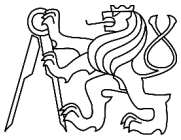
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF MASTER'S THESIS

Title:	Summarizing Linked Open Data Datasets
Student:	Bc. Jana Čabaiová
Supervisor:	Ing. Milan Dojčinovski
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

In the recent years, many datasets have been published as a part of the Linked Open Data cloud. Nevertheless, searching for an appropriate dataset requires significant amount of effort. The goal of this thesis is to simplify the process of searching for datasets by providing summarized information for each dataset.
Guidelines:
- Get familiar with the Linked Data principles and the current state of the Linked Open Data (LOD) cloud.
- Analyse the current datasets access mechanisms such as SPARQL, Linked Data Fragments, HDF, etc.
- Develop a method for dataset summarizations and implement an application that will enable users to:
  1) specify a dataset for summarization,
  2) define domains for summarization,
  3) perform macro visualization - domain coverage for a dataset, and micro visualization - completeness of information for a given entity type, and
  4) compare datasets.
- Validate the method on real data from the LOD cloud (DBpedia, LinkedGeoData, etc).

References

Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague January 5, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Master's thesis

Summarizing Linked Open Data Datasets

Bc. Jana Čabaiová

Supervisor: Ing. Milan Dojčinovski

7th January 2017

Acknowledgements

I wish to express my sincere thanks to my thesis supervisor, Ing. Milan Dojčinský for his time and advices throughout writing this master thesis. I am also grateful to the Department of Faculty for the implementation of the subject MI-SWE, thanks to which I learned the semantic technologies. I would like to express my gratitude also to my family and friends for their help, patience and mainly support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 7th January 2017 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Čabaiová, Jana. *Summarizing Linked Open Data Datasets.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Práce se zabývá studiem projektu Linked Open Data, jeho aktuálního stavu a také shrnutím jednotlivých sémantických technologii, jako je RDF model, dotazovací jazyk SPARQL, různé formáty datasetů a různé přístupy k jednotlivým datasetům. Součástí práce je také vývoj webové aplikace, který zahrnuje analýzu, návrh, implementaci a také testování dané aplikace. Hlavní metoda této aplikace má umožňovat výpočet sumarizace LOD datasetů na základě specifikovaných domén a entit, což znamená poměr zastoupení jednotlivých entit v doménách v rámci konkrétního datasetu.

Hlavním výsledkem této práce je vytvořená a otestováná webová aplikace s výše zmíněnou implementovanou metodou na reálných datesetech DBpedia a GeoNames a také zpracování a porovnání jednotlivých výsledků. Tato aplikace by měla být užitečná zejména pro ty, kteří potřebují zjistit doménové zaměření jejich Linked Open Data datasetu nebo potřebují porovnat dva různé datasety na doménové úrovni.

Klíčová slova Linked Open Data, LOD Cloud, RDF, SPARQL, HDT, DBpedia, GeoNames, dataset, doména, entita, predikát, sumarizácia, webová aplikace, ontologie

Abstract

The work deals with the study of the project Linked Open Data, its current state and also with the overview of the particular semantic technologies. It is RDF model, query language SPARQL, different formats for RDF datasets and the different accesses to the particular datasets. Part of the work is also the development of the web application which contains analysis, design, implementation and testing of the particular application. The main method of this application should enable the calculation of the summarization of LOD datasets on the base of domain specification, which means calculation of domains and entities proportion in particular dataset.

The main result of this work is created and tested web application with the above mentioned implemented method on the real datasets DBpedia a GeoNames and also the processing and comparing of the particular results. This application should be useful mainly for these, who need to find out domain representation of their Linked Open Data dataset or they need to compare domain representation of two different datasets.

Keywords Linked Open Data, LOD cloud, RDF, Sparql, HDT, DBpedia, GeoNames, dataset, domain, entity, predicate, summarization, web application, ontology

Contents

List of Figures

List of Tables

Introduction

Motivation

The world is surrounded by data resources all around us, in the electronic or press form, also structural and nonstructural. Furthermore, nowadays people try to substitute all manual works with computer technique. To this also belongs processing of data. For most people are websites the most often data resource, on which are data mostly in the nonstructural form. So, they are not machine-processable and usable to the next compute processing. Base on this problem there was risen the idea for the creation of Linked Open Data, what can be defined, for short as public linked structural data on the Web. This Linked Open Data recently have began to expand. Because of that the requirements for dataset exploration rises and it would be beneficial to know what the dataset is about, so which domains are the most covered in it.

Objectives

The aim of this master thesis is to introduce with the area Linked Open Data and to study particular possibilities of accesses to them. The other and also the main aim is analysis, design and the implementation of the web application, which could process this LOD data in the particular agreed formats. Also, it could make the analysis about the data according to defined domains and entities. In this case the analysis represents the calculation of the occurrence of data within the particular domains. Another functionality in the application should be possibility to compare two datasets from the domain point of view. In the conclusion it is necessary the developed application to test on the real Linked Open Data, as DBpedia or GeoNames.

Organization of the thesis

This work is divided into the following chapters:

- Theoretical background- this chapter is focused on the introduction with the required issue and defining of the main terms as Web of Data, Linked Open Data or RDF, which are needed for the development of this web application. It also includes the actual state of Linked Open Data and possibilities to access them such as SPARQL, Linked Data Fragment or HDT.
- State of the art - this chapter includes research about existing applications and tools which are focused on the summarization of Linked Open Data.
- Analysis - this chapter describes requests which are put on the designed application in detail. It is the processing of the functional and non-functional requirements, use case diagrams and domain model.
- Design - this chapter contains design models as diagram of classes and database diagram. Also, it includes the choice of technology, architecture and wireframes of the application.
- Implementation - this chapter contains particular parts of implementation and the comparison with the designed solution.
- Testing - this chapter contains integrating testing of the application and testing among different web browsers. It also includes application use cases and demonstration scenario of the application usage.
- Experiments and results - this chapter contains validation of implemented application on real LOD datasets, like DBpedia and GeoNames and their comparison.

Chapter 1

Theoretical background

This chapter discusses the issue of receiving and processing structured data from the Web. Further the reader will be introduced with the terms like Linked Open Data, Web of documents, Web of Data, Resource Description Framework, SPARQL and also how these terms are connected with the publishing structured Linked Data on the Web. Also this chapter describes what LOD Cloud represents and different mechanism to access LOD data.

1.1 What is Web of Data

Data are all around us. We create them, save them, share them, publish and work with them. One of the biggest and the most used data sources in the form, which is primarily aimed for the human processing are websites. Data can be found on the Web in various formats, for instance, CSV, PDF, HTML tables, plain text, etc. Current Web is considered as the Web of documents, so it directs to the documents, which the user can search and then refer to the other documents. Unfortunately, data in these documents can be processed only by human, they are not machine-processable. Many users would need to get data as autonomous entities directly from the document on the Web in the structured format for their next compute usage.

Actual Web, that is Web of documents, is built on these elementary principles:

- language HTML, in which documents on Web are published,
- URL like the unique global identifier of the documents,
- protocol HTTP, upon which it is possible to access to documents according to their URL,
- hyperlinks among documents [2] .

Based on principles above, it is possible to publish particular documents in the HTML language on the Web, access to them by protocol HTTP based on their unique URL, or through the hypertext link (anchor tag and href attribute). But mostly it is not possible to receive data directly about concrete entities, which are parts of particular documents on the classic website. For instance, tennis player Serena Williams. The user is capable of finding 1000 documents on the Web, in which this player will be mentioned. One of them would describe her personal information, the second one will write about her current ranking, the third one would describe her tournament performance timeline, fourth one, for instance, will reflect her actual life and so on. But this information mostly cannot be machine-processable, because they are not in the structured format, and they do not describe data directly about this tennis player as about an entity. The next problem, which has place in this section, is that data in documents are not connected. One website contains thousands of information. Although hyperlinks exist on the Web, but these hyperlinks refer from one document to another, but not from concrete entity in the document to another concrete entity in the different document.

According to the mentioned issues, there was made the idea to publish raw linked data on the Web directly, which can be further processed by machine with the aim to provide information about given entities from various resources, and to be able to answer more complex questions.

On the current Web there exist several possibilities of publishing of raw (structured) data:

- data files in various formats, for instance CSV, XML, XLSX, etc.,
- microformats,
- web API's.

Certainly, the easiest way of publishing of raw data on the Web is publishing the link to downloading data files. However, these data files can be in different formats, therefore external applications, which particular data need to use, have to be prepared on processing several formats, not unique one. In addition, with data, which are available from data files on the Web, it is mostly possible to work with them only in local way, because they have either local identifiers or they have none, so it is difficult to work with them from the global point of view, and to connect them e.g. with the data received from the different resource.

Next possibility of publishing of raw data on the websites is microformats. The definition of microformats is following: Designed for humans first and machines second, microformats are a set of simple, open data formats built upon existing and widely adopted standards [3]. Microformats hence enable to publish structured data through embedding of data in Web pages. The main deficiency of microformats is the fact, that they provide narrow set of categories

Web of documents	**Web of Data**
Uniform format HTML for documents publishing	Different formats for data publishing, e.g. CSV, XML, XLS, JSON, etc.
URL as unique global identification of document	No unique global identification for entities in data
HTTP protocol for localization and access to documents	HTTP protocol for localization data file or web API, but no access to entities in data
Hyperlinks between documents	No hyperlinks between entities in data in those formats

Table 1.1: Comparison: Web of documents vs. Web of Data providing data in microformats, web API's or data files

for the description of the particular entities, and also specific set of attributes for the description of the particular relations among entities. Categories and attributes are not possible to define arbitrarily, so this technology can be used only for specific data, at which it is possible to use defined categories and attributes of microformats.

Next advanced method of getting data from websites is web API's (Application Programming Interfaces). These API's provide machine-processable data for further processing, usually in formats JSON or XML with using HTTP protocol. Providing of data in this way has also some limitations, namely that equivalent to HTML anchor element with the attribute href, which could be referencing to similar data, does not exist. Furthermore, for every data source exists autonomous API, which has its own methods and concepts of usage, so it is very difficult to integrate these datasets, received from many web API's into one application.

