored in Mondial, because the triple `?a ?b ?c` is not bound outside of the `MINUS` clause.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT DISTINCT ?organization
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5     ?organization a mon:Organization.
6     MINUS {
7         ?a ?b ?c.
8     }
9 }

```

Listing 2.7: An example query to show the usage of the `FILTER NOT EXISTS` clause. The result set of this query will be empty, because the triple `?organization a mon:Organization.` matches any triple `?a ?b ?c`, so it evaluates to true.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT DISTINCT ?organization
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5     ?organization a mon:Organization.
6     FILTER NOT EXISTS {
7         ?a ?b ?c.
8     }
9 }

```

In order to delete single results from the query result set, SPARQL provides a `FILTER` keyword (Listing 2.8). With this keyword, it is either possible to filter out a single value for a specific variable (Listing 2.8 Line 6) or, with the concatenation of expressions, to filter multiple variables for multiple values (Listing 2.8 Line 7). Since SPARQL version 1.1 the restriction of the results to multiple

values for variables became much easier, by introducing the `VALUES` keyword (Listing 2.9).

Listing 2.8: Demonstration of the `FILTER` keyword syntax. It returns all countries, their names, their cities, and the city names. In Line 6, all cities with the name "New York" are deleted from the result set. Line 7 removes almost every other result, except those, where the country name is "Italy" and the corresponding city name is "Pisa" or where the country name is "Germany" and the city name is "Göttingen".

```

1 SELECT DISTINCT ?country ?countryName ?city ?cityName
2 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
3 WHERE {
4     ?country a :Country; :name ?countryName ; :hasCity ?city.
5     ?city :name ?cityName.
6     FILTER(?cityName != "New York")
7     FILTER((?countryName = "Germany" && ?cityName = "Göttingen") || (?countryName =
      ↪ "Italy" && ?cityName = "Pisa"))
8 }

```

Listing 2.9: This query shows the usage of the `VALUES` keyword in SPARQL. It returns all countries and cities, each with the corresponding name, where the country name is "Italy" and the corresponding city name is "Pisa", or where the country name is "Germany" and the city name is "Göttingen". Line 6 will cause the same effect as Line 7 of Listing 2.8

```

1 SELECT DISTINCT ?country ?countryName ?city ?cityName
2 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
3 WHERE {
4     ?country a :Country; :name ?countryName ; :hasCity ?city.
5     ?city :name ?cityName.
6     VALUES (?countryName ?cityName) { ("Germany" "Göttingen") ("Italy" "Pisa")}
7 }

```

In some cases, querying data from one data set only, might not be sufficient. Therefore, SPARQL has the `SERVICE` keyword, which allows a user to query data from a federated SPARQL endpoint. A SPARQL endpoint is nothing else as a Hypertext Transfer Protocol (HTTP) servlet, running on a web server, waiting to answer queries. If one, for example, wants to combine data of the Mondial and the Insee data set, this can be done as shown in Listing 2.10.

Listing 2.10: A demonstration of the `SERVICE` keyword. The user wants to know, which French city, stored in Mondial, corresponds to which French commune and which French department. These data are not stored in Mondial and, therefore, the `SERVICE` keyword has to be used. Because all string data in Insee have a French language tag ("`@fr`"), the province and the city name of Mondial have to be tagged. This is done in Lines 10 and 11

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX insee: <http://rdf.insee.fr/def/geo#>
3 SELECT DISTINCT ?cityName ?commune ?departmentName ?provinceNameFr
4 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
5 WHERE {
6     ?country a mon:Country; mon:carCode 'F'; mon:name ?countryName ; mon:hasCity ?city .
7     ?province a mon:Province; mon:hasCity ?city .
8     ?city mon:name ?cityName.
9     ?province mon:name ?provinceName .
10    BIND (STRLANG(?cityName,"fr") AS ?cityNameFr) .
11    BIND (STRLANG(?provinceName,"fr") AS ?provinceNameFr) .
12    SERVICE <http://rdf.insee.fr/sparql> {
13        ?commune a insee:Commune; insee:nom ?cityNameFr; insee:subdivisionDe ?department .
14        ?department a insee:Departement; insee:nom ?departmentName; insee:subdivisionDe
15        ↪ ?provinceInsee .
16        ?provinceInsee insee:nom ?provinceNameFr
17    }

```

Often a `SERVICE` clause is used get more information about an already known resource. If this is done more frequently, the execution time of the SPARQL query can be improved significantly, by creating an `owl:sameAs` entry for the resource and, therefore, link the two data sets to each other. This is what it means to deal with Linked Open Data.

2.5 Linked Open Data

The term "Linked Data" came up in 2006 when [Ber06] defined it on his website and describes data that are part of the Web of Data. As the name suggests, the Web of Data is a network of interconnected RDF data, where the way of the connection can be of any kind. To check, whether data are Linked Data, [Ber06] defined the following four rules.

1. Use URIs as names of things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up an URI, provide useful information, using the standards RDF and SPARQL
4. Include links to other URIs, so that they can discover more things

Linked Data only defines that data have to be linked to each other, but it does not mention the accessibility of these data. This, however, is defined by the term "Linked Open Data (LOD)" where data explicitly have to be published under an open access licence. Therefore, the four rules of Linked Data have been transformed into a five star rating system, describing to which part the current data are official LOD. This rating system looks like this:

- ★ Data are available on the web in any format but with an open licence
- ★★ Data are available as machine-readable structured data
- ★★★ Data are available as (2) but in a non-proprietary format
- ★★★★ All from the above and open standards of the W3C are used to identify things
- ★★★★★ All from the above and other data sets are linked within your data

The major advantage of using LOD is, that a single data set like the Mondial data set might be incomplete, but with LOD it became remarkable easy to check for resources on other data sets, like Wikidata. Even though, if the data of the second data set are not more complete than the original data, a combination of these data sets might be useful, because one data set might have more up-to-date data than the other one. In such a case it is important to remember, that the data sets, when queried together, do not interfere with each other. To create links between data set, the `owl:sameAs` property can be used.

If the needed `owl:sameAs` connections are available, using this LOD technology can, for example, shorten the execution time of queries. Listings 2.11 and 2.12 show the difference between a query without (Listing 2.11) and a query with usage of the LOD technique (Listing 2.12). Not only the query length is shortened (from 20 to 15 lines), but also the execution time is shortened drastically, from about 9 minutes to about 19 seconds.

To allow other people to query ones data, one has to set up a SPARQL endpoint, which handles the queries. The Mondial SPARQL endpoint, for example, is reachable under `http://www.semwebtech.org/mondial/10/sparql`. In this case, the URL can be used in any case to query data. No matter if one wants to use a web browser or a `SERVICE` clause from an other SPARQL endpoint. This is not always the case, as the endpoint of Wikidata shows. Here the web interface of the browser endpoint is available under `https://query.wikidata.org/`, while the URL to query data using the `SERVICE` clause of an other endpoint is `https://query.wikidata.org/sparql`. For Insee holds the same as for Mondial; the SPARQL endpoint as well as the web interface are available under the same URL `https://rdf.insee.fr/sparql`.

To set up a SPARQL endpoint and to access its data, there are many different ways. The setup, for example, can be done with the Jena-API, while the Raptor RDF parsing and serializing utility (rapper) can be used to access the SPARQL endpoint's data set.

Listing 2.11: Example query to ask for all mountains in Wikidata that are also in Mondial and where the elevation in both data sources is not equal. In this case, no `owl:sameAs` reference is used and, therefore, all Wikidata-Mountains have to be searched and returned by the Wikidata SPARQL endpoint. The execution time is about 9 minutes.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6 SELECT DISTINCT ?mondialMountain ?wikidataMountain ?mondialElevation ?wikidataElevation
7 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8 WHERE {
9     SERVICE <https://query.wikidata.org/sparql>{
10         ?wikidataMountain wdt:P31/wdt:P279* wd:Q8502 .
11         ?wikidataMountain wdt:P2044 ?wikidataElevation .
12         ?wikidataMountain rdfs:label ?label .
13         FILTER (LANGMATCHES (LANG(?label), "en" ))
14         BIND (STR(?label) AS ?wikidataName)
15     }
16     ?mondialMountain a mon:Mountain.
17     ?mondialMountain mon:name ?mondialName.
18     ?mondialMountain mon:elevation ?mondialElevation.
19     FILTER(?mondialName = ?wikidataName && ?mondialElevation != ?wikidataElevation &&
20         ↪ (?mondialElevation > 0.9 * ?wikidataElevation) && (?mondialElevation < 1.1 *
21         ↪ ?wikidataElevation))

```

Listing 2.12: Example query to ask for all mountains in Wikidata that are also in Mondial and where the elevation of both mountains is not equal. In this case, the LOD technique `owl:sameAs` is used and the execution time is reduced to 19 seconds.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
3 PREFIX owl: <http://www.w3.org/2002/07/owl#>
4
5 SELECT DISTINCT ?mondialMountain ?wikidataMountain ?mondialElevation ?wikidataElevation
6 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
7 WHERE{
8     ?mondialMountain a mon:Mountain.
9     ?mondialMountain owl:sameAs ?wikidataMountain.
10    ?mondialMountain mon:elevation ?mondialElevation
11    SERVICE<https://query.wikidata.org/sparql>{
12        ?wikidataMountain wdt:P2044 ?wikidataElevation.
13    }
14    FILTER (?mondialElevation != ?wikidataElevation)
15 }

```

2.6 Used Tools

2.6.1 The Jena-API

The Jena-API is a Java framework designed for Web of Data and LOD applications. It is published by [Fou] and nowadays available in version 3.10. Even though, the Jena-API can be used for a lot more than that, this thesis focuses on its ARQ package (Jena-ARQ) to process SPARQL queries stated against the Mondial data set. When downloading the precompiled version of the Jena-API, it comes together with some binaries, executable by the command line which makes it possible to state SPARQL queries against offline data sets. The Jena-API is licensed under the *Apache License, Version 2.0*, but available for free and open source, which makes it possible to review and optimize its source code.

The Jena-API is, for example, used to create the SemWebJena-Tool.

2.6.2 The SemWebJena-Tool

The SemWebJena-Tool is developed by the DBIS group of the Georg-August-Universität Göttingen and is based on the Jena-API version 2.10.0. It mainly consists of the Jena-API, but with an additional wrapper to add a powerful reasoner, like the openllet reasoner (<https://github.com/Galigator/openllet>), to the Jena-API. Additionally a command line tool is added to the SemWebJena-Tool, in order to make the Jena-API in combination with the openllet reasoner accessible from the command line.

Another way to access RDF data is the rapper tool.

2.6.3 rapper

The Raptor RDF parsing and serializing utility (rapper) tool is a command line tool published by [Bec14] and nowadays available in version 2.0. rapper can be used to access RDF data sets and thereby ask for results in a different way than web browsers do.

To transfer RDF data over the World Wide Web the W3C developed a syntax, similar to the Extensible Markup Language (XML) syntax, called RDF/XML. The first version of RDF/XML was released on September 6, 2001. Nowadays, it is available in version 1.1, which was published on February 25, 2014 [GS14]. In RDF/XML each resource is described by an `rdf:Description` element, like in Listing 2.13. To represent blank nodes in RDF/XML the `rdf:resource` parameter of the `rdf:Description` element is either omitted or replaced by a `rdf:nodeID` parameter (Listing 2.13 Lines 6 - 17).

Listing 2.13: Example for the RDF/XML syntax. Lines 2 to 4 show the definition of a Mondial-Neighbor-Element, which describes the neighborhood of "Poland" and "Germany". In lines 6 to 17 the description of the Mondial-Border-Element, describing the former neighborhood as a blank node, can be seen.