Web of Data, as it is Web, must correspond to principles of the classic Web. Based on above listed knowledge, it is possible to deduce, that concepts, which allow access to raw data on the websites, they do not reply with principles, on which the Web of documents was built. Therefore, they cannot create the Web of Data. With the usage of above mentioned raw data sources on the Web we can conclude, that compact format for publishing of data does not exist and also unique global identifier and links for similar entities do not exist, look at comparison in table 1.1.

Getting raw data from the web platform by links on the data sources, or by the web API's do not create the Web of Data, because they do not reply with main principle of the Web. Therefore, we are getting to the term Linked Data. Linked Data according to the definition is a method of publishing structured data so that it can be interlinked and become more useful through semantic queries. It builds upon standard web technologies such as

HTTP, RDF and URL's, but rather than using them to server web pages for human readers, it extends them to share information in a way that can be read automatically by computers. This enables data from different sources to be connected and queried. [4]. The main idea of Linked Data is to publish and next to machine-process data about concrete entities and relations among them on the websites. This method of publishing of data on the Web must correspond to principles of current Web (look at the table 1.1).

What is actually Web of Data? Publishing of big amount of data with use of principles of Linked Data leads to creation of one shared global system of entities, hence Web of Data. Detailed description of principles of Linked Data will be described in the next section 1.2.

1.2 Linked Data Principles

Tim Bernens-Lee who is the founder of Linked Data and Web of Data made up 5-star score system, which defines the quality of the published open data on the Web:

- 1 star - make your stuff available on the Web (whatever format) under an open license1.
- 2 stars - make it available as structured data (e.g., Excel instead of image scan of a table).
- 3 stars - make it available in a non-proprietary open format (e.g., CSV as well as of Excel).
- 4 stars - use URIs to denote things, so that people can point at your stuff.
- 5 stars - link your data to other data to provide context [5].

This system is cumulative, each next star predicts, that characteristics described in the previous stars was fulfilled. If open data have more stars, it means, that data are better. Linked Data represents open data, which are categorized to the highest category, so they come under category of five stars data.

Linked Data is set of principles and technologies for publishing and linking machine-processable data on the Web with the usage of actual Web standards. Term Linked Data also serves for describing of data, which match with these principles. Because Linked Data create global Web database of data, they allow accumulate data from many sources in one step.

Main principles of Linked Data were presented also by Tim Bernens-Lee [6] and they are following:

- Use URIs as names for things.

- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
- Include links to other URIs, so they can discover more things.

In following subchapters are mentioned several principles described in more details.

1.2.1 Principle 1

First principle is about using URI for naming of individual things. If the user wants to work with the concrete entity, it is necessary the entity was identified in some way. In Linked Data, entity (thing) can represents anything, material things, like for instance, books, cars, persons etc., abstract things, for instance, colors, event, relationships among people, evaluation etc., and each thing must be identified by global unique URI. When it was apparent that Linked Data are built on the principles of the classic Web, for identification of the things was chosen URI. It is Universal Resource Identifier, which identifies any source on the Web. URL (Universal Resource Location) is used for web documents, which uniquely determines concrete document, but also it provides its definite location on the Web.

1.2.2 Principle 2

The second principle is connected with the principle number 1, so with the identification of the things. It concretely specifies not to use arbitrary URI on the identification of the things, but HTTP URI. It means, that particular data should be searched by HTTP client with usage of HTTP protocol on the Web based on their URI. Ergo, the user will be able to search in the browser not only document as it is, but also data in the format processed by machine, look at picture 1.1. URI then identifies real things on the one hand, and on the other hand also documents.

1.2.3 Principle 3

Third principle is following. After the HTTP URI entry (identifier of the concrete entity) to the browser, the user has to be informed by providing useful information by the usage of the standards of Semantic Web. In this case useful information are data presented in the RDF language, about which is possible to query with usage of SPARQL language.

If we want the idea of Web of Data as complete global storage of data was real, it is necessary the structured data provided on the Web were represented on the principle of in advance agreed standards. Within the Semantic Web it is

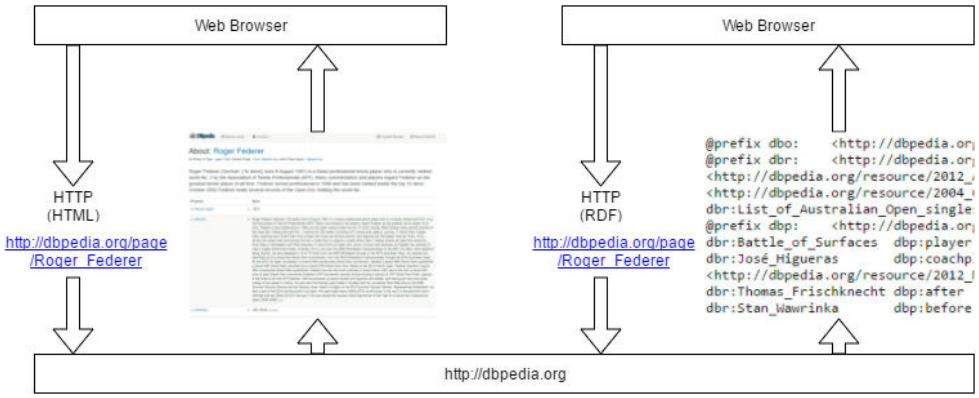


Figure 1.1: HTTP URI for resource "Roger Federer" and accessing RDF data on the Web

technology RDF (Resource Description Framework). Simply, framework RDF is data model, which includes the set of triples. Each triple is defines by three parts: subject, predicate and object. As representation of these three statements together is possible to imagine as a simple sentence, which describes relation between subject and object by predicate, or it describes some feature of the subject. In the figure 1.2 it is possible to read the first triple (called Literal triple) like: *"Roger Federer was born on 8.8.1981"* and the second one (called RDF link) we can read like: *"Roger Federer comes from Switzerland"*.

So, from the figure 1.2 it is visible, that the triples can be classified into two categories based on the type of the object:

- Literal triples – object is value
- RDF links – object is link to another resource

Subject in the triple describes particular resource. It is represented by the URI identifier of the concrete resource. In one triple, it is described by two remained statements in that triple.

Predicate represents either the feature of the object in the case of literal triple, for instance, the date of the birth, color, ID number, etc., or the concrete relation between subject and object, in the case of triple marked as RDF links, for instance, employer, husband, place of the birth in the case, that these things represnt another RDF source. Predicates are also represented by the URI. These URIs of the predicates are defined in the various dictionaries, which are used for the description of the concrete type of data. The user can define dictionaries by himself, but he has to observe some rules for the description of the ontology and predicates. In order to data could be easily processed by machine in the Web, there should be preferred standardized and mostly used predicates before own defined predicates. To the most

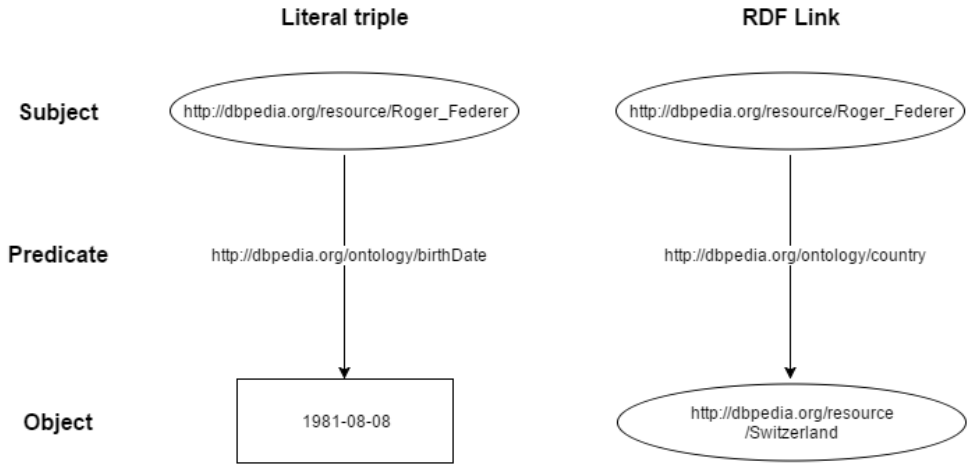


Figure 1.2: Example of basic kinds of RDF triples

spread dictionaries belong, for instance, Dublin Core, Good Relation, FOAF, schema, SKOS, etc. Section 1.4.1.2 is dedicated to dictionaries.

In the case of literal triples, object in the triple represents the concrete value represented by some literal. Therefore the triples are referred to as literal. The literal can be shown in various data types, for instance, string, date, number etc. Literal values are not represented by the identifier URI, they are only ordinary values. In the case of triples marked as RDF links, object represents not only the relation to the subject in the actual triple, but also represents another resource, hence this type of the object has to be identified also by the URI.

More details about technology RDF, which describes data model for Linked Data is in the separate chapter 1.4.1.1.

RDF framework does not represent format, in which are Linked Data written, but data model, which is used for the description of these data. By the usage of this data model, it is necessary to serialize the data. Various formats exist for serializing of the data described by RDF data model. The most used are RDF/XML, Turtle, N-Triples or JSON. Detailed description of the particular RDF formats is in the section 1.4.1.3.

The main idea of the principle 3 in Linked Data is to use standardized technology for the description of the data, so RDF data model and these data are necessary to serialize into particular format, in order to be processed by the machine, and in order to provide useful information to the users.

1.2.4 Principle 4

Following principle describes, that at the particular resource is appropriate to include links to the other RDF resource for the possibility of searching more information about the concrete resource. As the Web of documents is aimed on publishing hyperlinks, in order to the user could move from one document the other similar document, also it is necessary for Web of Data to reply this principle. Then the user can from one resource get to another, similar resource. RDF links which are used in Linked Data can be possibly divided into three main categories:

- Relational Link – it is the case, where the object represents another entity like is subject in triple, and from this object we can receive more information through this link
- Identity Link – it is subset of Relational Link, the object creates the same entity like the subject, but for instance, it contains different information in the different datasets
- Vocabulary Link – it is the link, which refers to the definition of the feature in the concrete dictionary, which was used for the definition of the feature in the particular triple between subject and object.

In the figure 1.3 are shown these three types of links. As we can see, from one resource can go out more links and from object, which is another resource can go out another links. In this example vocabulary link represents definition of the feature *givenName* described by the dictionary FOAF. Relational link in this figure represents relation between the resource *Roger Federer* described by DBpedia and the resource *Switzerland*, also described by DBpedia. Identity link in this picture says, that the resource *Roger Federer* described by DBpedia represents the same entity like resource *7341827877990687 4903* described by data.nytimes.com. Relational or identity link can be also the vocabulary link in one time.

The end of the fourth principle is clear. If we want the data really reflect the definition of Linked Data, it is necessary to use all types of above mentioned links. Users also should be able to surf through the particular entities like as they surf through documents. In conclusion, it is important the Web of Data really represents one global database of all resources.

1.2.5 Conclusion

After the introduction of the particular principles of Linked Data it is possible to deduce, that it is possible to realize Web of Data with the observance of principles in the way, that principles of the actual web, which is Web of documents will be remained, look at table 1.2. Linked Data then meet all principles of the Web: data on the Web described by the uniform data model

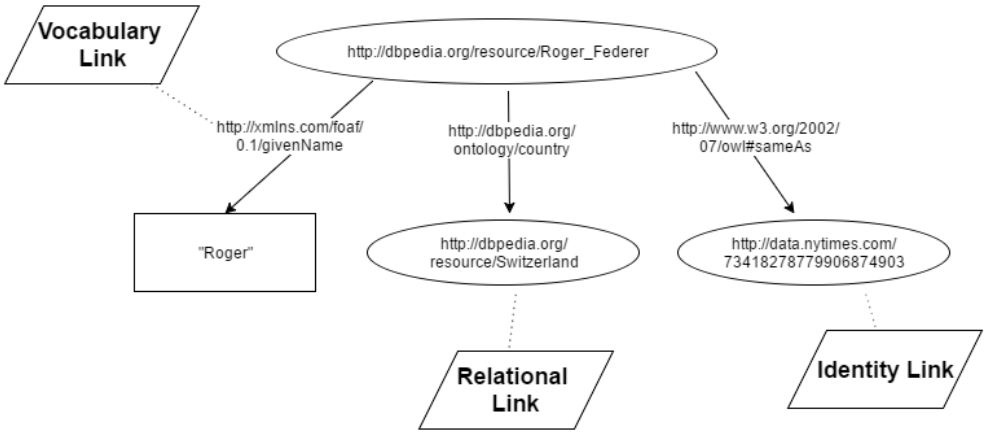


Figure 1.3: Different types of links in RDF data model

Web of documents	**Web of Data**
Uniform format HTML for documents publishing	Uniform RDF data model for data publishing
URL as unique global identification of document	URL as unique global identification of resource
HTTP protocol for localization and access to documents	HTTP protocol for localization and access to resource
Hyperlinks between documents	Hyperlinks between resources

Table 1.2: Comparison: Web of documents vs. Web of Data using data in RDF Data model

RDF (principle 3), particular resources have clear identification (principle 1), it is possible to access to them by the HTTP protocol (principle 2) and they contain links to the similar entities (principle 4).

1.3 LOD Cloud

Following section describes what LOD Cloud represents, how it was created, developed and also how its current state is.

Linked Open Data Project is a community activity started in 2007 by the W3C's Semantic Web Education and Outreach (SWEO) Interest Group. The Project's stated goal is to make data freely available to everyone [7].The result of this project is global data space, which is called Web of Data. Web of Data is shown in the huge graph, which consists of trillion RDF triples from various datasets covering different areas, like, for instance, geographical data, government statistic, people, media, movies, music, medical data,

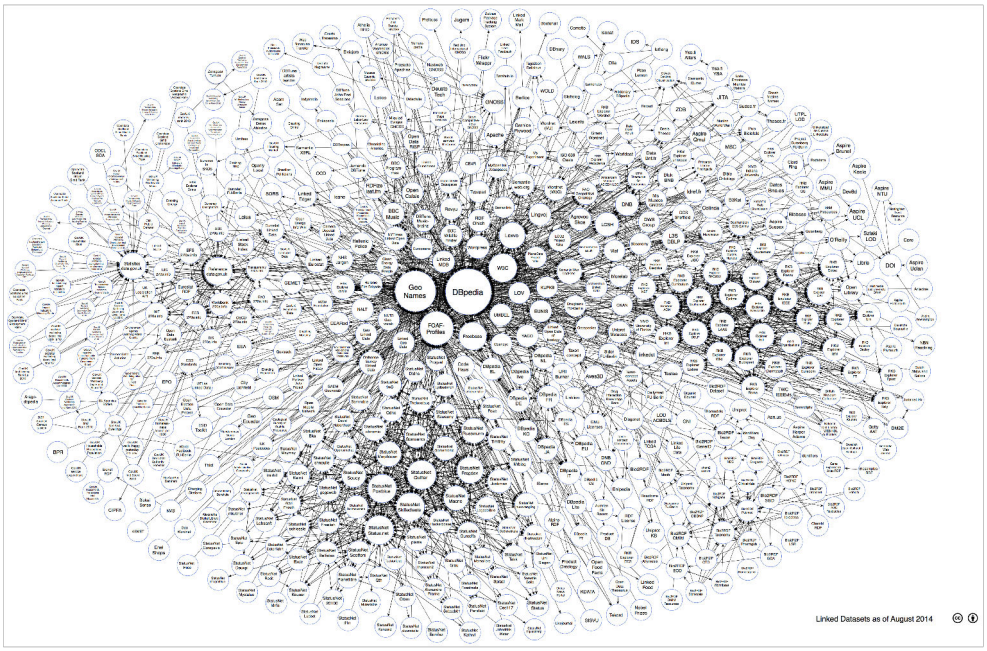


Figure 1.4: Linking Open Data cloud diagram 2014 [1]

statistical data, etc. LOD cloud shows datasets that have been published on the base of Linked Data principles, by contributors to the Linking Open Data community project and other individuals and organisations. It is based on metadata collected and curated by contributors to the Data Hub as well as on metadata extracted from a crawl of the Linked Data web conducted in April 2014 [1]. LOD Cloud diagram from the year 2014 is shown in the picture 1.4.

Each dataset, which is concluded or which will be concluded in LOD Cloud has to meet these criteria:

- Dataset must be resolvable by http:// (or https://) URIs (represents principle 1 in the section 1.2.1 and principle 2 in the section1.2.2).
- Dataset must be represented by RDF language in, at least one from the following formats: RDFa, RDF/XML, Turtle, N-Triples (represents principle 3 in the section 1.2.3).
- Dataset must contain at least 1000 triples.
- Dataset must be connected with RDF links with at least one dataset, which is currently occurring in the LOD Cloud. There is requested at least 50 links (represents principle 4 in the section 1.2.4).

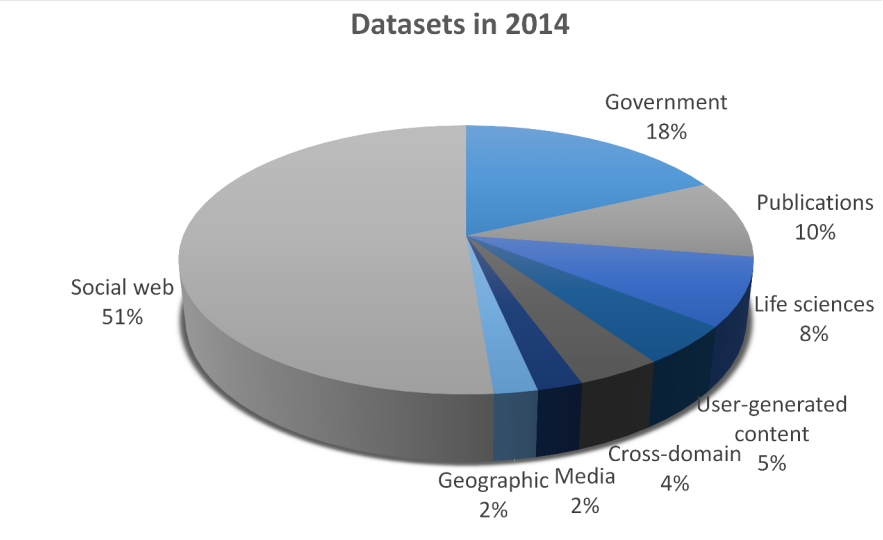


Figure 1.5: Percentage representation of datasets in particular categories of LOD Cloud in year 2014

- Dataset must be available via RDF crawling, via an RDF dump, or via a SPARQL endpoint.

Number of datasets in LOD Cloud increases every year. LOD Cloud was created in the 2007 with the initiative 12 datasets. In the 2009 it contained 95 datasets and in 2011 there was 295 datasets. Currently, LOD Cloud contains 1014 datasets, which follow principles of Linked Data and conditions of Linked Open Project.

Datasets in LOD Cloud are now divided into 8 main categories. More than half of LOD Cloud consists of datasets in the category *Social Web*. In the previous versions of LOD Cloud this category was not created at all, and today it makes the biggest group. The following category is area *Government* with 183 datasets. The third place takes category *Publications* with 96 datasets, which made the biggest group in year 2011. Actual percentage representation in particular categories in LOD Cloud from the year 2014 is shown in the chart 1.5. Comparison of the number of datasets in the individual categories in the year 2011 and 2014 is shown in the table 1.3.