```

1 <rdf:RDF>
2   <rdf:Description rdf:about="http://www.semwebtech.org/mondial/10/countries/PL/">
3     <mon:neighbor rdf:resource="http://www.semwebtech.org/mondial/10/countries/D/" />
4   </rdf:Description>
5
6   <rdf:Description rdf:nodeID="genid100">
7     <rdf:type rdf:resource="http://www.semwebtech.org/mondial/10/meta#Border" />
8   </rdf:Description>
9   <rdf:Description rdf:nodeID="genid100">
10    <mon:isBorderOf rdf:resource="http://www.semwebtech.org/mondial/10/countries/PL/" />
11  </rdf:Description>
12  <rdf:Description rdf:nodeID="genid100">
13    <mon:isBorderOf rdf:resource="http://www.semwebtech.org/mondial/10/countries/D/" />
14  </rdf:Description>
15  <rdf:Description rdf:nodeID="genid100">
16    <mon:length rdf:datatype="xsd:integer">456</mon:length>
17  </rdf:Description>
18 </rdf:RDF>

```

2.6.4 cURL

The tool `cURL` is a command line tool for Linux operating systems, which originally was designed to download files from servers without user interaction. Therefore, it can use multiple different protocols, like HTTP, but it also can specify its own headers. With this tool it is possible to query a SPARQL endpoint automatically via the command line, but to define in which data format the endpoint should respond. The command `curl -header 'Accept: text/html,application/xhtml+xml' http://www.semwebtech.org/mondial/10/sparql?query=select+distinct+?X+where+%7B?Y+a+?X%7D`, for example, will ask the Mondial SPARQL endpoint for all instances within the Mondial data set, but the result should be returned as a HTML document.

Chapter 3

The Mondial LOD Service

In order to set up a LOD endpoint and, therefore, to implement a web service which is part of LOD, it first was necessary to analyze the requirements of such a service. To do so, the Wikidata and the Insee endpoints have been analyzed with respect to different aspects. For one thing, these endpoints were analyzed in which way they respond when meta data are accessed (Section 3.4), and, for another thing, how they handle blank nodes (Section 3.6). In addition to this, this chapter will explain how both of these aspects were solved for the Mondial SPARQL endpoint and will also give an overview over other problems, which came up while implementing the Mondial web service (Section 3.7). In the end of this chapter, a short evaluation (Section 3.8) of the developed SPARQL endpoint will be given.

To start with, there are multiple different ways in which a web service can be accessed.

3.1 Use Cases of Web Services

When setting up a web service, one first has to ask, in which way the future user will access the data, and for LOD services, there are three main ways to do so (Table 3.1).

Table 3.1: Overview of the ways to access an LOD endpoint

| Access via | Way of access |
|-----------------|--------------------------------------|
| Web Browser | direct acces via URI |
| | access via web interface |
| | query data via URL |
| RDF application | access of RDF data via URI |
| SPARQL query | access the SPARQL endpoint via query |

The first way is, to make use of a web browser. With this browser, the user accesses the data of the web service either by entering a URI in the address line of the browser, by using the LOD endpoint

of the desired data set, or by entering a query within the address line of the browser. In all cases, the browser sends a `text/html` accept header to the web service, indicating that results should be formatted using HTML. The web service has to prepare its result set in such a way, that it satisfies the desired format.

The second way to access a data set of an LOD service is to make use of tools like `raper` or `cURL`. These tools usually await a different data format for their result set, than web browsers do. The `raper` tool, for example, awaits its data in RDF/XML syntax and, therefore, sends an `application/rdf+xml` accept header to the web service. Again, the web service has to respond in the desired data format, because otherwise the tool will return false data, if any data at all.

The `cURL` tool is a special case, because with it, the user can decide which accept header should be sent. Therefore, this tool is perfectly suitable to test an LOD endpoint and the ways it returns data.

The last way to access an LOD service is by stating a SPARQL query. In this case, the accept header contains the `application/sparql-results+xml` phrase, which indicates, that the requester wants a set of SPARQL results formatted using XML. In difference to the other two cases, the `application/sparql-results+xml` accept header does not want its data in triple format, but as variable bindings, where for each result of the result set, an own binding is returned.

Using one of these ways, each user is able to query data and meta data of, for example, the Mondial LOD web service.

3.2 The Mondial LOD Web Service

The Mondial LOD web service is an LOD endpoint. The endpoint is configured in such a way, that it is able to respond correctly to all three use cases mentioned above. Thus, it is accessible via three different ways, namely via web browser, via access RDF tools, and via SPARQL queries, each having different methods to access the data (Table 3.1).

To get access to Mondial's data by using a web browser, there are three different possibilities. For one thing, there is the direct URI access, where the user enters the desired URI in the address line of the web browser, for another thing, there is the Mondial LOD endpoint web interface, and for a third thing, the user can state queries via the web browsers address line.

Whenever data are accessed directly by their URI, the Mondial web service will return a HTML page containing all information about the URI. Therefore, three different SPARQL queries are stated against the Mondial data set, one asking for all results where the desired URI is the subject, one where it is the predicate, and one where it is the object. All this information is collected and presented to the user on a single HTML page, as shown in Figure 3.1. To represent this information, a table structure, copying the triple structure of RDF, has been chosen, where the first column contains the subject, the second column contains the predicate, and the third column contains the object. If the cell entry of the table is another URI or a blank node, the entry is clickable and leads

<http://www.semwebtech.org/mondial/10/continents/Europe/> is subject in 4 entries.

URLs enclosed in <...> are true resource identifiers, while those in [...] are actually blank nodes that get fake URLs just for HTML browsing and for accessing this single URL via HTTP GET.

| S | P | O |
|-----------------------------------------------------------|---------------------------------------------------|-------------------------------------------------------|
| <http://www.semwebtech.org/mondial/10/continents/Europe/> | <http://www.w3.org/2002/07/owl#sameAs> | <http://www.wikidata.org/entity/Q46> |
| <http://www.semwebtech.org/mondial/10/continents/Europe/> | <http://www.semwebtech.org/mondial/10/meta#area> | 9938000 |
| <http://www.semwebtech.org/mondial/10/continents/Europe/> | <http://www.semwebtech.org/mondial/10/meta#name> | Europe |
| <http://www.semwebtech.org/mondial/10/continents/Europe/> | <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> | <http://www.semwebtech.org/mondial/10/meta#Continent> |

<http://www.semwebtech.org/mondial/10/continents/Europe/> is object in 108 entries.

URLs enclosed in <...> are true resource identifiers, while those in [...] are actually blank nodes that get fake URLs just for HTML browsing and for accessing this single URL via HTTP GET.

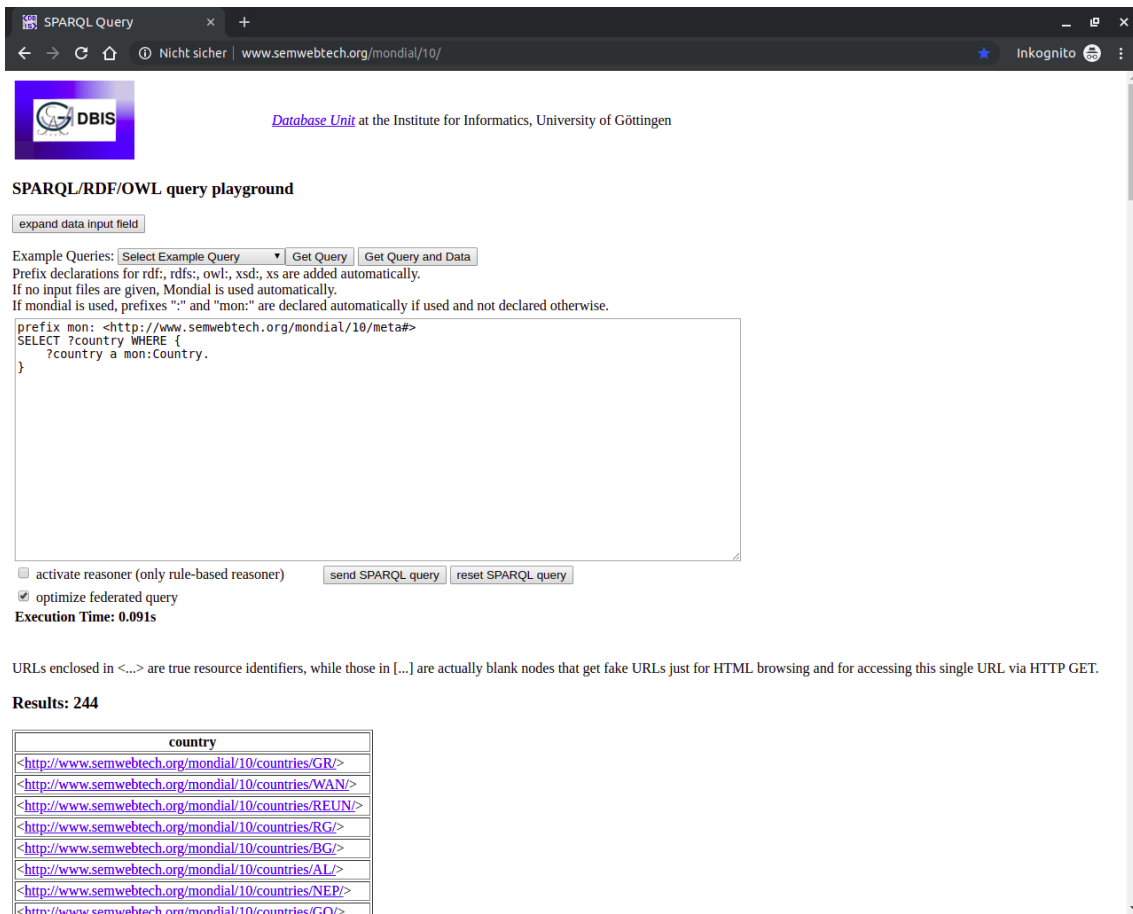
| S | P | O |
|----------------------------------------------------------------------------------|-----------------------------------------------------------|------------------------------|
| <http://www.semwebtech.org/mondial/10/countries/N/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_BG] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/UA/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/BG/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_GBZ] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_LV] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_FARX] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/SVAX/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_FL] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_E] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_GBG] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/MD/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/DK/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_B] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_UA] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_R] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_SRB] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/KAZ/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/E/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/A/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/S/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/GBG/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/GB/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/R/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| <http://www.semwebtech.org/mondial/10/countries/NL/> | <http://www.semwebtech.org/mondial/10/meta#encompassed> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_RSM] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |
| [http://www.semwebtech.org/mondial/10/Encompassed_Continent_Europe_Country_A] | <http://www.semwebtech.org/mondial/10/meta#encompassedBy> | <http://www.semwebtech.org/m |

Figure 3.1: Overview of the Mondial LOD endpoint, when accessed via URI from a HTML client (Web Browser). In this case, the Mondial-Continent "Europe" is shown.

to the corresponding URI. A special behavior occurs, if the clicked URI is not part of the Mondial ontology. In this case, the responsibility of resolving the URI lays with its owner and, therefore, the resolving behavior can not be predicted.

Another way to access Mondial's data via the web browser is to make use of the Mondial SPARQL web interface, available at <http://www.semwebtech.org/mondial/10/>. This web interface can be used to get in touch with SPARQL by stating queries against, for example, the Mondial RDF data set. Therefore, the user can either choose one of the example queries available in the drop down menu of the interface, or by entering a query into the available text field, as shown in Figure 3.2.

Again, the response of the Mondial web service is a web page containing a table structure, but this time the table has no triple structure. For each variable used within the SELECT clause of the query, an own column within the table is created and the corresponding value is shown. If



The screenshot shows a web browser window titled "SPARQL Query" at the URL "www.semwebtech.org/mondial/10/". The page features the DBIS logo and the text "Database Unit at the Institute for Informatics, University of Göttingen". Below this is the "SPARQL/RDF/OWL query playground" section. It includes an "expand data input field" button, a dropdown menu for "Example Queries", and buttons for "Get Query" and "Get Query and Data". A text area contains the following SPARQL query:

```
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
SELECT ?country WHERE {
  ?country a mon:Country.
}
```

Below the query area are checkboxes for "activate reasoner (only rule-based reasoner)" and "optimize federated query", along with "send SPARQL query" and "reset SPARQL query" buttons. The "Execution Time: 0.091s" is displayed. A note explains that URLs in <...> are true resource identifiers, while those in [...] are blank nodes. The results section shows "Results: 244" and a table of country URIs:

| country |
|--------------------------------------------------------|
| <http://www.semwebtech.org/mondial/10/countries/GR/> |
| <http://www.semwebtech.org/mondial/10/countries/WAN/> |
| <http://www.semwebtech.org/mondial/10/countries/REUN/> |
| <http://www.semwebtech.org/mondial/10/countries/RG/> |
| <http://www.semwebtech.org/mondial/10/countries/BG/> |
| <http://www.semwebtech.org/mondial/10/countries/AL/> |
| <http://www.semwebtech.org/mondial/10/countries/NEP/> |
| <http://www.semwebtech.org/mondial/10/countries/GO/> |

Figure 3.2: Overview over the Mondial SPARQL web interface. All instances of the Mondial-Country class are queried and printed below the text field.

the value is resolvable (URIs and blank nodes), the entry is clickable and the user is able to surf through Mondial's data by clicking on the corresponding link.

A third way to use a web browser to access Mondial's data is to state the query directly within the address line of the browser. Therefore, the SPARQL endpoint at <http://www.semwebtech.org/mondial/10/sparql> has to be accessed in combination with the `&query=` parameter. After this parameter, a regular SPARQL query, encoded with percentage encoding¹, can be stated. This encoding is necessary, because URLs are sensitive against some unicode symbols, for example #, {, and [, which can be avoided by using the percentage encoding.

The result of such a query against the Mondial web service, for example [http://www.semwebtech.org/mondial/10/sparql?query=select%20distinct%20C%](http://www.semwebtech.org/mondial/10/sparql?query=select%20distinct%20C%20)

¹A good encoder is available at <https://www.branah.com/unicode-converter>.

20where%20%7B?C%20a%20%3Chttp://www.semwebtech.org/mondial/10/meta%23Country%3E%7D, is a HTML page similar to the one when using the SPARQL web interface.

With command line tools like *raper*, it is not possible to state an own SPARQL query against an endpoint, but only to access data via URIs. When accessed via such a tool, the Mondial web service responds in the usual way. All triples containing information about the queried URI are returned using the RDF/XML format.

The last way to access Mondial's data is to make use of the SPARQL endpoint. Therefore, the Mondial SPARQL endpoint at <http://www.semwebtech.org/mondial/10/sparql> can be queried by every other SPARQL endpoint allowing federated queries or by an offline query using a SPARQL command line tool. Whenever such a query reaches the Mondial SPARQL endpoint, it returns a list of variable bindings in XML format.

3.3 Integrating the Mondial Data Set into the LOD-Cloud

In order to become part of the LOD-Cloud it was necessary to transform the Mondial data set into LOD. With the Mondial LOD endpoint already implemented as part of this thesis, only the last step described in Section 2.5, the step offering the fifth star of the LOD rating system had to be done. Therefore, links to other data sets had to be created and integrated into the Mondial data set. The decision was made, to keep all these links in a separate file, available at <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial-sameas.n3>, in order to keep an overview over the links and, if there should be any adjustments necessary, to update a relatively small file only, instead of the whole data set. At this moment, the file contains links to the Wikidata data set only, because it is a huge data set and should contain information about most entities, also stored in Mondial.

Due to the fact, that not every single link should have been created by hand, multiple different SPARQL queries were created and executed. All of these queries can be found in Appendix B. In order to do so, the already existing SemWebJena-Tool was updated to make use of the Jena-API in version 3.9.0. The Jena-API version, the SemWebJena-Tool was originally based on, was version 2.10.0, which did not yet support path expressions within SPARQL queries. A path expression can be used, to query for instances of all subclasses of a class like shown in Listing 3.1. This feature was necessary, because Wikidata has a much more detailed structure of its entries than Mondial has and everything listed as a Mondial-Country is not necessarily stored as a Wikidata-Country, but maybe as a Wikidata-Sovereign State, which is a subclass of Wikidata-Country.

Listing 3.1: Example query to return all instances of all subclasses of the Mondial-Place class. Line 5 makes use of SPARQL's path expressions. The / indicates, that in any case, a property *a* is followed by zero or more edges labeled as `rdfs:subClassOf` to return instances of these classes, while the * indicates that it does not matter how many `rdfs:subClassOf` properties are used to define the instance. It means, that the query, among others, will return all instances of Mondial-Place, as well as all instances of Mondial-Volcano, which is a subclass of Mondial-Mountain only and Mondial-Mountain itself is a subclass of Mondial-Place.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 SELECT ?instance ?class
4 WHERE {
5   ?instance a/rdfs:subClassOf* mon:Place .
6   ?instance a ?class .
7 }

```

By using these path expressions, the execution time for the queries could be reduced remarkably, for some queries from more than four days to less than an hour. Nevertheless, to execute these queries, a virtual cloud server with 8 processing cores of 2 GHz each and 16 GB RAM was used.

The automated creation of the connections, as well as the missing of a default identifier for entities in RDF, lead to the fact that some of the links within the Mondial-SameAs file were faulty or incomplete. In Mondial, for example, only one continent "America" is stored, while Wikidata differs between "North America" and "South America". Due to the fact, that the continent is used to identify matching countries, the matches of the Mondial-Continent "America" to the two Wikidata-Continents "North America" and "South America" were added by hand. Other errors, however, like the faulty match of Mondial's "North Atlantic Treaty Organization" to Wikidata's "National Association of Theatre Owners", due to the same abbreviation "NATO", were not removed, because they do not interfere with the correct execution of queries on the Mondial LOD servlet.

3.4 Technical Issues

When experimenting with different RDF data sets and SPARQL endpoints one recognizes, that there are many URIs within these data sets, which contain the hash symbol #. This symbol might lead to some unwanted behavior, when an RDF data set is examined via URIs and the underlying SPARQL endpoint is not configured properly. The reason therefore is, that # is a reserved character for web browsers, when it is used within URLs. It describes an anchor, like some special headline, which is defined by the queried web site. The symbol itself and everything after it is not transmitted to the SPARQL endpoint and, therefore, the URI `http://www.semwebtech.org/mondial/10/meta#Country` will return an empty result. To avoid the problem with the # symbol within URLs the Insee endpoint, for example, uses the percentage encoding, where

each non-alphanumerical character is escaped by a corresponding percent sequence. A space, for example, would translate into %20 and the # symbol into %23, resulting in a URL that looks like `http://www.semwebtech.org/mondial/10/meta%23Country`.

3.5 The Meta Data Access in Mondial's SPARQL Endpoint

As described above, the Insee SPARQL endpoint is implemented in such a way that symbols in URIs are replaced by percentage encoding. This solution was chosen for the Mondial URIs as well, because using this encoding method, the queried URIs still keep their human readable format, besides that the # symbol is replaced by %23. If one wants to query the Mondial-Country class, for example, the URL to query changes from `http://www.semwebtech.org/mondial/10/meta#Country` to `http://www.semwebtech.org/mondial/10/meta%23Country`. In addition to this, the already existing URIs of the Mondial data set had not to be changed.

A special case occurs, when a meta data query asks for the RDF/XML return type, instead of the HTML type. This indicates, that the data are queried by an application, like the rapper tool, and, therefore, that more data than requested can be sent back in order to give the enquirer more information, probably needed in the future. In order to do so, queries for meta data using the `application/rdf+xml` accept header get the whole Mondial meta ontology in return.

3.6 The Resolving of Blank Nodes

Blank nodes are an essential part of RDF and, therefore, worth a deeper analysis of how they are used within RDF datasets. This analysis additionally gives an impression of how the blank nodes within the Mondial data set can be resolved.

3.6.1 The Problem with Blank Node Resolving

The main problem when resolving blank nodes is, that they have no URI. This fact makes it hard to query for blank nodes and to analyze how they are used within RDF data sets without accessing the data files themselves. Blank nodes often contain important or interesting information for the user, which will be lost, if the blank node is not resolved. Using SPARQL's ability to check for a nodes characteristics, the query `SELECT * WHERE{?X ?P ?O. FILTER(isBlank(?X) || isBlank(?O))}` lists all blank nodes of a SPARQL endpoint so that they can be analyzed.

3.6.2 Wikidata's Way to Resolve Blank Nodes

The first SPARQL endpoint to analyze for its usage and resolving of blank nodes was the Wikidata endpoint. Due to the fact, that the Wikidata SPARQL endpoint has a query timeout of one minute and the huge amount of data stored within Wikidata, the query to find out about blank nodes `SELECT * WHERE{?X ?P ?O. FILTER(isBlank(?X) || isBlank(?O))}` resulted in an timeout error. Therefore, the analysis had to be done in a different way. Another point that complicated the examination of the blank nodes was the manner of Wikidata of redefining everything regarding RDF and its ontologies. With this in mind, the idea came up, that Wikidata might also redefine the usage of blank nodes, which, after a short research, proofed to be true. Instead of usual blank nodes as defined by the W3C, Wikidata uses so-called "Statement Nodes". Each entity in Wikidata points to several of these nodes using special properties which were created for this purpose. These statement nodes themselves either point to the desired value, a reference node, or a value node. A schema of this interconnection of node types can be seen in Figure 3.3.

3.6.3 How does Insee Resolve Blank Nodes?

An other way of resolving blank nodes is used by Insee. When stating the aforementioned query `SELECT * WHERE{?X ?P ?O. FILTER(isBlank(?X) || isBlank(?O))}` against the Insee data set, the result set contains multiple string values as object entries, looking like this `_:node1chfpgbhg7x3280` (Figure 3.4). `_:` is a common indicator, that the following string is a blank node label, and, therefore, that the returned node is a blank node. This object value, also appears multiple times as subject value within the same results set. At this points all information stored within the blank node are shown and resolved.

3.6.4 Blank Node Resolving in the Mondial SPARQL Endpoint

Even though, RDF blank nodes are usually not publicly accessible, the decision was made that the blank nodes stored in Mondial should be accessible and even queryable by anybody, since the Mondial blank nodes contain many information which might be interesting for users. To do so, two different approaches were implemented, one for blank node access via web browsers and one via tools like rapper.

For blank node resolving via rapper, an approach similar to the one used by the Insee SPARQL endpoint was followed and, therefore, the blank nodes are already resolved when the subject containing the blank node is queried (Figure 3.5). Even a syntax, similar to the Insee syntax has been chosen for this purpose. Each blank node is assigned an ID consisting of `_:`, followed by a blank node label, which, in this case, is simply chosen as `genid` and a number. For each query, the number starts by 1 and is increased for each new blank node. This fact also points out, why

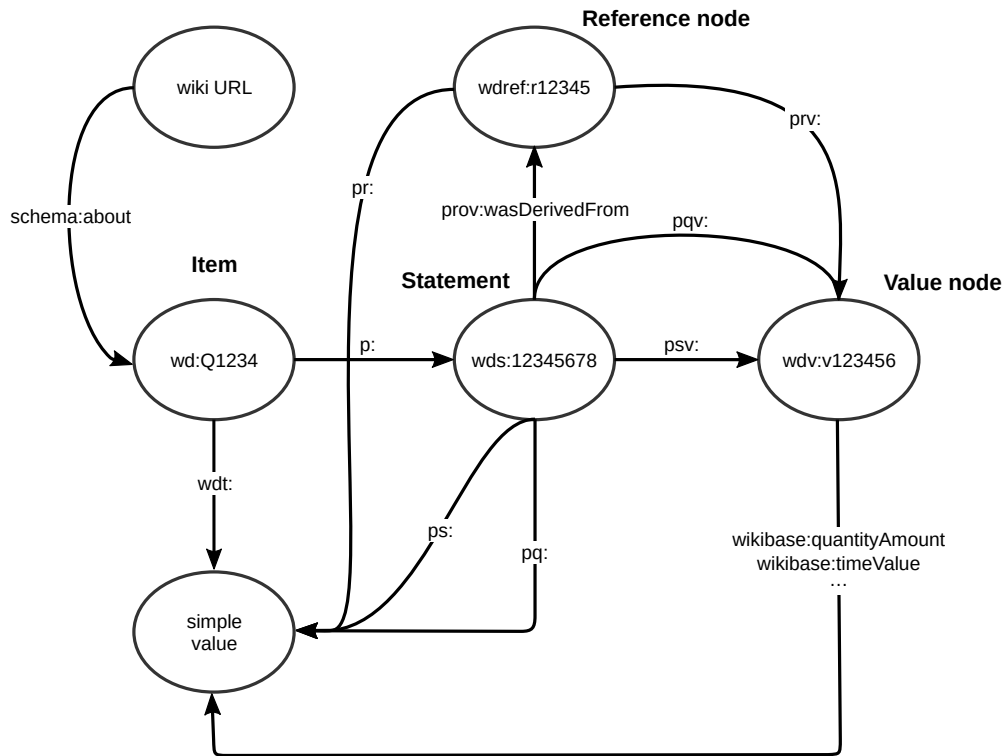


Figure 3.3: Description, how Wikidata avoids using usual blank nodes. Each normal node is linked to multiple statement nodes by properties, especially created for this purpose. These statement nodes are connected either to a reference node, a value node, or the simple value. The image was taken from https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format#/media/File:Rdf_mapping-vector.svg. All prefixes used in this figure are described at https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format#Prefixes_used.

The screenshot shows a web browser window with the URL `https://rdf.insee.fr/sparql`. The page header includes the Datalift and Insee logos, along with the text "Institut national de la statistique et des études économiques" and "Mesurer pour comprendre". A "SPARQL Endpoint" button is visible in the top right.

The main content area is titled "SPARQL Query" and contains a query editor with the following text:

```

Response format: HTML RDF/XML N3/Turtle NTriples THG TRIX CSV
Query(Aide) :
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX lgeo:<http://rdf.insee.fr/def/lgeo#>

SELECT * WHERE{?X ?P ?O. FILTER(isBlank(?X) || isBlank(?O))}

```

Below the query editor, there are several predefined queries: "1 - Region by its name", "2 - Population of a municipality", "3 - List of classifications", "4 - NAF rév.2 item by its code", "5 - CSP 2003 item by its label", and "6 - List of concepts". A "Max. results" field is set to "1000000" and a "Display literal types" checkbox is present. An "Execute query" button is located at the bottom right of the query area.

The results table shows a search for the blank node `_:node1chfbh7x3280` in all columns. The table has three columns: "X", "P", and "O". The results are as follows:

| X | P | O |
|-----------------------------------------------------|------------------------------------------------------------------|-----------------------------------------------------------------|
| <code>_:node1chfbh7x3280</code> | <code>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</code> | <code>http://rdf.insee.fr/def/lgeo#Creation</code> |
| <code>_:node1chfbh7x3280</code> | <code>http://www.w3.org/2000/01/rdf-schema#comment</code> | <code>"Zévaos est rattaché au département (Anc. 20358)."</code> |
| <code>_:node1chfbh7x3280</code> | <code>http://rdf.insee.fr/def/lgeo#mouvementTerritoire</code> | <code>http://rdf.insee.fr/def/lgeo#commune/2A358</code> |
| <code>_:node1chfbh7x3280</code> | <code>http://rdf.insee.fr/def/lgeo#mouvementPropriete</code> | <code>http://rdf.insee.fr/def/lgeo#subdivisionDe</code> |
| <code>http://id.insee.fr/geo/mouvements/1712</code> | <code>http://rdf.insee.fr/def/lgeo#mouvementBase</code> | <code>_:node1chfbh7x3280</code> |
| <code>_:node1chfbh7x3280</code> | <code>http://rdf.insee.fr/def/lgeo#referenceMouvementBase</code> | <code>http://id.insee.fr/geo/mouvements/1712</code> |
| <code>_:node1chfbh7x3280</code> | <code>http://rdf.insee.fr/def/lgeo#valeurAreesCreation</code> | <code>http://id.insee.fr/geo/arrondissement/2A1</code> |

At the bottom of the page, there is a link: "More information about Datalift at <http://www.datalift.org>".

Figure 3.4: Snapshot from the Insee SPARQL endpoint. In order to increase readability, the results have been filtered for the blank node `_:node1chfbh7x3280`.

the blank nodes are not accessible by rapper directly, because `_:genid1` might point to a blank node describing the population count of Göttingen in 1987 in one query, while in the next it may describe the membership of Ireland in the UN. The Jena-API internally assigns other identifiers to the blank nodes, which change every time the Mondial LOD servlet is restarted. This, together with the reuse of the `genid` identifiers point out, that the blank nodes of the Mondial LOD servlet are not queryable via the `SERVICE` clause of SPARQL.