The biggest datasets in LOD cloud are DBpedia a GeoNames. DBpedia is dataset in the category Cross-Domain. It is project, which was created with the aim to extract structured data from Wikipedia and publish them on the Web as Linked Data. The English version of the DBpedia knowledge base currently describes 4.58 million things, out of which 4.22 million are classified in a consistent ontology (http://wiki.DBpedia.org/Ontology2014), including 1,445,000 persons, 735,000 places (including 478,000 populated places),

Topic	Datasets in 2014	Datasets in 2011
Government	183	49
Publications	96	87
Life sciences	83	41
User-generated content	48	20
Cross-domain	41	41
Media	22	25
Geographic	21	31
Social web	520	0

Table 1.3: Comparison of categories represenation of datasets in LOD Cloud from the year 2011 and 2014

411,000 creative works (including 123,000 music albums, 87,000 films and 19,000 video games), 241,000 organizations (including 58,000 companies and 49,000 educational institutions), 251,000 species and 6,000 diseases [8]. DBpedia forms the biggest dataset in the Web with the 3 trillion triples and 50 million RDF links, which connect DBpedia with other datasets. GeoNames is dataset from the category Geographic and it contains more than 11 million of several places. There exists the online application, in which it is possible to search places according to the name of country, postal codes etc. Dataset GeoNames is also described by own ontology which is free avilable in the Web in GeoNames documentation , which is available at [9].

1.4 Semantic technologies

This chapter is describing the main semantic technologies, which are used for the creation, description and querying Linked Data and for the creation of Linked Data applications too. It will contain the description of RDF data model, and also RDF formats and RDF dictionaries, which are connected to it. Furthermore, this chapter contains description of language SPARQL, which functions is querying Linked Data, introduction with the project Linked Data Fragments and also with the format HDT, which is compression format of RDF data.

1.4.1 RDF

RDF is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed [10].

Since it was mentioned above, when introducing with the issue, RDF is a framework, which main function is representation of Linked Data. It

describes data model, which contains the concept of triples, which consists of subject, object and predicate and this triple describes resource and its feature.

RDF framework enables the realization of Web of Data, so it allows accomplish all four principles of Linked Data, which were described in the section 1.2. The first principle says, that is necessary to use URI for the description of the particular resource. RDF allows this principle, because in RDF, only URI is used for identification of things. The second principle tells us, that for the identification of the things it has to be used HTTP URI to be visible on the Web. RDF enables using of any URI, for instance FTP URI, IBAN URI, own URI, but also it allows usage of HTTP URI, which is necessary for the accomplishing the second principle of Linked Data. The third principle says, that it is necessary to provide to the user useful information. RDF accomplishes this principle in the way that it provides information to user by human and also by machine processed RDF format through web browser, because it allows to use HTTP URI. The fourth principle reports, that in Linked Data it is important to linking data with similar one. RDF accomplishes this principle for example in the way that objects in triples can be link to another resources identifiable by URI, and that predicates and subjects are defined by URI, which refers to related information.

Unlike the other formats for publishing structured data on the Web, RDF allows to represent Linked Data in the way, that there will be maintained all required principles.

1.4.1.1 RDF Data Model

In this chapter the user will be introduced with the main elements of RDF data model for the description of Linked Data. All these examples were made after studying RDF data model in the book [11].

Blank node Piece of information about some entity in Linked Data is described by one triple: subject, predicate, object. In the section 1.2.3 was mentioned, that object is represented either by another resource, that is identified by HTTP URI, or by literal. One other possibility exists and that is blank node. Blank node is basically object without URI, which is used for dividing one predicates to several more specific predicates, and the user does not want to create abstract URI for the collection of objects. Ideal example for this case is representation of address. The address is composed of, for instance, street, city, country. It is possible to use blank node as object, which represent whole address, from which will come out links on the particular entities of addresses, look at the figure 1.6. Some database tools, which are searching through Linked Data, for instance, SPARQL, automatically create URI for blank node. Blank node is marked by "_", so instead of URI is used mentioned mark.

Typed Literals Next component in RDF is usage of typed literals. Up to now, in the examples mentioned above, there were objects, which were either URI of another subject, or they contained value in the version of literal. There also exists the possibility of the literal, at which is even defined data type, for instance, string, integer, timestamp, etc. Furthermore, there is possible to define not only the data type at literals, but, for instance, at data type string there is possible to add also the language, in which is the particular string written. Data type in Typed Literal is written in the way, that after the particular value of literal is written a double caret symbol followed by URI for particular data type, as you can see on the figure 1.6. The language of the literal is denoted in the way, that after URI of data type is written the mark @ following by abbreviation of particular language. All data types applicable for typed literals in RDF are described in the dictionary W3C XML Schema Definition Language [12].

Classes and subclasses In RDF is also possible to use classes, that means to determine, that some subject is the instance of some concrete class. RDF concept *rdf:type* is used for this. This predicate can be applied also in the short form, in the form of the char *a*. Classes in RDF are possible to comprehend like classes in the object oriented programming. Members of the class (objects assigned to the concrete class by predicate *rdf:type*) can be represented like instances of particular class. The class in RDF can be created by the usage of predicate *rdfs:Class* or *owl:Class*, look at the picture 1.6. Also it is possible to define subclasses, for instance by the predicate *rdfs:subClassOf*. RDFS (RDF Schema) [13] is the main dictionary for the description of RDF ontology.

1.4.1.2 RDF Vocabularies

Unfortunately, RDF framework does not provide general overview of domains or definition of classes, which would be the reflection of the real things in the world. It does not even define relations between particular things. Therefore, dictionaries were made, where each dictionary deals with particular domain or area, which it describes. RDF Vocabularies are used for the definition of predicates, which are used for connecting subject and object in the triple, but also for the definition of the classes and ontology. In the case of creation own dataset, it is ideal to use predicates from as many already existed dictionaries as possible, so data could be easier translated and better processed, by machine or also by human. It is also possible to create own dictionary, which has to fulfill set of criterias for dictionaries representing Linked Data.

Currently, there exists big amount of dictionaries for the description of various areas, for instance, for the description of conceptual models, hierarchy, metadata, persons, geographical areas, places, consumer products etc.

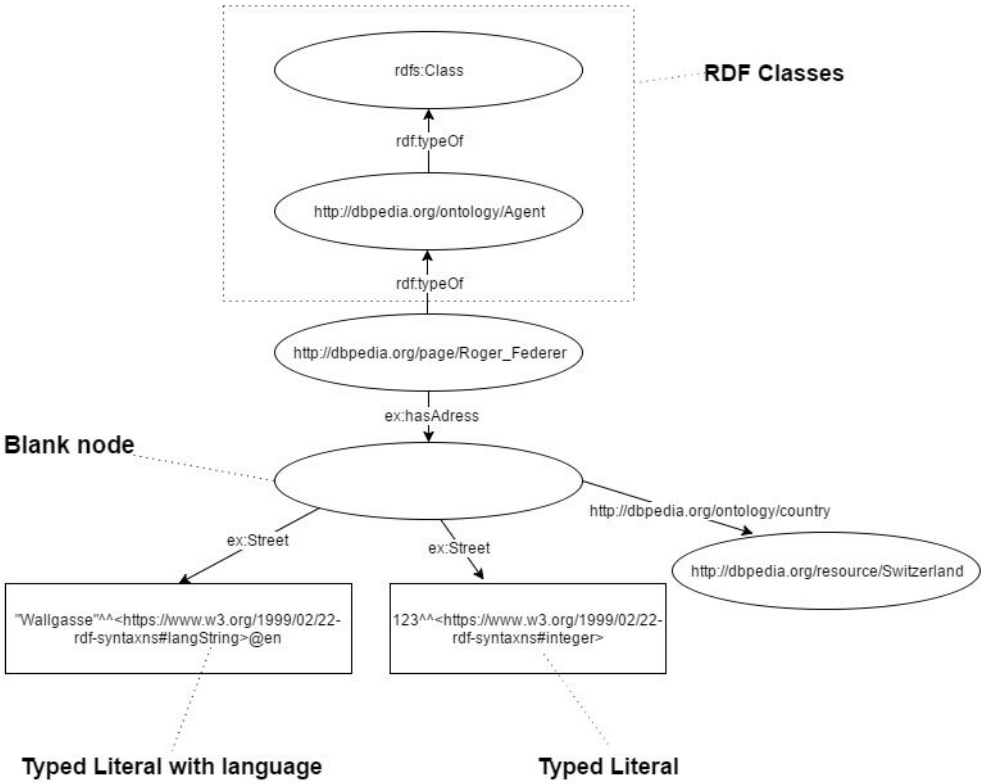


Figure 1.6: RDF example of blank node, typed literals and usage of classes

RDF and RDFS The dictionary RDFS (RDF Schema) is the main RDF dictionary, and it is used mainly for the defining taxonomy of classes and predicates, for the creation of members to the particular classes, but also for providing of more information about subject. Official definition says that it provides a data-modelling vocabulary for RDF data and it is an extension of the basic RDF vocabulary [13].

RDFS uses two domains, one for the description of the classes, and the second one for the description of the predicates:

- *http://www.w3.org/1999/02/22-rdf-syntax-ns#* - with this domain it is used prefix rdf, and one of the basic classes is *rdf:predicate*, which describes all RDF predicates.
- *http://www.w3.org/2000/01/rdf-schema#* - with this domain it is used prefix rdfs, and one of the basic classes is *rdfs:Class*, which describes all RDF classes.

To the most basic predicates of RDFS dictionaries belong:

- rdfs:Class - defining of the concrete class.
- rdf:Property - defining of the concrete property.
- rdf:type - defining of membership to the class.
- rdfs:Resource - class of all resources. All other classes are subclasses of this class.
- rdfs:subClassOf - defining of subclass to the concrete class.
- rdfs:subPropertyOf - defining of subpredicate to the particular predicate.
- rdfs:comment - comment readable by the users.
- rdfs:seeAlso - link to get further information about resource.

OWL Dictionary OWL (Ontology Web Language) is similar to dictionary RDFS, because it also defines data model, so the ontology, classes and their predicates. According to definition, OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web and id developed as a vocabulary extension of RDF [14]. To its elementary classes belong, for instance, *owl:Class*, *owl:Thing*, *owl:ObjectProperty* or *owl:Restriction*. But the most used predicate from this dictionary is *owl:sameAs*, which is used to claim, that two resources are identical. Based on this predicate, the user knows that it is one and the same resource, only described in various ways in different datasets.