To make blank nodes accessible and queryable for users using web browsers a completely different approach was followed. For this case, each blank node was assigned a pseudo URI consisting of the blank node type and two different identifiers, which uniquely define the blank node. Therefore, it is important to know, that the type and the identifiers are separated by `__` (double underscore), while each identifier contains its Mondial class and its corresponding ID, each separated by `_` (single underscore). All blank node types used in Mondial, together with an example for a corresponding blank node URI are listed below.

- Population Count (Country) : `PopulationCount__Country_D__1997`

```

$ rapper http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/Göttinge
n/
rapper: Parsing URI http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/Göttingen/ with parser rdf
xml
rapper: Serializing with serializer ntriples
_:genid1 <http://www.semwebtech.org/mondial/10/meta#value> "114698"^^<xsd:integer> .
_:genid1 <http://www.semwebtech.org/mondial/10/meta#year> "1987"^^<xsd:integer> .
_:genid1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semwebtech.org/mondial/10/meta#PopulationCount> .
<http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> <http://www.semwebtech.org/m
ondial/10/meta#hadPopulation> _:genid1 .
<http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> <http://www.semwebtech.org/m
ondial/10/meta#longitude> "9.94"^^<xsd:decimal> .
<http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> <http://www.w3.org/2002/07/o
wl#sameAs> <http://www.wikidata.org/entity/Q3033> .
<http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> <http://www.semwebtech.org/m
ondial/10/meta#population> "115843"^^<xsd:integer> .
<http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> <http://www.semwebtech.org/m
ondial/10/meta#latitude> "51.53"^^<xsd:decimal> .
_:genid2 <http://www.semwebtech.org/mondial/10/meta#value> "115843"^^<xsd:integer> .
_:genid2 <http://www.semwebtech.org/mondial/10/meta#year> "2011"^^<xsd:integer> .
_:genid2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semwebtech.org/mondial/10/meta#PopulationCount> .
<http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> <http://www.semwebtech.org/m
ondial/10/meta#hadPopulation> _:genid2 .
<http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> <http://www.semwebtech.org/m
ondial/10/meta#locatedAt> <http://www.semwebtech.org/mondial/10/rivers/Leine/> .
<http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> <http://www.semwebtech.org/m
ondial/10/meta#elevation> "150"^^<xsd:integer> .
<http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> <http://www.w3.org/1999/02/2
2-rdf-syntax-ns#type> <http://www.semwebtech.org/mondial/10/meta#City> .
<http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> <http://www.semwebtech.org/m
ondial/10/meta#name> "G\u00F6ttingen" .
<http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/> <http://www.semwebtech.org/mondial/10/meta#hasCity
> <http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> .
<http://www.semwebtech.org/mondial/10/countries/D/> <http://www.semwebtech.org/mondial/10/meta#hasCity> <http://www.semwebtech
.org/mondial/10/countries/D/provinces/Niedersachsen/cities/G\u00F6ttingen/> .
rapper: Parsing returned 18 triples

```

Figure 3.5: Overview of the rapper tool. The yellow marking shows the resolving of a Mondial blank node, describing the population count of Göttingen in 1987.

- Population Count (City): `PopulationCount__City_NL_Noord-Holland_Amsterdam__2010`
- Membership: `Membership__Country_B__Organization_NATO`
- Border: `Border__Country_CDN__Country_USA`
- Encompassed: `Encompassed__Continent_America__Country_MEX`
- EthnicProportion: `EthnicProportion__EthnicGroup_European__Country_RA`
- BelievedBy: `BelievedBy__Religion_Roman+Catholic__Country_GBZ`
- SpokenBy: `SpokenBy__Language_French__Country_CH`

All of these URIs are created on the fly, meaning that each time when a resource containing a blank node is queried, these pseudo URIs are created again. This makes clear, that these URIs are not stored within the Mondial RDF data set.

The reason that this approach was not used for the resolving via rapper or the SPARQL `SERVICE` clause as well was for one thing that this is not the common use of blank nodes and for another thing that within a web browser it is possible to mark the pseudo URIs in a special way, to show that they are not real. The marking in HTML was done by not putting the pseudo URIs in `<>` (angle bracket) as usual, but in `[]` (square bracket) instead.

3.7 Problems with Mondial's URIs

While testing one of the first versions of the Mondial servlet, it became clear, that there is a problem regarding the blank node resolving (Section 3.6), especially when resolving population counts. The pseudo URIs for the blank nodes were created using the URI of the node described by the blank node in combination with the key attributes from the blank node itself, as described above. The reason for this behavior is, that the hierarchical structure of Mondial's cities and provinces is stored in one direction only. Each Mondial-Country knows its Mondial-Provinces, and each Mondial-Province its Mondial-Cities, but the Mondial-Cities do not know in which Mondial-Country or Province they are located. For a population count of the German capital "Berlin", for example, the corresponding blank node URI

```
[http://.../PopulationCount__City_D_Berlin_Berlin__1987]
```

was created by extracting the first identifier (`City_D_Berlin_Berlin`) from the described node representing Berlin, while the blank node type (`PopulationCount`) and the second identifier (`1987`) came from the blank node itself.

During the resolution of this blank node, the Mondial servlet queries its data set for a `Mondial-City`, whose Mondial-Name is "Berlin", and which is located in the Mondial-Province "Berlin" and the Mondial-Country "D".

This behavior worked perfectly, if the name of the city in the URI really matched the Mondial-Name of the city, which was not always the case. In the original data set, the URI of the City "New York" was `<http://.../cities/NewYork/>`, which resulted in a blank node URI like

```
[http://.../PopulationCount__City_USA_NewYork_NewYork__1990].
```

During the resolution of this blank node, the Mondial-City with the Mondial-Name "NewYork" was queried, which did not exist. Therefore, all URIs were modified in such a way, that, if the Mondial-Name of the node contains a space this space is represented in the URI by a + (plus) symbol. Thus, the URI for New York changed to `<http://.../cities/New+York/>`, while the corresponding blank node URI changed to `[http://.../PopulationCount__City_USA_New+York_New+York__1990]`.

A similar behavior was found, when the Mondial-Name of a subject contains a / (slash), like in "Australia/Oceania". In this case, the / was replaced by a \$ (dollar) symbol, leading to URIs like `<http://www.semwebtech.org/mondial/10/continents/Australia$Oceania>`.

3.8 Evaluation of the Mondial LOD Web Service

To evaluate the first part of this thesis, the Mondial LOD servlet at `http://www.semwebtech.org/mondial/10/` was analyzed in multiple different ways. For one thing, the HTML representation was tested, if there are any inconsistencies during the resolving of the URIs or if there are some broken links created by the servlet. Besides some minor bugs during the creation of the links to some URIs, a major problem was assumed for the encoding of some Mondial-Localnames, where the name of an entity is returned in the official language and the corresponding alphabet of the country where the entity is located. These bugs, however, could be removed and resulted in a nearly flawlessly working Mondial LOD servlet.

Additionally, the Mondial SPARQL endpoint was analyzed and thoroughly tested. Therefore, the example queries available at the endpoint were executed and their results verified. During this evaluation it became clear, that it is impossible to resolve blank nodes that were queried within a SERVICE clause and should be returned to the user. This is, because when a blank node is returned by the SERVICE clause, its internal representation within the Jena-API is returned. This representation, however, is unique for each single instance of the Mondial LOD servlet, leading to the fact, that the blank node ID returned by the SERVICE clause does not refer to the same blank node within the servlet instance which returns the results to the user. Therefore, a blank node URI, only containing the base URI (`[http://www.semwebtech.org/mondial/10/]`) is returned.

Another way to test the Mondial LOD servlet was, to access its RDF data, using the tools rapper and curl. This testing resulted in the fact, that this part of the servlet runs flawlessly.

Chapter 4

Evaluation of Distributed Queries

To understand the evaluation of distributed queries in the Jena-API, it first was necessary to understand the way, how the Jena-API evaluates queries at all. Therefore, it was necessary to dive into the evaluation path and into the Jena-API.

4.1 Analysis of Queries within the Jena-API

The first step performed by the Jena-API, whenever a SPARQL query is stated, is, to translate the query into a parse tree. Therefore, each keyword within the query is transformed into a single element object containing information about this query part. A `SERVICE` clause, for example, is transformed into a `ElementService`, which has, among others, information about the federated SPARQL endpoint. After this transformation, the parse tree is translated into an algebra tree, containing an operator for each element. These operators have information about their own operation and the operators that are connected with this operation. The `OpJoin` operator, for example, knows that it should join together its two child operators, called left-hand side and right-hand side. These operators additionally provide an iterator over their data. After the algebra tree is created, the Jena-API tries to optimize it by using its internal optimizer. This optimizer could, for example, reorganize filter operations in such a way, that the size of intermediate results sets decreases as soon as possible, to save execution time. After the optimizer is finished, the execution of the query starts.

One case where the internal optimizer of the Jena-API is used is, for example, the evaluation of the `SERVICE` clause.

4.2 Analysis of the SERVICE and the VALUES Keyword

The second major part of this master's thesis was, to analyze, and, if possible, to optimize, the evaluation of the SERVICE keyword within the Jena-API.

In our opinion, it would have been the best way to evaluate the part outside the SERVICE clause first, and afterwards send all the necessary information to the federated endpoint using the VALUES tag. To check, whether this is the case, the evaluation of the query in Listing 4.1 or rather its evaluation tree was analyzed. Within the evaluation tree (Listing 4.2) each SPARQL operator is listed, that is executed during the evaluation. Due to the integrated optimizer of the Jena-API, the original evaluation tree was modified in such a way, as that the JOIN operation in Line 2 of Listing 4.2 was replaced by a SEQUENCE operator (Listing 4.3, Line 2). A SEQUENCE operator is similar to a FOREACH-loop of many programming languages. In this case, it executes the SERVICE clause for each result of the BGP in Line 3 of Listing 4.3, therefore, once for each continent in Mondial.

Listing 4.1: A very simple query to analyze the evaluation strategy of the SERVICE clause of the Jena-API. It selects all Mondial-Continents, and asks the Mondial SPARQL endpoint within the SERVICE clause for the names of the continents.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT ?continent ?name
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5     ?continent a mon:Continent.
6     SERVICE <http://www.semwebtech.org/mondial/10/sparql>{
7         ?continent mon:name ?name.
8     }
9 }

```

Listing 4.2: The original algebra tree of the query in Listing 4.1. Each SPARQL operator and its parameters called during the evaluation of the query is listed. At this point, the Jena-API originally would have replaced the prefixes used within the query by their original URIs. This, however, has been reversed, in order to increase the readability.

```

1 (project (?continent ?name)
2   (join
3     (bgp (triple ?continent rdf:type mon:Continent ))
4     (service <http://www.semwebtech.org/mondial/10/sparql>
5       (bgp (triple ?continent mon:name ?name))
6     )
7   )
8 )

```

Listing 4.3: The evaluation tree for the query in Listing 4.1, after it has been optimized by the Jena-API. The JOIN operator of the original evaluation tree (Listing 4.2) has been replaced by a SEQUENCE operator. At this point, the Jena-API originally would have replaced the prefixes used within the query by their original URI. This, however, has been reversed, in order to increase the readability.

```

1 (project (?continent ?name)
2   (sequence
3     (bgp (triple ?continent rdf:type mon:Continent ))
4     (service <http://www.semwebtech.org/mondial/10/sparql>
5       (bgp (triple ?continent mon:name ?name))
6     )
7   )
8 )

```

Some experimenting with the VALUES clause and the Jena-API quickly showed, that the Jena-ARQ module is able to understand the VALUES clause and delivers correct results. It became clear, that the positioning of the clause within the query is very important for the evaluation. For example, if one wants to query for all countries in Mondial which are located in Europe or Africa, one can use either Listing 4.4 or Listing 4.5. Even though, the difference between these queries is minimal, its effect is huge. With the VALUES clause outside of the SERVICE clause (Listing 4.4) for each binding of the ?continent variable, all possible bindings for the ?continentName are tried. With the VALUES clause within the SERVICE clause (Listing 4.5) the number of queries per continent is reduced to one, namely the one that checks whether the continents name is either "Europe" or "Africa". Therefore the total number of connections to the federated endpoint is reduced by factor two and, resulting in a reduction of the execution time.

Listing 4.4: Query to return all countries of Mondial that are located in Europe or Africa. For this query, the VALUES clause is located outside of the SERVICE clause.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT ?countryName ?country ?continentName ?continent
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5   ?continent a mon:Continent.
6   SERVICE<http://www.semwebtech.org/mondial/10/sparql>{
7     ?continent mon:name ?continentName.
8     ?country a mon:Country; mon:encompassed ?continent; mon:name ?countryName.
9   }
10  VALUES ?continentName {"Europe" "Africa"}
11 }

```

Listing 4.5: This query returns all Mondial-Countries which are either in Europe or in Africa. This time, the VALUES clause is used within the SERVICE clause.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT ?countryName ?country ?continentName ?continent
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5     ?continent a mon:Continent.
6     SERVICE<http://www.semwebtech.org/mondial/10/sparql>{
7         ?continent mon:name ?continentName.
8         ?country a mon:Country; mon:encompassed ?continent; mon:name ?countryName.
9         VALUES ?continentName {"Europe" "Africa"}
10    }
11 }

```

Further experimenting with the Jena-API confirmed the difference between the MINUS and the FILTER NOT EXISTS expressions of SPARQL, already mentioned in Section 2.4. While the MINUS expression is translated into a single MINUS operator (Listing 4.6), the FILTER NOT EXISTS expression results in two different operators, one FILTER operator and one NOTEXISTS operator (Listing 4.7).

Listing 4.6: The evaluation tree for the query in Listing 2.6. It can be seen, that the MINUS clause of SPARQL is translated into a single MINUS operator (Line 3).

```

1 (distinct
2   (project (?organization)
3     (minus
4       (bgp (triple ?organization rdf:type mon:Organization))
5       (bgp (triple ?a ?b ?c))
6     )
7   )
8 )

```

Listing 4.7: The evaluation tree for the FILTER NOT EXISTS expression of Listing 2.7. The FILTER NOT EXISTS clause is translated into two different operators. One FILTER operator (Line 3) and one NOTEXISTS operator (Line 4).

```

1 (distinct
2   (project (?organization)
3     (filter
4       (notexists
5         (bgp (triple ?a ?b ?c))
6       )
7     (bgp (triple ?organization rdf:type mon:Organization))
8   )
9 )
10 )

```

4.3 Optimization of the SERVICE Clause Evaluation within the Jena-API

The main idea for the optimization of the evaluation of the SERVICE clause within the Jena-API was to automatically add a VALUES clause to it and, therefore, to make the evaluation more time efficient. As already shown in Section 4.2, the Jena-ARQ evaluates a SERVICE clause in such a way, that it transforms the SERVICE clause into a SEQUENCE clause and sends one SERVICE query for each variable binding resulting from the outer part of the original query. To change this behavior into the desired one, there are two different approaches possible. First, it would be possible to change the original query string into one containing a VALUES clause, and second, a change of the evaluation tree of the SERVICE clause within the Jena-API is possible.

The first approach was implemented in such a way, that it searched for a valid SERVICE clause within an query string and changed the query thusly, that the SERVICE clause afterwards contained a valid VALUES clause within the SERVICE clause. If, for example, one wanted to query for all Mondial-Continents, and ask the Mondial SPARQL endpoint for the Mondial-Names of these continents, the corresponding query would look like Listing 4.8. The changed query would contain a VALUES clause and is given in Listing 4.9. For this modification, however, the original query string had to be compiled by the Jena-API and, therefore, been translated into an execution plan. During this compilation, the SERVICE clause was automatically translated into a SEQUENCE clause and evaluated. After that, a modification of the query string would be possible, but pointless, because the interesting SERVICE clause has already been evaluated in the probably inefficient way. Therefore, the second approach, changing the execution tree of the Jena-API, was implemented.

Listing 4.8: Query to query all continents stored in Mondial using the SERVICE clause. For each Mondial-Continent an own query is sent to the Mondial SPARQL endpoint, in order to ask for its Mondial-Name. The sense of this query is questionable, because the Mondial LOD endpoint asks itself within the SERVICE clause which is necessary to test the evaluation of the SERVICE clause.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT ?continent ?name
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5     ?continent a mon:Continent.
6     SERVICE <http://www.semwebtech.org/mondial/10/sparql>{
7         ?continent mon:name ?name.
8     }
9 }

```

Listing 4.9: Modified query from Listing 4.8 to use the `VALUES` keyword. This time, only one SPARQL query is sent to the Mondial SPARQL endpoint, which asks for all Mondial-Names of all Mondial-Continents at once.

```

1 SELECT ?continent ?name
2 WHERE {
3   ?continent a :Continent.
4   SERVICE<http://www.semwebtech.org/mondial/10/sparql>{
5     ?continent :name ?name.
6     VALUES (?continent) {
7       (<http://www.semwebtech.org/mondial/10/continents/Australia$Oceania/>)
8       (<http://www.semwebtech.org/mondial/10/continents/Asia/>)
9       (<http://www.semwebtech.org/mondial/10/continents/America/>)
10      (<http://www.semwebtech.org/mondial/10/continents/Europe/>)
11      (<http://www.semwebtech.org/mondial/10/continents/Africa/>)
12    }
13  }
14 }

```

To modify the evaluation tree, it was necessary to dig into it and analyze in which way the Jena-ARQ evaluates its queries. A part of the execution process of the Jena-ARQ is shown in Figure 4.1 as a directed graph, starting with the `doPost` method which is called when a user clicks the "send SPARQL query" button of the Mondial SPARQL endpoint. Everything executed before this method are internal Java functions related to the used server environment and are not important for the evaluation of the query. Within the graph, each node represents a Java class that is called with the function, mentioned at the directed edges. Some edges are unlabeled, but result in a text, which means, that these methods are called within the same Java class where their edge originates. The first row of nodes describes classes implemented during this thesis, while every other class is part of the Jena-ARQ. The execution order of these methods is highly related to the position where the edge enters respectively leaves the corresponding node. Opposing methods are executed after each other, while a function call never turns around a corner within a class node. The graph results in an cyclic function call, where the "OpOPERATOR" node represents the different operators used in the query, for example, `OpProject` for a projection to the desired variables or `OpService` for a `SERVICE` node. If there is no operator left to evaluate, the execution terminates after visiting the last `OpOPERATOR` node. During the optimization process it became clear, that the translation of the `JOIN` node into a `SEQUENCE` node by the Jena-API has to be inhibited. Therefore, the blue part of the evaluation tree was bypassed and the plan creation of the `QueryEngineBase` resulted directly in its evaluation (dotted edge).

In order to modify the original `SERVICE` operator (Listing 4.10 Line 4) in such a way that it contains a valid `VALUES` clause, it was necessary to begin the modification in the operator above the `SERVICE` operator. Whenever a `SERVICE` clause is used within a SPARQL query which also contains a BGP asking for local bindings, the Jena-API translates these two operators into three

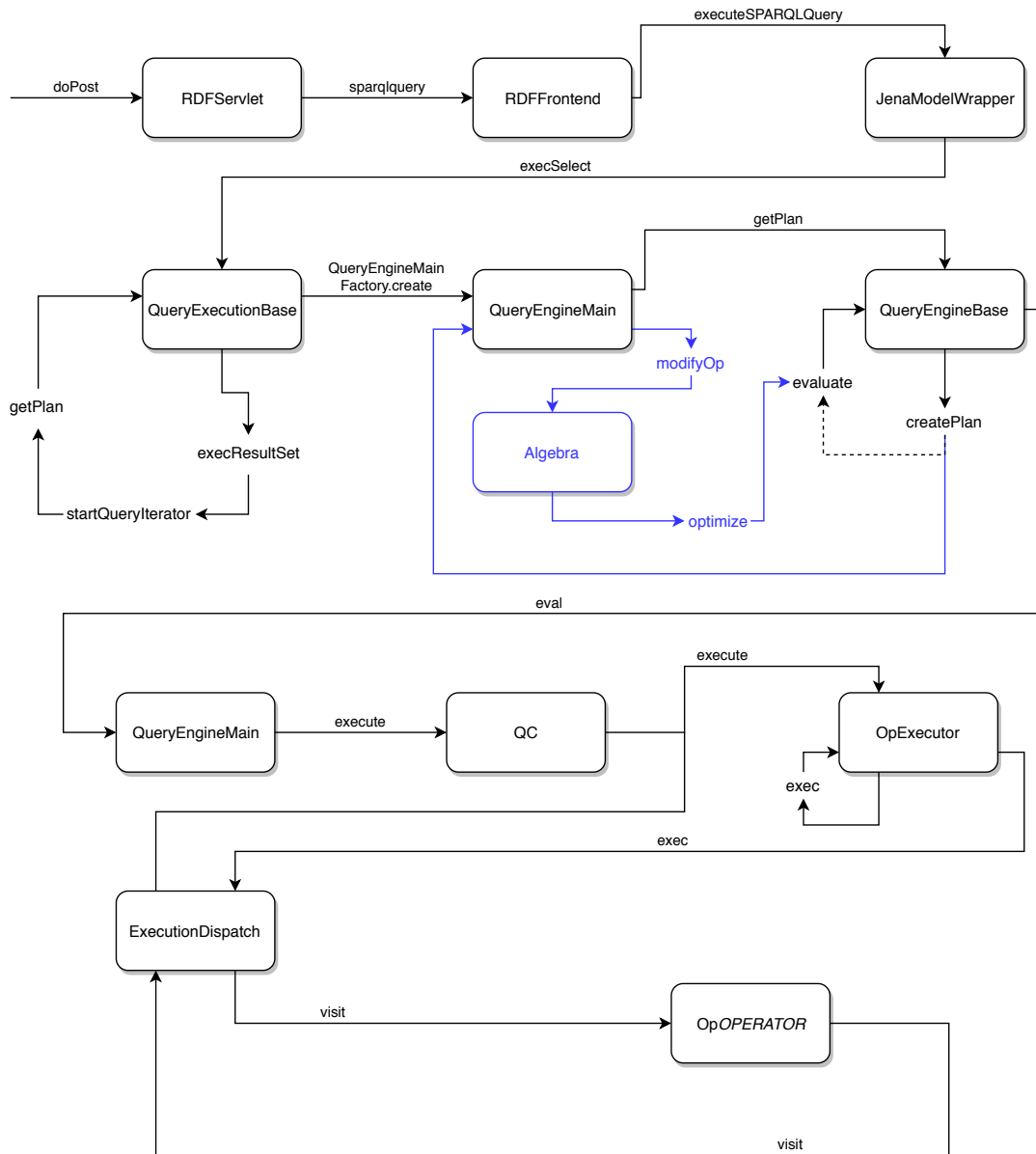


Figure 4.1: Overview over the Java methods used during the evaluation of a SPARQL query.

operators. One `BGP` operator (Listing 4.10 Line 3), one `SERVICE` operator (Listing 4.10 Line 4), and one `JOIN` operator (Listing 4.10 Line 2), joining the two other operators. This behavior made it possible to modify the `JOIN` operator of each of these queries. Therefore, an additional Java class, the `OpServiceJoin` class, was created.

The `OpServiceJoin` class (Listing 4.12) is an extension of the regular `OpJoin` class of the Jena-API and is used whenever there is a `JOIN` operator whose right-hand side operator is an instance of the `OpService` class. In such a case, the left-hand side operator is fully evaluated (Listing 4.11 Line 6) and previously checked for shared variables between itself and the `SERVICE` operator (Listing 4.11 Lines 4 and 5). This is necessary because it is reasonable to reduce the variables within the `VALUES` clause to the shared ones, because all additional information would not be used by the `SERVICE` clause but extend the execution time. Afterwards, all duplicate bindings of these shared variables are removed (Listing 4.11 Lines 9 to 24), because each variable binding is needed only once within the `SERVICE` clause and, therefore, within the `VALUES` clause. Now, a new `OpServiceJoin` instance is created, representing the new `SERVICE` clause within the query (Listing 4.11 Line 27). To do so, the query string, sent by the old `OpService` is reused, the `VALUES` clause added and then sent to the federated endpoint (Listing 4.12 Lines 9 to 47).

In contrast to the first approach explained in this section, the old `OpService` is never evaluated, but only created. As a result, only a single SPARQL `SERVICE` clause, consisting of a `OpServiceJoin`, is evaluated during the query execution. Afterwards, the results of the left-hand side operator of the `OpJoin` and the `OpServiceJoin` operator are joined (Listing 4.11 Line 29) and returned to the user (Listing 4.11 Line 30).

In addition to the changes within the `OpJoin` execution, the evaluations of the `OpMinus` and the `OpLeftOuterJoin` operators had to be modified analogously. This was necessary, because a `SERVICE` clause could be placed within a `MINUS` or an `OPTIONAL` clause, which would lead to the fact, that this `SERVICE` clauses would not be recognized and, therefore, not be optimized by the modifications of the `OpJoin` operator execution.