SKOS SKOS dictionary (Simple Knowledge Organization System) is a W3C recommendation designed for representation of thesauri, classification schemes, taxonomies, subject-heading systems, or any other type of structured controlled vocabulary [15]. SKOS is hence similar to the dictionaries RDFS and OWL, but when RDFS and OWL deal with the conceptual model of classes and their predicates, SKOS is more concerning with conceptual hierarchy and collections.

To the most elementary classes and predicates of this dictionary belong:

- skos:Concept - main class of SKOS dictionary which is used for the definition of the concept.
- skos:ConceptScheme - class which aggregates more classes of type *skos:Concept*.
- skos:inScheme - predicate for the insertion of the concept to the scheme of concepts.
- skos:hasTopConcept - predicate for the specification of the main concept in the particular scheme of concepts.

- skos:prefLabel - predicate for the preferred concept name, for one concept it can be only one in one language.
- skos:altLabel - predicate for the alternative concept name, for one concept it can be more in many languages.
- skos:hiddenLabel - predicate for the hidden title because of potential typing error during searching.
- skos:notation - string of characters for the unique identification of the concept in the particular scheme of concepts.

DCAT,VoID and DCMI These dictionaries describe mostly datasets, linkset and metada about them.

Dictionary DCAT (Data Catalog Vocabulary) is used mostly for the description of catalogues, datasets and themes in the particular datasets. It contains classes such as *dcat:Catalog* for the defining of catalogue, *dcat:Dataset* for the defining of dataset or *dcat:Keyword* for the description of the key words in discribed dataset.

For the description of datasets and linksets there is mostly used the dictionary VoID. VoID is an RDF Schema vocabulary for expressing metadata about RDF datasets [16]. In this concept, the dataset is represented as set of RDF triples, which are published, maintained and aggregated by one provider. Linkset is represented as the collection of RDF triples, which describes link between two resources in two various datasets. Dictionary contains classes and predicates such as *void:Dataset* for definition of dataset, *void:Linkset* for definition of linkset, *void:triples* for number of triples in dataset, *void:classes* for number of classes in dataset etc.

Main function of dictionary DCMI (Dublin Core Metadata Information) is description of metadata, for instance, about datasets, linksets, concepts etc. To the most basic predicates belong *dct:titul* for the description of the name of dataset, linkset, concept, etc, *dct:language* for the description of language, in which specific concept is written or *dct:publisher* for defining the user, who publishes concrete dataset.

1.4.1.3 RDF Formats

In the beginning, it is necessary to realize that RDF is not data format but a framework which defines data model for the description of data in the format of triples: subject, predicate, object. With the aim to save data to the file or on the Website and process them by machine, it is necessary to serialize RDF data model into one of RDF formats. All allowed RDF formats represents RDF data model, and data serialized in various formats can be connected to one RDF graph. This flexibility enables to use format on serialization

of data, which is the most suitable for developer, for instance, for Web applications is the most used format JSON, because Javascript provides many suitable libraries supporting JSON. Applications, which are based on XML rather use format RDF/XML etc. In the next subchapters will be described the most used RDF formats with the examples and advantages/disadvantages of their usage.

RDF/XML Format RDF/XML is original format for the serialization of RDF data model, which is standardized by organization W3C. This format, however, is not completely simply readable by the user hence it is not used very often. This format, of course, contains XML tags, and particular components of triple, that means subject, predicate, object can be divided to XML tags, XML attributes and XML content. Abbreviations for namespaces are defined by XML attribute in XML tag rdf:RDF, look at example in the listing 1.1.

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
    syntax-ns#"
          xmlns:rdfs="http://www.w3.org/2000/01/rdf-
              schema#"
          xmlns:ns0="http://dbpedia.org/ontology/">

  <rdf:Description rdf:about="http://dbpedia.org/
      resource/Roger_Federer">
    <rdf:type rdf:resource="http://dbpedia.org/ontology/
        Agent"/>
    <rdf:type rdf:resource="http://dbpedia.org/ontology/
        Person"/>
    <rdfs:label xml:lang="es">Roger Federer</rdfs:label>
    <ns0:residence rdf:resource="http://dbpedia.org/
        resource/Switzerland"/>
  </rdf:Description>

</rdf:RDF>
```

Listing 1.1: RDF/XML format for representation of RDF Data

RDFa RDFa is the format which is also standardized by organization W3C. It is the format, which embeds RDF data directly on HTML websites, not by HTML comment, but normally by HTML DOM elements. It means that to arbitrary existing HTML website is possible to add structured data with the usage of RDFa, and HTML website will become not only ordinary Website with the unstructured content, but also with structured data suitable for the processing by machine. It works easily, it is necessary to change DOCTYPE in HTML document to support RDFa format, and based on this change,

web browser by processing of HTML just ignores elements with attributes which it does not know.

Turtle Turtle is format, which can be in the most unique way converted on RDF data model, because it is the most intuitive and the easiest for understanding by the user. Turtle is derived from the Terse RDF Triple Language and is the most often used for serialization of RDF data model by the user. It provides support for the prefixes, which are mostly defined in the initial part of the serialization, therefore particular components of triple do not have so long names, ergo the file is more transparent and easier for understanding. Besides, it is not always necessary to write the whole triples. At triples, which share mutual subject and predicate are particular objects separated by comma. Triples which share only the subject are divided by semi-colon, look at the example shown in the listing 1.2.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ns0: <http://dbpedia.org/ontology/> .

<http://dbpedia.org/resource/Roger_Federer>
  a~<http://dbpedia.org/ontology/Agent>, <http://dbpedia
      .org/ontology/Person> ;
  rdfs:label "Roger Federer"@es ;
  ns0:residence <http://dbpedia.org/resource/Switzerland
     > .
```

Listing 1.2: Turtle format for representation of RDF Data

N-Triples Format N-Triples is subset of format Turtle minus prefixes and abbreviations. It is basically more primitive format, because it describes every triple apart and without prefixes, so with the whole URI, which causes high redundancy. Therefore files in format N-Ttriples have much bigger size like files, for instance, in format Turtle. In spite of this redundancy this format is processed by machine in the best way, as it has no optimalization and every triple is written separately, see example in the listing 1.3.

```
<http://dbpedia.org/resource/Roger_Federer> <http://www.
    w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia
    .org/ontology/Agent> .
<http://dbpedia.org/resource/Roger_Federer> <http://www.
    w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia
    .org/ontology/Person> .
<http://dbpedia.org/resource/Roger_Federer> <http://www.
    w3.org/2000/01/rdf-schema#label> "Roger Federer"@es .
```

```
<http://dbpedia.org/resource/Roger_Federer> <http://
    dbpedia.org/ontology/residence> <http://dbpedia.org/
    resource/Switzerland> .
```

Listing 1.3: N-Triples format for RDF Data

1.4.2 SPARQL

As well as there exists query language SQL for data saved in relation database, also exists query language for Linked Data serialized to some RDF formats, and that is SPARQL language. SPARQL is based on standards of SQL and is defined by organization W3C. Whole SPARQL specification is available in [17].

1.4.2.1 SPARQL components

Structure of SPARQL query is very similar to SQL query, but it is assimilated to RDF data model. It contains, for example, components PREFIX, which in SQL is not defined. Furthermore, WHERE clausule is not written in the form of condition like in SQL, but in the form of triple patterns, based on which the result will be evaluated.

Individual components in order of SPARQL query are following:

- PREFIX ...
- SELECT — DESCRIBE — ASK — CONSTRUCT ...
- FROM ...
- WHERE ...
- ORDER BY ... LIMIT ... OFFSET

PREFIX is used for defining namespaces, which are used in SPARQL request. The main advantage is, that the user or machine does not have to write the whole titles in defining triple patterns, but it uses only abbreviations defined in this clausula. On the Web is available the good application for searching of namespaces, which the user needs to use in SPARQL query, or in the creation of datasets. It is available on the website [18].

There exist more types of SPARQL queries, which are classified according to the result. The most known is SELECT, which returns the set of required variables corresponding to triple pattern defined in WHERE clausula. It can be represented as classic SELECT, which is known from SQL language. The other possibility is the usage of DESCRIBE, which answers on the question: “What all do we know about particular resource?” DESCRIBE query can contain only URI, where the result will be the RDF graph describing the whole

resource. It can also contain clausula WHERE, in which the user defines subset of dataset, which he needs to display, look at the example in the listing 1.4.

```
DESCRIBE <http://dbpedia.org>

DESCRIBE ?s

FROM <http://dbpedia.org>

WHERE { ?s a~<http://dbpedia.org/ontology/Agent>}
```

Listing 1.4: Example of DESCRIBE query for whole dataset and for subset of dataset

The next type of SPARQL query is ASK, which answers on the question: "Exists at least one triple quering to particular triple pattern in this dataset?" The result is either true or false.

The last type is CONSTRUCT, which on the base on defined scheme (exactly in the clausula CONSTRUCT) creates corresponding RDF graph. In the example listing 1.5 are firstly shown triples, which correspond to triple pattern in WHERE clausula, and the result is transferred to the form of triple defined in CONSTRUCT.

```
CONSTRUCT { ?s my:name concat(?gn, " ", ?sn) }

FROM <http://dbpedia.org>

WHERE { ?s foaf:givenName ?gn; foaf:surname ?sn.}
```

Listing 1.5: Example of CONSTRUCT query

SELECT chooses the result, which corresponds to conjunction of all triple patterns in WHERE component. SPARQL also provides the possibility to define patterns, which are OPTIONAL, so in the case of equality they will get to the result, and in the case of discrepancy, the original result is not changed. Furthermore, same as language SQL, SPARQL allows function like UNION for the unity of more resources or MINUS for the finding contrasts between graphs.

For result limitation, it is also possible to use component FILTER, which filtrates the results according to assignment. It contains operators and functions and it is applying on the whole result according to pattern in WHERE component. For the limitation of results there works also classic functions like DISTINCT, OFFSET, LIMIT, etc. Also there are available aggregation functions as COUNT, SUM, AVG, MIN, MAX, etc., logical functions as IF, functions for the handling text and numeric values. All available functions are defined in SPARQL specification.

In SPARQL also exists federated queries, which are used for receiving additional data from other dataset. For this type of query is used SERVICE component.

As well as SQL, SPARQL is not used only for querying data, but also on the creation and editing of data. To main functions of data update belong:

- INSERT DATA - it uses no patterns, no variables, there are directly entered concrete data which user wants to insert.
- DELETE DATA - it uses no patterns, no variables, user directly entered concrete data to be removed.
- INSERT - it is used for inserting data from some dataset which corresponds to pattern in WHERE component.
- DELETE - it is used for deleting data from particular dataset which correspond to patterns in WHERE component.
- LOAD - it is used for the loading data from file to graph.
- CLEAR - it is used for deleting of all triples in graph.

1.4.2.2 Querying Data

Linked Data, about which the user needs to query, are available either on the Web, or locally in the file. Many datasets available on the Web provide SPARQL endpoint, which is web-accessible query service, which accepts SPARQL language. Specific HTTP GET mostly shown website with HTML form for typing the query, plus some additional possibilities, for instance, format of result, execution timeout, etc. In the figure 1.7 there is displayed SPARQL endpoint for DBpedia available on the address: http://dbpedia.org/sparql.

In the case, that the user needs to use SPARQL language for locally stored data, it is necessary to use some tool for querying RDF data or to use Triple Store with the support SPARQL, which is used for backup and querying RDF data. To the most known tools for querying Linked Data belong:

- OpenLink Virtuoso [19] - it provides HTTP server involving SPARQL Endpoint. There exists commercial version, but also open source single server.
- OpenRDF Sesame [20] - open source framework for RDF and Linked Data, which allows store RDF data and querying them.
- Apache Jena [21] - open source JAVA framework for the creation of semantic Web and Linked Data, it enables to store data also to create and querying data. It provides also the tool for the single querying, which is ARQ SPARQL 1.1 – compliant engine.

Figure 1.7: Virtuoso Sparql Query Editor

1.4.3 Linked data fragments

According to the definiton, Linked Data Fragments is a conceptual framework that provides a uniform view on all possible interfaces to RDF, by observing that each interface partitions a dataset into its own specific kind of fragments. A Linked Data Fragment (LDF) is characterized by a specific selector (subject URI, SPARQL query, ...), metadata (variable names, counts, ...), and controls (links or URIs to other fragments) [22].

The main idea of the project Linked Data Fragments is to unify all resources of Linked Data, and to provide to the user the possibility to querying about data in the most effective way, and even balance the load pressure between client and server part.

Linked Data fragments currently distinguish three resources of Linked Data, about which it is possible to querying:

- data dump - all triples included in the particular dataset
- subject page - it contains triples about specific subject in dataset shown on the webpage
- SPARQL result - it contains triples, which are result of SPARQL query

But every Linked Data interface, which was mentioned above, make full use of either client or server part. SPARQL result is for the client, of course, the easiest, but on the other hand, the server part has to make so much effort

in order to maintained high accessibility of data. If the user does not want to rely on this SPARQL endpoint, he can download data to local directory, and he processed them independently, which, on the other hand, extremely makes full use of client part. Therefore the new resource of Linked Data was invented, and that is Tripple Pattern Fragment, which requires minimal server effort and it provides effective client data querying. It is made from these parts:

- data - all triples corresponding to entered query,
- metadata - approximate number of triples,
- controls - queries linking to other fragments of particular dataset.

Linked Data fragments provide currently 23 datasets, about which it is possible to query by the usage of fragments. Preview of all datasets is available on [22].

Linked Data Fragments provides open source software. It offers client programs, which can execute queries about datasets, for which there exist Triple Patterns Fragments. It provides client application in languages Javascript, Java, Perl and Python. The example of online client developed by Linked data Fragments is shown in the figure 1.8. Also there are available server parts, in the languages Javascript, Java, Perl, Ruby, Python, Java and Netkernel. Unfortunately, in the time of implementation of our web application, Java client was broken and we could not use it.

1.4.4 HDT

According to the definition, HDT (Header, Dictionary, Triples) is a compact data structure and binary serialization format for RDF that keeps big datasets compressed to save space while maintaining search and browse operations without prior decompression. This makes it an ideal format for storing and sharing RDF datasets on the Web [23].

HDT is then compression binary format for RDF Data. It occupies a lot less place like any RDF format, it is used on sharing of data on the Web, it is suitable for various analyses and visualization of data. For the comparison, dataset DBpedia from the year 2015 contains more like 800 million triples, and it has 4,7 GB, what is almost three times less than the whole DBpedia in the format turtle.

HDT provides globally 20 datasets, together more than 7 trillion triples in data size less than 50 GB, which would in format N-Triples occupies more than 1 TB. List of all datasets available in HDT format is placed in [23].

HDT provides GUI tool, which is called HDT-it and it enables to load RDF Data in HDT format, also to browse them and querying them. As well, it provides graphical view on data and some metadata about particular dataset.

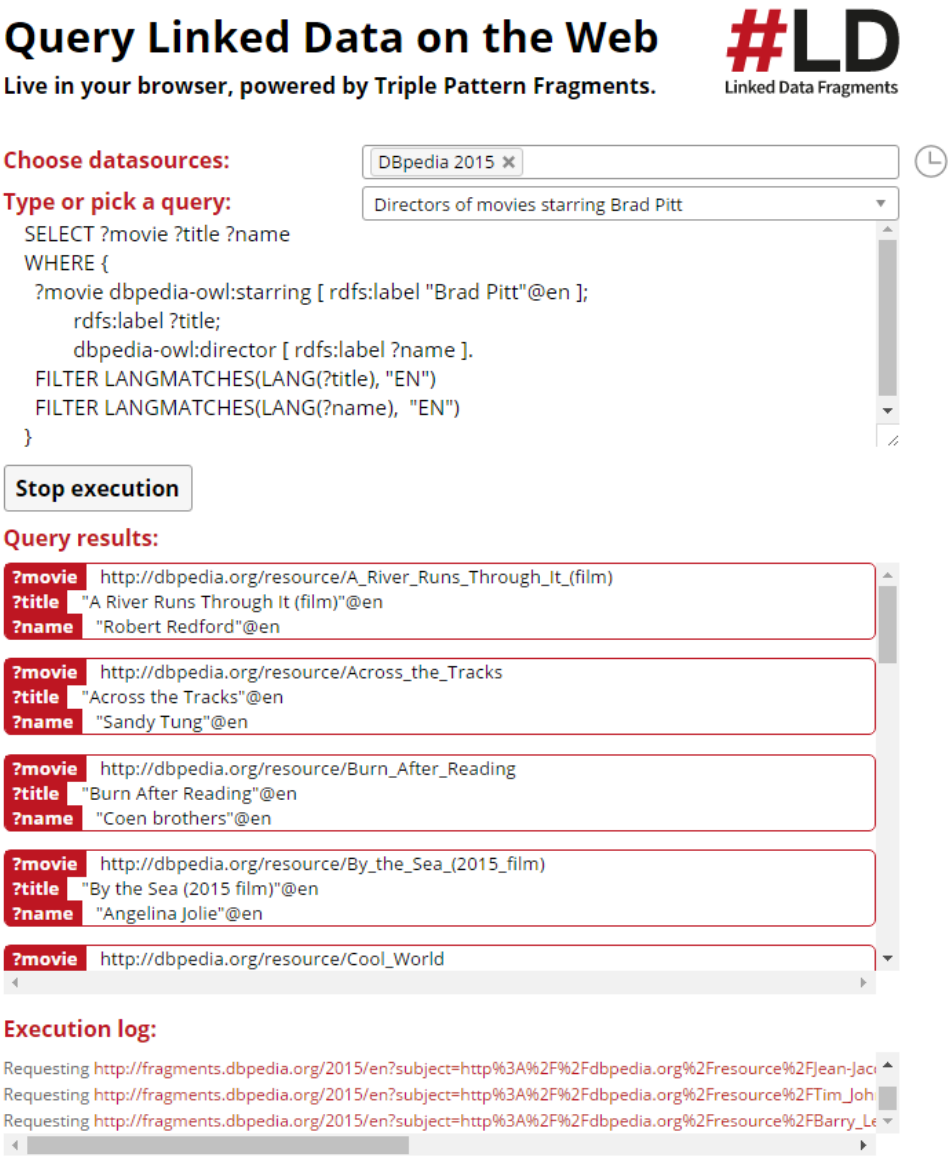


Figure 1.8: Example of Javascript client querying data in Linked Data Fragment format

It is capable of exporting HDT data to some no compress RDF format too. This application is available on the platform of Windows, Linux and Mac OS. Besides, it is possible to download HDT libraries which facilitate to work with the format HDT. Libraries are freely available either in the language C++ or Java, and it is possible to use them for the development of own applications.

Chapter 2

State-of-the-art

Because of the fact that Linked Open Data and LOD cloud are spreading more and more every year, a need for the summarization and various statistics of already existing datasets arose. A necessity to know which data are contained in particular dataset came into being. There is many various tools for exploring of LOD data. Several tools provides various statistic information about dataset content, next tools are focused on user friendlier possibilities of dataset querying and there exist also tools, which provide information to which domain particular dataset belongs, which means what type of information dataset contains.

To the statistical tools belong for instance tools as RDFStats [24], LODStats [25] or proLOD++ [26], which provide metadata specification about dataset. RDFStats generates RDF file with the statistical data about dataset by the running of the default SPARQL queries. Then the results is URI histogram over URI subjects or number of blank nodes. LODStat provides statistical information as the number of triples, subjects, predicates, objects, used links, blank nodes etc. Furthermore it offers also the most occuring vocabularies, data types or languages in datasets. ProLOD++ provides similar statistical information as LODStat, but it displays them also graphically.