Listing 4.10: The original evaluation tree of the query in Listing 4.8. Each SPARQL operator and its parameters called during the evaluation of the query is listed.

```

1 (project (?c ?name)
2   (join
3     (bgp (triple ?c rdf:type mon:Continent ))
4     (service <http://www.semwebtech.org/mondial/10/sparql>
5       (bgp (triple ?c mon:name ?name))
6     )
7   )
8 )

```

Listing 4.11: Snippet of the `OpExecutor` class of the Jena-API. These lines show the modifications done to the `OpJoin`, `OpMinus`, and the `OpLeftOuterJoin` execution in order to optimize the evaluation of federated queries.

```

1  if (!(opJoin instanceof OpServiceJoin) && opJoin.getRight() instanceof OpService &&
    ↪ (!bla || RDFFrontend.getOptimize()) && ParseRDF.federatedOptimizer){
2  Op lhsOp = opJoin.getLeft() ;
3  OpService rhsOp = (OpService) opJoin.getRight() ;
4  Set<Var> commonVars = OpVars.visibleVars(lhsOp) ;
5  commonVars.retainAll(OpVars.mentionedVars(rhsOp)) ;
6  QueryIterator left = exec(lhsOp, input) ;
7  List<Binding> leftMat = all(left);
8  ArrayList<Binding> distinctMat = new ArrayList<Binding>();
9  for (Binding binding : leftMat){
10     ArrayList<Boolean> duplicate = new ArrayList<>();
11     boolean found = false;
12     for (Binding bindingAdded : distinctMat){
13         for (Var var : commonVars){
14             duplicate.add(bindingAdded.get(var).equals(binding.get(var)));
15         }
16         if(!duplicate.contains(false) && duplicate.size() != 0){
17             found = true;
18         }
19         duplicate.clear();
20     }
21     if (!found){
22         distinctMat.add(binding);
23     }
24 }
25 QueryIterator leftMatIter1 = new QueryIterPlainWrapper(distinctMat.iterator(),
    ↪ execCxt) ;
26 QueryIterator leftMatIter2 = new QueryIterPlainWrapper(distinctMat.iterator(),
    ↪ execCxt) ;
27 Op newOpService = OpServiceJoin.create(distinctMat, lhsOp, rhsOp, commonVars,
    ↪ execCxt); {
28 QueryIterator right = exec(newOpService, leftMatIter1) ;
29 QueryIterator qIter = Join.join(leftMatIter2, right, execCxt) ;
30 return qIter;
31 }

```

Listing 4.12: The two main methods of the `OpServiceJoin` class created during this thesis. The `create` function (Line 9) is called within the optimization in the `OpExecutor` class (Listing 4.11 Line 27) and returns an instance of the `OpServiceJoin` operator.

```

1  private OpServiceJoin(List<Binding> leftMat, Op leftOp, Op rightOp, Set<Var>
    ↪ commonVars, ExecutionContext qcxt, Op newService){
2  super(leftOp, newService);
3  this.newService = newService;
4  this.leftMat = leftMat;

```

```

5   this.commonVars = commonVars;
6   this.qCxt = qCxt;
7 }
8
9 public static OpServiceJoin create(List<Binding> leftMat, Op leftOp, OpService rightOp,
   ↪ Set<Var> commonVars, ExecutionContext qCxt) {
10  Op newService;
11  String rightOpQueryString = OpAsQuery.asQuery(rightOp).toString();
12  StringBuffer newQuery = new StringBuffer();
13  int valuesPos = rightOpQueryString.lastIndexOf("}");
14  valuesPos = rightOpQueryString.substring(0, valuesPos-1).lastIndexOf("}");
15  newQuery.append(rightOpQueryString.substring(0, valuesPos) + " VALUES (");
16  for (Var var : commonVars){
17    newQuery.append(var + " ");
18  }
19  newQuery.append(") {");
20  for (Binding b : leftMat) {
21    newQuery.append("(");
22    for (Var var : commonVars) {
23      Node n = b.get(var);
24      if (n == null || n.isBlank()){
25
26      } else if (n.isLiteral()){
27        String language = n.getLiteralLanguage();
28        String lexicalForm = n.getLiteralLexicalForm();
29        String dataTypeURI = n.getLiteralDatatypeURI();
30        String dataType = n.getLiteralDatatype().toString();
31        newQuery.append("\"" + n.getLiteralLexicalForm() + "\"");
32        if (!language.equals(""))
33          newQuery.append("@"+language + " ");
34        else {
35          newQuery.append("^<" + n.getLiteralDatatypeURI() + "> ");
36        }
37      } else if (n.isURI()){
38        newQuery.append("<" + n.toString() + "> ");
39      }
40    }
41    newQuery.append("\n");
42  }
43  newQuery.append("}\n"+rightOpQueryString.substring(valuesPos,
   ↪ rightOpQueryString.length()));
44  Query serviceQuery = QueryFactory.create(newQuery.toString(), "", Syntax.syntaxARQ);
45  newService = Algebra.compile(serviceQuery);
46  return new OpServiceJoin(leftMat, leftOp, rightOp, commonVars, qCxt, newService);
47 }

```

4.4 Evaluation of the SERVICE Clause Optimization

In order to test the optimization of the SERVICE clause evaluation within the Jena-API, multiple queries were stated against the Mondial SPARQL endpoint in two different ways. The first way, further on referred to as console execution method, was to use the command line tool known as SemWebJena-Tool, which is automatically compiled and created, whenever the Java code for the Mondial LOD servlet is compiled, whereas the second way was to use the SPARQL endpoint of the Mondial LOD servlet. Both ways were executed on an Lenovo ThinkPad E555 laptop running Ubuntu 18.04 with 16 GB RAM and a quad-core AMD[®] A8-7100 radeon r5 CPU with 1.9 GHz. The SPARQL endpoint ran on a virtual server on this laptop using Apache Tomcat[®] in version 9.0.13. To check if the designed optimization had any effect, the queries were executed both with and without the developed modification.

The queries used for testing are from different degrees of complexity and, therefore, ask for a broad variety of entities. While Listing 4.13 uses a very simple query to test the capability of the Mondial SPARQL endpoint, the query in Listing 4.14 is far more complex. The queries in Listing 4.15 and Listing 4.16 use a similar query as Listing 4.14, but one makes usage of the OPTIONAL keyword, while the other one uses the MINUS keyword. Listing 4.17 shows the same query as Listing 4.14, but this time uses the Mondial SPARQL endpoint at the online representation at <http://www.semwebtech.org/mondial/10/> instead the one running on the laptop, to test how the connection over the Internet affects the execution time. All of these queries ask for all Mondial-Cities that have the same Mondial-Name as a Mondial-Country. The last query used for testing (Listing 4.18) uses the Insee SPARQL endpoint as federated endpoint, in order to check if the Mondial endpoint is able to communicate with other SPARQL endpoints.

Listing 4.13: Simple query to test the Mondial SPARQL endpoint. During this query, all Mondial-Continents with their Mondial-Names are queried. During this query, the Mondial SPARQL endpoint at the localhost is evaluated.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT ?continent ?name
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5     ?continent a mon:Continent.
6     SERVICE <http://localhost:8080/mondial/10/sparql>{
7         ?continent mon:name ?name.
8     }
9 }

```

Listing 4.14: This query asks for all Mondial-Cities in Mondial-Countries which have the same Mondial-Name as a Mondial-Country. During this query, the Mondial SPARQL endpoint of the localhost is evaluated.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT DISTINCT ?country ?countryName ?city
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5     ?country a mon:Country.
6     ?country mon:name ?countryName .
7     ?country mon:hasCity ?cityCountry .
8     SERVICE <http://localhost:8080/mondial/10/sparql> {
9         ?city a mon:City.
10        ?city mon:name ?countryName
11    }
12 }

```

Listing 4.15: This query is similar to the one in Listing 4.14, but the `SERVICE` clause is embedded into an `OPTIONAL` clause. This will lead to the result, that all Mondial-Countries are returned. If there is a Mondial-City with the same name as a country, the city is returned as well. During this query, the Mondial SPARQL endpoint at the localhost is evaluated.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT DISTINCT ?country ?countryName ?city
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5     ?country a mon:Country .
6     ?country mon:name ?countryName .
7     ?country mon:hasCity ?cityCountry .
8     OPTIONAL {
9         SERVICE <http://localhost:8080/mondial/10/sparql> {
10            ?city a mon:City .
11            ?city mon:name ?countryName .
12        }
13    }
14 }

```

Listing 4.16: This query is similar to the one in Listing 4.14, but the `SERVICE` clause is embedded into an `MINUS` clause. This will lead to the result, that all Mondial-Countries are returned if there is no Mondial-City with the same name. During this query, the Mondial SPARQL endpoint at the localhost is evaluated.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT DISTINCT ?country ?countryName ?city
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5     ?country a mon:Country .
6     ?country mon:name ?countryName .
7     ?country mon:hasCity ?cityCountry .
8     MINUS {
9         SERVICE <http://localhost:8080/mondial/10/sparql> {
10             ?city a mon:City .
11             ?city mon:name ?countryName .
12         }
13     }
14 }

```

Listing 4.17: This query is the same as the one in Listing 4.14, but now the `SERVICE` clause asks the SPARQL endpoint at `http://www.semwebtech.org/mondial/10/`. This is done to check how the execution time is affected by the fact that the query is sent over the Internet instead of asking the localhost.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 SELECT DISTINCT ?country ?countryName ?city
3 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
4 WHERE {
5     ?country a mon:Country.
6     ?country mon:name ?countryName .
7     ?country mon:hasCity ?cityCountry .
8     SERVICE <http://www.semwebtech.org/mondial/10/sparql> {
9         ?city a mon:City.
10        ?city mon:name ?countryName
11    }
12 }

```

Listing 4.18: This query tests whether the developed optimization also affects other SPARQL endpoints. Therefore, the Insee endpoint is asked to return all cities, their department and their commune, if the corresponding Mondial-City name and the Mondial-Province name are matching.

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX insee: <http://rdf.insee.fr/def/geo#>
3 SELECT DISTINCT ?cityName ?commune ?departementName ?provinceNameTagged
4 FROM <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
5 WHERE {
6   ?country a mon:Country; mon:hasCity ?city.
7   ?province a mon:Province; mon:hasCity ?city .
8   ?city mon:name ?cityName.
9   ?province mon:name ?provinceName .
10  BIND (STRLANG(?cityName,"fr") as ?cityNameTagged) .
11  BIND (STRLANG(?provinceName,"fr") as ?provinceNameTagged) .
12  {
13    SERVICE <http://rdf.insee.fr/sparql> {
14      ?commune a insee:Commune; insee:nom ?cityNameTagged; insee:subdivisionDe
15      ↪ ?departement .
16      ?departement a insee:Departement; insee:nom ?departementName;
17      ↪ insee:subdivisionDe ?region .
18      ?region insee:nom ?provinceNameTagged
19    }
20  }

```

For each query, the execution time was measured and evaluated. Each measurement has been repeated ten times and the means of these execution times together with their standard deviation as error bars are displayed in Figure 4.2. All single measurements are listed in Appendix C. It can be seen, that for the queries of Listing 4.13 and Listing 4.16 only three respectively two execution methods are visible. This is, because the execution time without the developed modification has already been very low (circa 0.1 - 0.2 seconds), so that the effect of the optimizer is not visible in this graph. The difference of circa 9 seconds between the execution times of the console execution and the localhost execution can be explained by the fact, that the console execution always downloads the Mondial data set and loads it into the Jena-API, while the localhost execution uses a static model of the Mondial data set, which is already loaded when the servlet is started and queried for the first time. Furthermore, the graph shows, that the automatical insertion of the `VALUES` clause into the `SERVICE` clause reduces the execution time drastically. In cases, where more than 200 entities were queried within the `SERVICE` clause, the execution time was reduced from about 105 seconds to about 10 seconds for the Mondial SPARQL endpoint respectively from more than 170 seconds to less than 10 seconds for the method using the SemWebJena-Tool.

Unfortunately, this improvement is highly dependent of the query executed by the federated endpoint. The Insee endpoint was very generous when it comes to execution times, while the Wikidata endpoint terminated each query after 60 seconds of execution time. Due to the fact, that

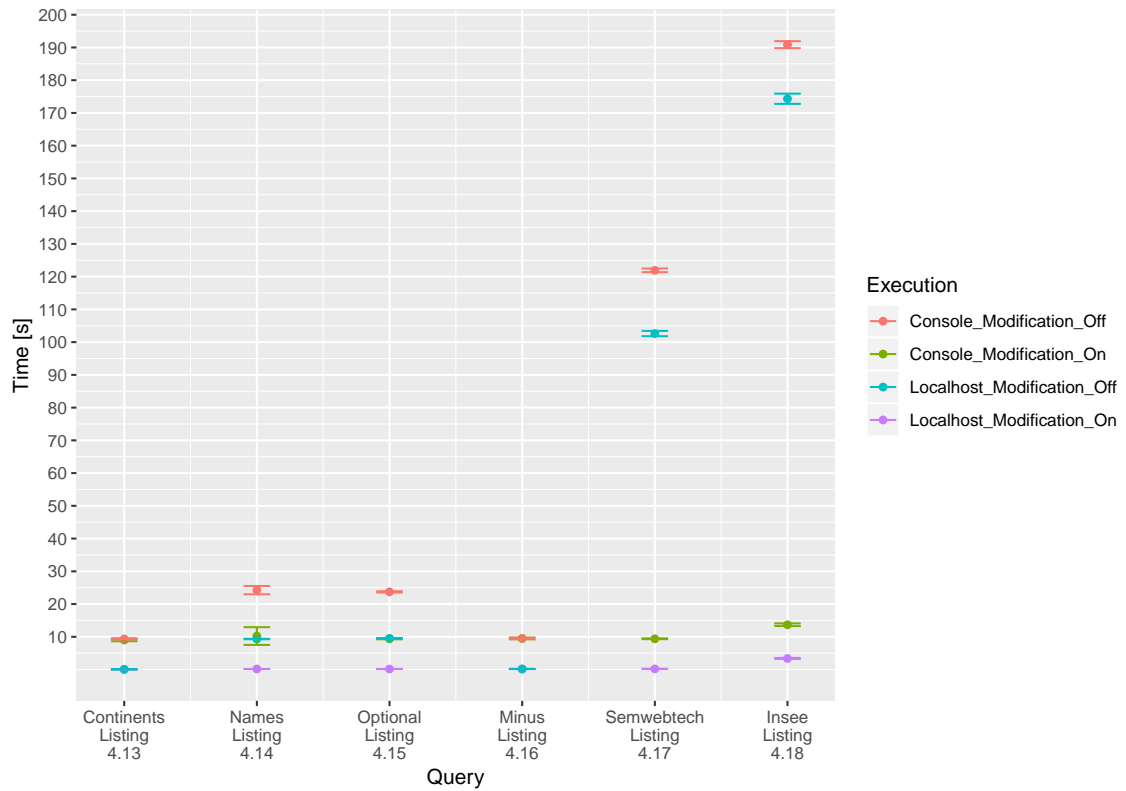


Figure 4.2: This figure shows the mean execution time for the queries listed above. To execute these queries, two different methods were used, first, the SemWebJena-Tool is used, and second the SPARQL endpoint running on a localhost is used. Both methods were executed with and without the developed optimization. It can be seen, that the modification had a huge effect if the asked query is complex, but almost no effect, if the query is simple.

Insee contains data about France only, Wikidata, however, contains data about everything, the queries stated against the Wikidata endpoint had much more local results which were not filtered beforehand and always ran into a server timeout. Therefore, the evaluation of these queries was not possible and is not displayed within the graph.

Chapter 5

Conclusion

This master's thesis was split into two different parts. While the first part was about including the Mondial data set into the LOD-Cloud and, therefore, creating a SPARQL endpoint for this data set, the second part was about the evaluation of the `SERVICE` clause within the Jena-API.

The first part began with the creation of an `owl:sameAs` file, containing connections from the Mondial data set to the Wikidata data set. Afterwards a Java servlet was created, which is used to serve as a SPARQL endpoint for the Mondial data set. This servlet can be queried in two different ways, for one thing by using the direct access via the URI by entering it in the address line of a web browser or by querying it with tools like `rappor`, and for another thing, as SPARQL endpoint, where a user can enter queries and get results from the Mondial data set.

Already during the creation of the servlet, multiple unwanted behaviors were observed, where any of them could be eliminated. Some misbehaviors, however, could not be resolved. When queried via tools like `rappor`, the implemented SPARQL endpoint works flawlessly.

In the second part of this thesis, the evaluation of the `SERVICE` clause by the Jena-API was analyzed and optimized. Therefore, it was necessary to understand in which way the Jena-API executes `SERVICE` clauses and if it makes use of the `VALUES` clause. Afterwards, the decision was made, to not only change the original query string, but to change the original evaluation path of the Jena-API, whenever there is a `SERVICE` clause used within a query. This was done, because the change of the query string would not positively affect the execution time, because for analyzing the string, the query had to be compiled and, thereby, the `SERVICE` clause would be executed. After digging through the evaluation path of a SPARQL query within the Jena-API and introducing a new Java class into this evaluation tree, the internal query optimizer of the Jena-API had to be bypassed.

In the end of this thesis, an evaluation of this optimization was done. This evaluation showed, that the developed optimization had a huge effect, if the federated query part was complex, but almost no effect, if the query was simple. However, a negative effect on the query was never observed, if

the query was executed correctly. This was not always the case, because some queries, especially when stated against the Wikidata data set, ran into server timeouts, which were caused by the fact, that Wikidata allows only queries with an execution time from less than 60 seconds.

All in all, the results of this thesis have to be evaluated positively. Besides some minor bugs and some execution time complaints of other SPARQL endpoints, the Mondial SPARQL endpoint together with the optimized `SERVICE` clause evaluation works well.

5.1 Future Work

During this thesis, some issues came up, which have not yet been resolved. For one thing, it is possible to replace each `OpJoin` operator within the evaluation tree, whenever the right child operator of this `OpJoin` operator is an `OpService` operator. The new operator could be called `LeftMatServiceJoin`. This would lead to the effect, that the internal optimizer of the Jena-API would not translate this `JOIN` operation into a `SEQUENCE` operation and, therefore, could stay within the execution path to optimize some other operators.

Another point, not analyzed during this thesis, is the evaluation of `CONSTRUCT` queries within the Jena-API. In extension to this analysis, it is thinkable to create the possibility to use nested `CONSTRUCT` queries, where an RDF graph is created within a `CONSTRUCT` query inside the `FROM` clause of a regular `SELECT` query. To do so, it would be necessary to trace the execution of a regular `CONSTRUCT` query within the Jena-API and to call these functions whenever there is a `CONSTRUCT` clause within a `FROM` clause.

Bibliography

- [BB08] D. Beckett and T. Berners-Lee. Turtle - Terse RDF Triple Language. Website, January 2008. Online available at <https://www.w3.org/TeamSubmission/2008/SUBM-turtle-20080114/>. Last visited 04/2019.
- [Bec14] D. Beckett. Raptor RDF Syntax Library. Website, November 2014. Online available at <http://librdf.org/raptor/>. Last visited 03/2019.
- [Ber06] T. Berners-Lee. Linked Data. Website, July 2006. Online available at <https://www.w3.org/DesignIssues/LinkedData.html>. Last visited 03/2019.
- [BG14] D. Brickley and R. Guha. RDF Schema 1.1. Website, February 2014. Online available at <https://www.w3.org/TR/rdf-schema/>. Last visited 03/2019.
- [BM12] P. Biron and A. Malhotra. W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. Website, April 2012. Online available at <https://www.w3.org/TR/xmlschema11-2/>. Last visited 03/2019.
- [Fou] The Apache Software Foundation. Apache Jena. Website. Online available at <https://jena.apache.org/>. Last visited 03/2019.
- [GS14] F. Gandon and G. Schreiber. RDF 1.1 XML Syntax. Website, February 2014. Online available at <https://www.w3.org/TR/rdf-syntax-grammar/>. Last visited 03/2019.
- [HS13] S. Harris and A. Seaborne. SPARQL 1.1 Query Language. Website, March 2013. Online available at <https://www.w3.org/TR/sparql11-query/>. Last visited 03/2019.
- [KCM14] G. Klyne, J. Carroll, and B. McBride. RDF 1.1 Concepts and Abstract Syntax. Website, February 2014. Online available at <https://www.w3.org/TR/rdf11-concepts/>. Last visited 03/2019.
- [Mv02] D. McGuinness and F. van Harmelen. Feature Synopsis for OWL Lite and OWL. Website, July 2002. Online available at <https://www.w3.org/TR/2002/WD-owl-features-20020729/>. Last visited 04/2019.

- [OWL12] W3C. OWL Working Group. OWL 2 Web Ontology Language. Website, December 2012. Online available at <https://www.w3.org/TR/owl2-overview/>. Last visited 03/2019.
- [PS04] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Website, October 2004. Online available at <https://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/>. Last visited 04/2019.
- [RDF01] W3C. RDF Core Working Group. N-Triples. Website, June 2001. Online available at <https://www.w3.org/2001/sw/RDFCore/ntriples/>. Last visited 04/2019.

Glossary

| Notation | Description | Page |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| API | An Application Programming Interface (API) is a collection of subroutines and libraries used to create applications. | xi, xii, 1, 2, 13, 15, 21, 26, 29, 31–39, 41, 44, 47, 48, 51, 55, 56 |
| BGP | A set of RDF triples intended as a conjunctive query is called a Basic Graph Pattern (BGP). | 7, 8, 32, 36, 38, 51 |
| CPU | The Central Processing Unit (CPU) is the main electronic circuit within a computer used to execute programs. Therefore, each program is translated into a pile of arithmetic, logic, controlling and input/output operations, which are executed by the CPU. | 41, 51 |
| DBIS | Databases and Information Systems | 1, 3, 15, 51 |

| Notation | Description | Page |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| HTML | The Hypertext Markup Language (HTML) is a markup language especially designed for hypertexts and therefore used to display websites within a web browser. | 1, 16, 18, 19, 21, 23, 28, 29, 52 |
| HTTP | The Hypertext Transfer Protocol (HTTP) is used to transfer data between computers. It is mostly used to transfer websites from the World Wide Web to web browsers. | 11, 12, 16, 52 |
| LOD | Linked Open Data | vii, xi, xii, 1–3, 12–15, 17–22, 24, 26, 28–30, 35, 41, 47, 55 |
| N-Triples | N-Triples is a special syntax to describe RDF data. Each RDF statement is represented by one triple. | 5, 6, 52, 53 |
| OWL | Web Ontology Language | xi, 6, 7 |
| RAM | The Random Access Memory (RAM) is a data storage for computers. It has a much higher access rate than hard drives or solid state drives and, therefore, is used to store data and machine code, that is currently being executed. | 22, 41, 52 |
| raper | The Raptor RDF parsing and serializing utility (raper) is a command line tool which provides a parser for RDF data. | xi, 13, 15, 18, 21, 23, 24, 27– 29, 47, 52 |

| Notation | Description | Page |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|
| RDF | The Resource Description Framework (RDF) is a framework to represent information within the Web of Data. | xi, 1, 3–10, 12, 15–19, 21–24, 28, 29, 48, 51–53 |
| RDFS | RDF Schema | xi, 6, 7 |
| SPARQL | The SPARQL Protocol and RDF Query Language (SPARQL) is a query language which can be used to query RDF data. | vii, xi, xii, 1, 2, 7–24, 26, 28, 29, 31, 32, 34–38, 41–45, 47, 48, 53 |
| Turtle | The Terse RDF Triple Language (Turtle) is an abbreviated syntax for RDF data. It is based on N-Triples syntax, but shortened in such a way, that prefixes are used to replace repeatedly occurring URI parts. | 5–8, 53 |
| URI | A Uniform Resource Identifier (URI) is a world-wide uniquely defined identifier for a resource. It is highly related to the URL in such a way, that URIs form a super set of the URLs. | xii, 4–6, 12, 17–23, 26, 28, 29, 32, 33, 47, 53 |
| URL | The Uniform Resource Locators (URLs) form a subset of the URIs and are worldwide uniquely defined identifiers for a resource on the World Wide Web. | 2, 4, 13, 17, 20, 22, 23, 53 |

| Notation | Description | Page |
|-----------------|----------------------------------------------------------------------------------------------------------------|-------------------------------------|
| W3C | The World Wide Web Consortium (W3C) is an international community to develop standards for the World Wide Web. | 1, 4–7, 13, 15, 24, 50, 54 |
| XML | Extensible Markup Language | 15, 16, 18, 21, 23 |

Appendix A

How to Integrate the Developed Optimizations into a new Jena-API

Due to the fact that the Jena-API is continuously being developed, it might be necessary to update its version within the Mondial LOD servlet. In order to keep all changes within the Jena-API developed during this thesis, some of the original classes have to be changed.

Within the `org.apache.jena.sparql.algebra` package, there are two classes which have to be modified and one class which has to be added to the package.

First of all, the class `OpJoinService.java` created during this thesis has to be added to the package. This can be done by copying it into the corresponding directory within the Jena-API-Structure. The first class to change is the `OpJoin.java` class. Within it the constructor (about line 68) has to be changed from `private` access to, at least, `protected` or maybe even to `public` access. This is necessary, because the `OpJoinService` class extends the `OpJoin` class and within the constructor of the `OpJoinService` class the `super` constructor is called.

The last modification in this package has to be done within the `Algebra.java` class. First of all, a `private static boolean` variable `DEBUG` has to be defined, in order to activate or deactivate some debugging outputs. A good point to define this variable is about line 55 (after the class is opened and before the first method is declared). A second modification of this class is within the `public static Op optimize(Op op, Context context)` method. Everything after the line `// Modified by Martin Heinemann ...` (this line included) until the end of the method has to be added. This part will listen to the boolean variable indicating if the optimized `SERVICE` clause evaluation should be activated or not. If it is, the internal optimizer of the Jena-API has to be bypassed.

The second package to modify is the `org.apache.jena.sparql.engine.http` package. Within it, the file `HttpQuery.java` has to be changed in such a way, that the boolean variable `forcePOST` is set to `true` (~line 67). This will force the Mondial LOD servlet to always use

the `doPost` request instead of a `doGet` request in order to avoid redundant code within the `RDFServlet.java` class.

The final file which has to be changed in order to transfer all optimizations of this master's thesis into a new version of the Jena-API is the `OpExecutor.java` class of the `org.apache.jena.sparql.engine.main` package.

Again, a debug boolean variable, which has `private static` access, has to be added. A good point for this might be line 69 (after the class is opened and before the first method is declared).

In addition to this, the methods `protected QueryIterator execute(OpJoin opJoin, QueryIterator input)`, `protected QueryIterator execute(OpLeftJoin opLeftJoin, QueryIterator input)` and `protected QueryIterator execute(OpMinus opMinus, QueryIterator input)` have to be modified. In all three cases, the case that one of the child operators contains a `SERVICE` clause has to be caught. Therefore, the lines between the line `boolean bla = false;` and the first `return qIter;` line (both included), have to be added to these methods.