One of the available tool providing visual querying of LOD datasets and providing various informations of the particular dataset is called LODeX. LODeX supports visual querying of a LOD source on the basis of the Schema Summary. The main functionalities are a new interface that allows to compose a visual query browsing the Schema Summary of a source; a SPARQL compiler automatically produces and submits the corresponding SPARQL query to the SPARQL endpoint and the result is shown in a tabular view and a refinement panel that allows to refine the visual query by adding or removing attributes, by defining some filter and ordering conditions [27]. Example of visual query by LODex is shown in the figure 2.1. The main idea of this project is that it is not simple to query dataset of unknown structure. Because of that this tool processing SPARQL endpoint input and producing a

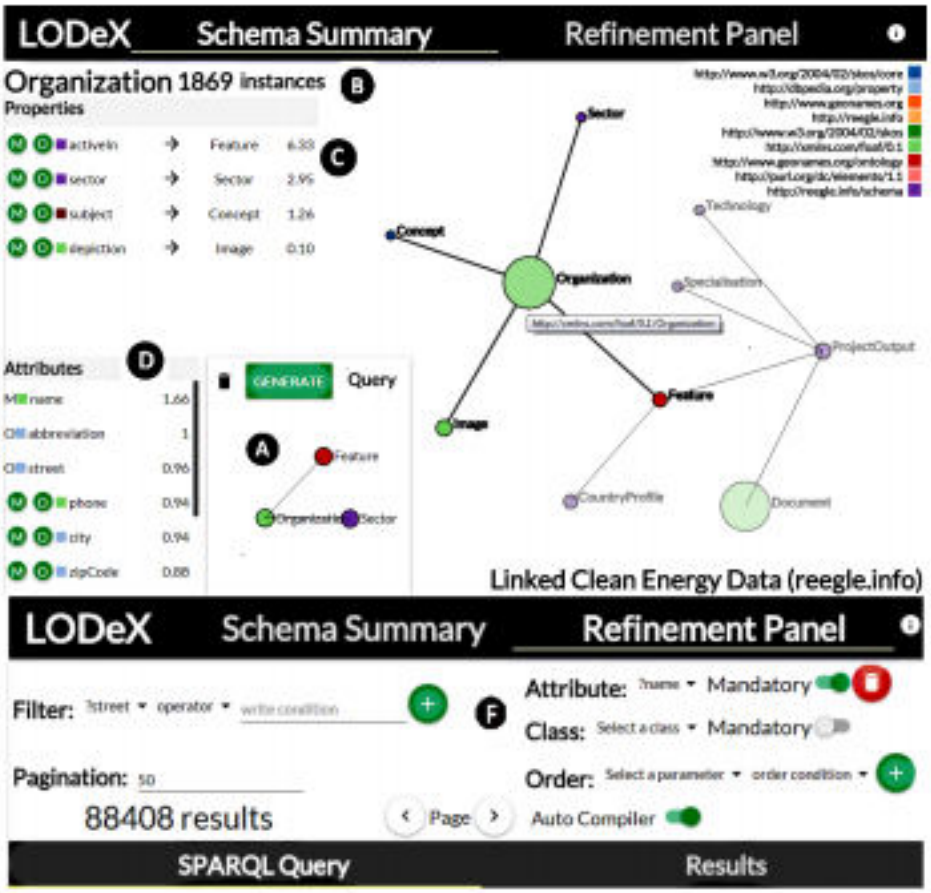


Figure 2.1: LODeX - example of visual query and Schema Summary

set of default queries for creation of Schema Summary over it. The tool shares the idea of our goal which is exploring of datasets we do not know but from the different point of view. Our goal is to determine which domains are represented by dataset while LODeX provides calculated statistic data and is more focused to optimization and answering of queries.

The other tool determined for the sumarization of LOD dataset is for instance LODSight. LODSight is a tool that shows a summary of an RDF dataset as a visualization of a graph formed from classes, datatypes and predicates used in the dataset. The visualization should allow to quickly and easily find out what kind of data the dataset contains and its structure. It also shows how vocabularies are used in the dataset [28]. On the base of this graph it is possible to guess, which data contains the particular dataset. It is on the base of its classes and predicates. However, the tool also does not

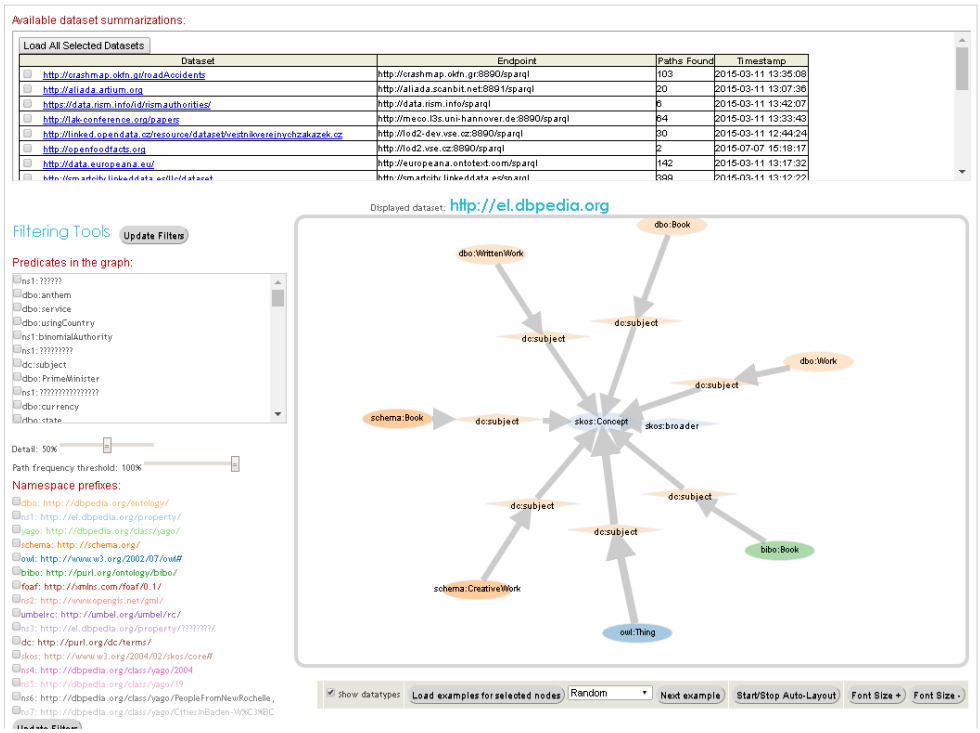


Figure 2.2: LodSight - example of the dataset summarization

provide domain representation of the dataset. Example of the summarization of dataset is shown in the figure 2.2.

The tool, which closes in to our aim which is the sumarization of LOD datasets on the domain level, is the tool developed by Lalithsena with the usage of dataset Freebase and its available API. Their approach provides a technique to automatically identify the main topics of LOD datasets by utilizing Freebase as both background knowledge and to provide the vocabulary for the topic tags [29]. It sends labels of their classes to Freebase API and it returns Freebase domain type. The results are merged to create a category hierarchy where only hierarchies with the most common root are kept and the hierarchy with the most frequency is defined as the main domain of dataset. However, our aim is not automatic definition of one of the most frequent domain by the help of Freebase API, but determination of the domain proportion by the manual definition of the domains and classes, which creates particular domain. Then on the base of the existence of the given entities will be defined domain classification with the percentage coverage of the particular domains.

The other tool, which aims on the identification of the content and the domain classification of LOD datasets is implemented by Andrejs Abele. Their main goal is providing of the detail description of the dataset (classification

to the domains) and also the providing of metadata which publisher wants to publish and suggestions for possible datasets that the dataset could be linked to. The main domain categories are represented by DBpedia categories, because they cover big amount of domains. Categorization of the dataset to the particular domain is defined according to particular literals, on which is used statistical method TF-IDF and then on the base of the calculated ranking are determined top results.These top result are subsequently send to DBpediaSpotlight. The whole way of classification of the dataset to the domain is available on [30].

As the result of this research we found out that many tools for LOD datasets sumarization exist. There are tools providing mainly statistical information about datasets but also tools providing automatical domain identification of datasets mostly by using some statistical methods. We found no tool providing domain representation based on a count of occurrences of particular classes in datasets which creates the particular domain.

Chapter 3

Analysis

Analysis is considered as one of the initial phase of software development process. It is necessary to process it in detail and in a good-quality, so it can ensure elimination of mistakes, which could occur in the next stages of developing. Correction of the mistakes in the phase of design or implementation is much more expensive and difficult in comparison to this analytical phase. Furthermore, analysis is also the tool for primary familiarize customers with the developing application. Just on the basis of analysis, the customer can imagine the application like a unit and he is realizing, what is really necessary for him. He becomes aware, which originally requests he can pass or otherwise, which requests in the assignment were missing and they are necessary to fulfill to the analysis and to process them. It is often happening, that after early processed analysis, some mistakes are eliminated, which were caused by the inaccurate comprehension of the customer and the provider of software.

Software analysis usually contains some default analytical models. It includes for example functional and non-functional requirements about application, use case diagrams, data models and diagram of activities describing current running processes at customer in the comparison with the processes, which can be created by the usage of the requested software. As the developed application does not improve any established process at customer, diagram of the activities describing current processes will be missing in this paper.