Appendix B

owl:sameAs Queries

This chapter contains all queries used to create the Mondial-SameAs file available at <https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial-sameas.n3>. Due to the fact, that these queries are self explaining, no captions are added.

```
1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6 SELECT DISTINCT ?mondial ?wikidata
7 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8 WHERE {
9     SERVICE <https://query.wikidata.org/sparql>{
10         ?wikidata wdt:P31 wd:Q5107. # Continent
11         ?wikidata rdfs:label ?label.
12         FILTER (langMatches( lang(?label), "*" ) )
13         BIND (STR(?label) AS ?labelWithoutLanguageTag)
14     }
15     ?mondial a mon:Continent.
16     ?mondial mon:name ?labelWithoutLanguageTag.
17 }
```

```
1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 PREFIX owl: <http://www.w3.org/2002/07/owl#>
6
7 SELECT DISTINCT ?mondial ?wikidata
8 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
9 FROM <file:../Mondial/sameAs.n3>
```

```

10 WHERE {
11   ?mondialContinent a mon:Continent.
12   ?mondialContinent owl:sameAs ?wikidataContinent.
13   SERVICE <https://query.wikidata.org/sparql>{
14     ?wikidata wdt:P31/wdt:P279* wd:Q6256. # Country.
15     ?wikidata wdt:P30 ?wikidataContinent.
16     ?wikidata rdfs:label ?label.
17     FILTER (langMatches( lang(?label), "en" ) )
18     BIND(STR(?label) AS ?labelWithoutLanguageTag)
19   }
20   ?mondial a mon:Country.
21   ?mondial mon:name ?name.
22   ?mondial mon:encompassed ?mondialContinent.
23   FILTER(?labelWithoutLanguageTag = ?name)
24 }

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 prefix owl: <http://www.w3.org/2002/07/owl#>
6
7 SELECT DISTINCT ?mondial ?wikidata
8 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
9 FROM
10   ↔ <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial-sameAs.n3>
11 WHERE {
12   ?mondial a mon:City.
13   ?mondialProvince a mon:Province.
14   ?mondialProvince owl:sameAs ?wikidataProvince.
15   SERVICE <https://query.wikidata.org/sparql>{
16     ?wikidata wdt:P31/(wdt:P279)* wd:Q1549591 .
17     ?wikidata wdt:P131 ?wikidataProvince .
18     ?wikidata rdfs:label ?label.
19     FILTER (langMatches( lang(?label), "en" ) )
20     BIND(STR(?label) AS ?labelWithoutLanguageTag)
21   }
22   ?mondialProvince mon:hasCity ?mondial.
23   ?mondial mon:name ?mondialName.
24   ?mondial mon:otherName ?mondialOtherName.
25   FILTER(?mondialName = ?labelWithoutLanguageTag || ?mondialOtherName =
26     ↔ ?labelWithoutLanguageTag)

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5

```

```

6 SELECT DISTINCT ?mondial ?wikidata
7 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8 WHERE {
9     SERVICE <https://query.wikidata.org/sparql>{
10         ?wikidata wdt:P31/wdt:P279* wd:Q107425. # Landscape
11         ?wikidata rdfs:label ?label.
12         FILTER (langMatches( lang(?label), "en" ) )
13         BIND(STR(?label) AS ?labelWithoutLanguageTag)
14     }
15     ?mondial a mon:Desert.
16     ?mondial mon:name ?labelWithoutLanguageTag.
17 }

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6 SELECT DISTINCT ?mondial ?wikidata
7 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8 WHERE {
9     SERVICE <https://query.wikidata.org/sparql>{
10         ?wikidata wdt:P31/wdt:P279* wd:Q41710. # Ethnic group
11         ?wikidata rdfs:label ?label.
12         FILTER (langMatches( lang(?label), "*" ) )
13         BIND(STR(?label) AS ?labelWithoutLanguageTag)
14     }
15     ?mondial a mon:EthnicGroup.
16     ?mondial mon:name ?Name.
17     FILTER(?Name = ?labelWithoutLanguageTag || CONCAT(?Name, "s") =
18         ↪ ?labelWithoutLanguageTag)

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 prefix owl: <http://www.w3.org/2002/07/owl#>
6
7 SELECT DISTINCT ?mondial ?wikidata
8 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
9 FROM
10     ↪ <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial-sameAs.n3>
11 WHERE {
12     ?mondial a mon:Island.
13     ?mondial mon:locatedInWater ?mondialSea.
14     ?mondialSea owl:sameAs ?wikidataSea.
15     SERVICE <https://query.wikidata.org/sparql>{
16         ?wikidata wdt:P31/wdt:P279* wd:Q23442. # Island

```

```

16     ?wikidata rdfs:label ?label.
17     ?wikidata wdt:P206 ?wikidataSea.
18     FILTER (langMatches( lang(?label), "en" ) )
19     BIND(STR(?label) AS ?labelWithoutLanguageTag)
20 }
21 ?mondial mon:name ?name .
22 FILTER(?name = ?labelWithoutLanguageTag)
23 }

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 prefix owl: <http://www.w3.org/2002/07/owl#>
6
7 SELECT DISTINCT ?mondial ?wikidata
8 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
9 WHERE {
10     SERVICE <https://query.wikidata.org/sparql>{
11         ?wikidata wdt:P31/wdt:P279* wd:Q1402592. # Island group
12         ?wikidata rdfs:label ?label.
13         FILTER (langMatches( lang(?label), "*" ) )
14         BIND(STR(?label) AS ?labelWithoutLanguageTag)
15     }
16     ?mondial a mon:Islands.
17     ?mondial mon:name ?labelWithoutLanguageTag .
18 }

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6 SELECT DISTINCT ?mondial ?wikidata
7 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8 WHERE {
9     SERVICE <https://query.wikidata.org/sparql>{
10         ?wikidata wdt:P31/wdt:P279* wd:Q15324. # lake.
11         ?wikidata rdfs:label ?label.
12         ?wikidata wdt:P2046 ?wikidataArea.
13         FILTER (langMatches( lang(?label), "*" ) )
14         BIND(STR(?label) AS ?labelWithoutLanguageTag)
15     }
16     ?mondial a mon:Lake.
17     ?mondial mon:area ?mondialArea.
18     ?mondial mon:name ?labelWithoutLanguageTag.
19     FILTER((?mondialArea >= .99*?wikidataArea && ?mondialArea <= 1.01*?wikidataArea) ||
20             ↔ (?wikidataArea >= .99*?mondialArea && ?wikidataArea <= 1.01*?mondialArea))

```



```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6 SELECT DISTINCT ?mondial ?wikidata
7 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8 WHERE {
9     SERVICE <https://query.wikidata.org/sparql>{
10         ?wikidata wdt:P31/wdt:P279* wd:Q17376908. # Q17376908 = Languoid
11         ?wikidata rdfs:label ?label.
12         FILTER (langMatches( lang(?label), "*" ) )
13         BIND(STR(?label) AS ?labelWithoutLanguageTag)
14     }
15     ?mondial a mon:Language.
16     ?mondial mon:name ?labelWithoutLanguageTag.
17 }

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6 SELECT DISTINCT ?mondial ?wikidata
7 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8 WHERE {
9     SERVICE <https://query.wikidata.org/sparql>{
10         ?wikidata wdt:P31/wdt:P279* wd:Q8502.
11         ?wikidata rdfs:label ?label.
12         ?wikidata wdt:P2044 ?wikidataElevation.
13         FILTER (langMatches( lang(?label), "en" ) )
14         BIND(STR(?label) AS ?labelWithoutLanguageTag)
15     }
16     ?mondial a mon:Mountain.
17     ?mondial mon:name ?name.
18     ?mondial mon:elevation ?mondialElevation.
19     FILTER(?name = ?labelWithoutLanguageTag && ?mondialElevation = ?wikidataElevation)
20 }

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX wikibase: <http://wikiba.se/ontology#>
5 PREFIX bd: <http://www.bigdata.com/rdf#>
6 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
7
8 SELECT DISTINCT ?mondial ?wikidata
9 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>

```

```

10 WHERE {
11   SERVICE <https://query.wikidata.org/sparql>{
12     ?wikidata wdt:P31/wdt:P279* wd:Q43229 . # Organization
13     ?wikidata wdt:P1813 ?wikidataAbbreviation .
14     FILTER(langMatches( lang(?wikidataAbbreviation), "en" ))
15     BIND(STR(?wikidataAbbreviation) AS ?abbreviationWithoutLanguageTag)
16   }
17   ?mondial a mon:Organization.
18   ?mondial mon:abbrev ?abbreviationWithoutLanguageTag.
19 }

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 prefix owl: <http://www.w3.org/2002/07/owl#>
6
7 SELECT DISTINCT ?mondial ?wikidata
8 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
9 FROM <file:./Mondial/sameAs.n3>
10 WHERE {
11   ?mondialCountry a mon:Country.
12   ?mondialCountry owl:sameAs ?wikidataCountry.
13   SERVICE <https://query.wikidata.org/sparql>{
14     ?wikidata wdt:P31/wdt:P279* wd:Q10864048. # first-level administrative country
15     ↪ subdivision
16     ?wikidata wdt:P17 ?wikidataCountry.
17     ?wikidata rdfs:label ?label.
18     FILTER(langMatches( lang(?label), "*" ))
19     BIND(STR(?label) AS ?labelWithoutLanguageTag)
20   }
21   ?mondialCountry mon:hasProvince ?mondial.
22   ?mondial mon:name ?labelWithoutLanguageTag.
23 }

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 prefix owl: <http://www.w3.org/2002/07/owl#>
6
7 SELECT DISTINCT ?mondial ?wikidata
8 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
9 FROM
10   ↪ <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial-sameAs.n3>
11 WHERE {
12   ?mondialCountry a mon:Country.
13   ?mondialCountry owl:sameAs ?wikidataCountry.
14   SERVICE <https://query.wikidata.org/sparql>{

```

```

14     ?wikidata wdt:P31/wdt:P279* wd:Q13220204. # second-level administrative country
      ↪ subdivision
15     ?wikidata wdt:P17 ?wikidataCountry.
16     ?wikidata rdfs:label ?label.
17     FILTER(langMatches( lang(?label), "*" ) )
18     BIND(STR(?label) AS ?labelWithoutLanguageTag)
19   }
20   ?mondialCountry mon:hasProvince ?mondial.
21   ?mondial mon:name ?labelWithoutLanguageTag.
22 }

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6 SELECT DISTINCT ?mondial ?wikidata
7 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8 WHERE {
9     SERVICE <https://query.wikidata.org/sparql>{
10         ?wikidata wdt:P31/wdt:P279* wd:Q5390013. # Belief system.
11         ?wikidata rdfs:label ?label.
12         FILTER (langMatches( lang(?label), "*" ) )
13         BIND(STR(?label) AS ?labelWithoutLanguageTag)
14     }
15     ?mondial a mon:Religion.
16     ?mondial mon:name ?name.
17     FILTER(?name = ?labelWithoutLanguageTag || CONCAT(?name, "s") =
18             ↪ ?labelWithoutLanguageTag)

```

```

1 PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2 PREFIX wd: <http://www.wikidata.org/entity/>
3 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6 SELECT DISTINCT ?mondial ?wikidata
7 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8 WHERE {
9     SERVICE <https://query.wikidata.org/sparql>{
10         ?wikidata wdt:P31 wd:Q4022. # river.
11         ?wikidata wdt:P2043 ?wikidataLength.
12         ?wikidata rdfs:label ?label.
13         FILTER (langMatches( lang(?label), "*" ) )
14         BIND(STR(?label) AS ?labelWithoutLanguageTag)
15     }
16     ?mondial a mon:River.
17     ?mondial mon:length ?mondialLength.
18     ?mondial mon:name ?labelWithoutLanguageTag.

```

```

19  FILTER((?mondialLength >= .99*?wikidataLength && ?mondialLength <=
    ↪ 1.01*?wikidataLength) || (?wikidataLength >= .99*?mondialLength &&
    ↪ ?wikidataLength <= 1.01*?mondialLength))
20 }

```

```

1  PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2  PREFIX wd: <http://www.wikidata.org/entity/>
3  PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6  SELECT DISTINCT ?mondial ?wikidata
7  FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8  WHERE {
9      SERVICE <https://query.wikidata.org/sparql>{
10         ?wikidata wdt:P31/wdt:P279* wd:Q15324. # body of water.
11         ?wikidata rdfs:label ?label.
12         FILTER (langMatches( lang(?label), "en" ) )
13         BIND(STR(?label) AS ?labelWithoutLanguageTag)
14     }
15     ?mondial a mon:Sea.
16     ?mondial mon:name ?labelWithoutLanguageTag.
17 }

```

```

1  PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2  PREFIX wd: <http://www.wikidata.org/entity/>
3  PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6  SELECT DISTINCT ?mondial ?wikidata
7  FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
8  WHERE {
9      SERVICE <https://query.wikidata.org/sparql>{
10         ?wikidata wdt:P31/wdt:P279* wd:Q8072. # ?CW is a (wdt:P31) Volcano (wd:Q8072)
11         ?wikidata rdfs:label ?label.
12         FILTER (langMatches( lang(?label), "en" ) )
13         BIND(STR(?label) AS ?labelWithoutLanguageTag)
14     }
15     ?mondial a mon:Volcano.
16     ?mondial mon:name ?name.
17     FILTER(?name = ?labelWithoutLanguageTag)
18 }

```

```

1  PREFIX mon: <http://www.semwebtech.org/mondial/10/meta#>
2  PREFIX wd: <http://www.wikidata.org/entity/>
3  PREFIX wdt: <http://www.wikidata.org/prop/direct/>
4  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5  PREFIX owl: <http://www.w3.org/2002/07/owl#>
6
7  SELECT DISTINCT ?mondial ?wikidata

```

```
8 FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
9 WHERE {
10   SERVICE <https://query.wikidata.org/sparql>{
11     ?wikidata wdt:P31/wdt:P279* wd:Q33837. # Archipelago.
12     ?wikidata rdfs:label ?label.
13     FILTER (langMatches( lang(?label), "en" ) )
14     BIND(STR(?label) AS ?labelWithoutLanguageTag)
15   }
16   ?mondial a mon:Archipelago.
17   ?mondial mon:name ?name.
18   FILTER(?labelWithoutLanguageTag = ?name)
19 }
```


Appendix C

Execution Times of Evaluation Queries

In this chapter, the execution times collected during the evaluation of the `SERVICE` clause optimization (Section 4.4) are listed. For each query (Listing 4.13 - 4.18) an own graph is shown. Within these graphs, the **red** dots represent the execution times without the developed modifications, and the **blue** one the times with the developed optimizations.