3.1 Requirements

This section contains the list of all requirements, which are important within the developing of the application. Requirements are usually divided on functional and non-functional. Functional requirements describe requests about particular functions of application. On the other hand, non-functional requirements describe technical requests. Particular functional and

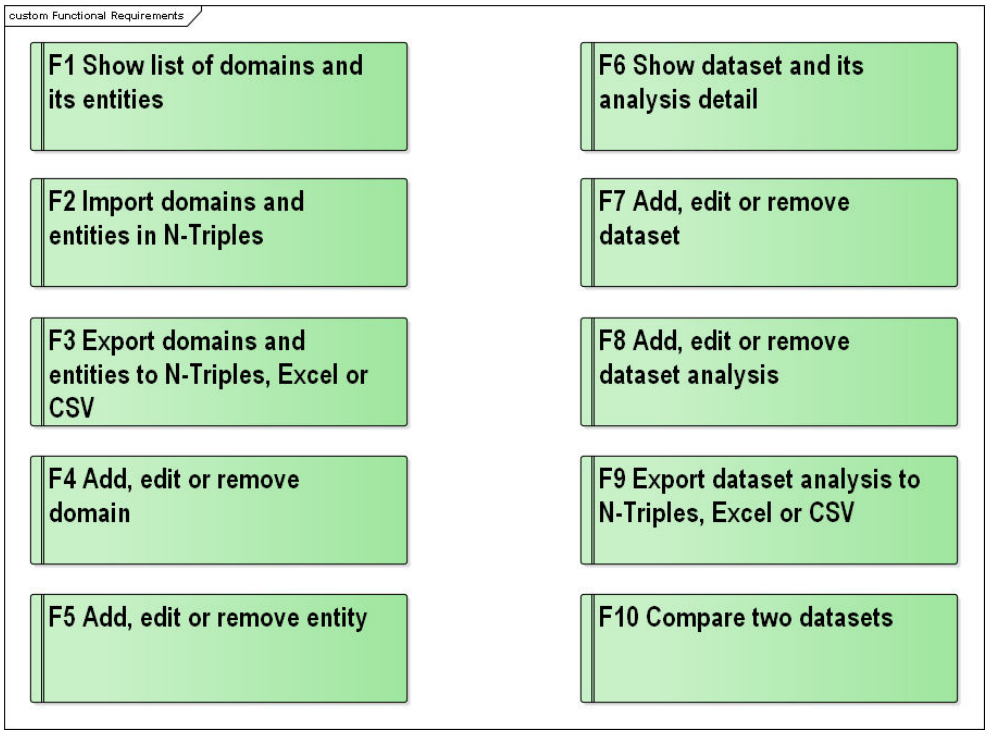


Figure 3.1: Functional requirements

non-functional requirements for this application are described in the sub-sections below, in the section 3.1.1 and 3.1.2.

3.1.1 Functional requirements

The figure 3.1 represents the summary of functional requirements. In the next sub-sections are presented particular requirements in more details.

F1 Show list of domains and its entities

User can view through list of particular domains. Domain represents some scope, for instance sport, art, technology, architecture etc., based on which the particular entities will be located to it. Entity represents some class in dataset ontology. Resource of this entity is usually RDF triple which contains specific predicate *rdf:typeOf* and object in this triple is this entity. User will be able to show/hide the list of entities in the database belongs to the particular domain by unpacking/packing this domain. Only one domain in the time can be unpacked. Below the domain list, there will be a pie chart, which will show representation of particular domains, which means how many entities every domain contains.

F2 Import domains and entities in N-Triples

User can import domains and entities from the local file in format N-Triples (prefix .nt or .NT). In the import file user can define new domain, create new entity under existed domain and define link between entities, which can be useful in case of processing different datasets.

F3 Export domains and entities to N-Triples, CSV or Excel

User will be able to look at domains and entities directly in the application (functional requirement F1), but also he will have the opportunity to export this overview into local file, to format N-Triples, CSV or to format compatible with the program MS Office Excel (version 2007 and newer). File in N-Triples will be in the same structure like file for import domains and entities. Others formats will represent overview in the structure: domain path, entity group (links between entities), entity path. In addition Excel file will contain the chart expressing representation of particular domains.

F4 Add, edit or remove domain

User will have the opportunity to add new domain by the usage of form. For the addition of new domain, it is enough to fill the name of it, and path will be prefilled automatically. User will also have the right to remove the existing domain (with all entities in it) and to edit its name or path.

F5 Add, edit or remove entity

User will have the chance to add the entity to the particular domain by the usage of the form, where it is enough to fill only the path of the entity. In this case, the new entity group within the domain will be created. Entity group is used for grouping same entities but differently described in different datasets (it represents the predicate *owl:SameAs*. If the user will choose the existing entity group, new entity group will not be created and the entity will be assigned to the selected entity group. Beside the addition of the new entity, the user will be able to remove or edit the existing entity. Within the editing, he will be able to change the path.

F6 Show dataset and its analysis detail

User will be able to display detail of calculated dataset analysis within the chosen dataset. After choosing dataset, in the first place, the user will notice meta-data about dataset, which contains name, description, path to the HDT file / SPARQL endpoint for the future calculation, type of dataset (SPARQL endpoint or HDT file) and ontology predicate. Ontology predicate represents the predicate which is used in the particular ontology for entity specification. After that, there will be list of dataset analysis. User can

show/hide detail of one dataset analysis in time, so the information will be more transparent for him. Detail of dataset analysis will contain meta-data about it, like name, description, number of domains used for the calculation, number of calculated triples and type of the calculation (short or long one with the calculation of predicates to the entity). The hierarchical pie chart will be following. First level will show proportion of domains and second level proportion of entities. In the end of dataset analysis detail, there will be shown the table with whole calculation. That means participation of domains, participation of entities and also the list of predicates (name and count) which are used with the particular entity.

F7 Add, edit or remove dataset

Also, there will be the possibility to import new dataset to the application, on which the user will be able to initiate particular analyses on base of defined domains and entities. The addition of the dataset will be possible through the form. The user will have to add the name, path to dataset, ontology predicate of dataset, optional description of dataset. Available formats for dataset import are SPARQL endpoint or HDT file. Import of dataset will contain also default calculation of analysis based on all domains. Therefore the user will have the option of short or long calculation in this form too. Inserting new dataset will be asynchronous operation (because of plenty time needed) and user will be informed about processing and finally finishing calculation. Beside the addition of the new dataset, the user can delete dataset (with deleting of dataset there will be deleted all dataset analysis too) or edit its name, description or ontology predicate.

F8 Add, edit or remove dataset analysis

User will be able to create his own dataset analysis according to his needs. This function can be used for instance, if the user will be need calculated information only through particular domains. Creation of the new analysis will be possible by usage of form, in which the user can choose domains, for which the analysis will be calculated. Also he will have to specify the title and type of calculation. The description of the analysis will be optional. Also he can erase the analysis or edit it, concretely the title and description.

F9 Export dataset analysis to N-Triples, Excel or CSV

User will be able to export the analysis to the format CSV, to the format compatible with the program MS Office Excel (version 2007 and newer), and also to the N-Triples format. Formats CSV and Excel will display data in the same structure, that is in the structure shown in the third part of the analysis (in the table), directly in the application. Moreover, Excel file will contains graph with domain partition. Export of dataset analysis in N-Triples format

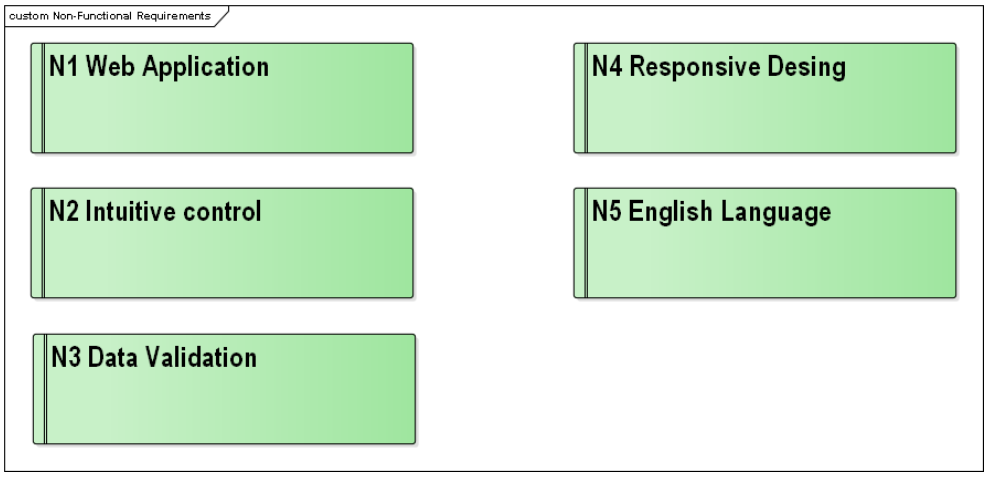


Figure 3.2: Non-Functional requirements

will contain triples with the number representation of domains and entities by the usage of predicate *void:triples*.

F10 Compare two datasets

User will have the possibility to compare two analyses of two different datasets in one application window. He will choose datasets and the analyses, which he wants to compare by usage of combo-boxes. After filled all mandatory fields, there will be displayed table with domains and entities, and their representation in particular datasets. Entities with the same entity group will be displayed in one row for better comparison.

3.1.2 Non-functional requirements

In the figure 3.2 there is shown the overview of the non-functional requirements. In the following subchapters, there are described particular requirements in more details.

N1 Web application

Created web application will be available through web interface with the support of the Internet browser Google Chrome, Mozilla Firefox, Microsoft Edge, Opera and Safari and their newest versions.

N2 Intuitive controlling

Application will have intuitive controlling and it will be user-friendly also for inexperienced users. Arrangement of the components will correspond

to the standards of web application, no misguided or confusing features included. Graphical interface will be transparent and it will contain no useless elements or advertisements, which could bother the user attention.

N3 Data validation

Form of data needed for instance for the import of dataset, addition of the domain, entity or analysis will be controlled during filled of particular fields, not after the confirmation of the form. In the case of wrong form, the application reminds the user to edit the form to be right. It will be proceeded with the notification, in what place the mistake has occurred at. After the right fulfilled of form, the application allows to the user to confirm it.

N4 Responsive design

Application will be available and transparent within all devices, so it will be accessible through Internet browser in the mobile phone, tablet or desktop computer. Design for all the types of the devices will be customized to the concrete device and its standard display size.

N5 Language

Entire web application will be in English language.

3.2 Use case diagrams

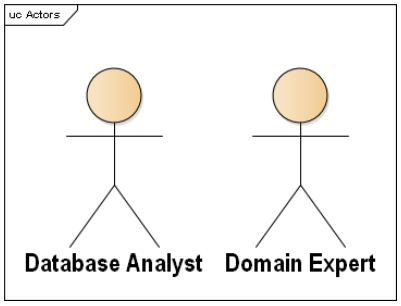


Figure 3.3: The user roles

Use Case Diagram describes functions, which the particular system has to do. That is, from the point of view of the user. It comes from the collection of the functional requirements (section 3.1.1)), which are described in more details here. This part of analysis also concerns with the description of the particular types of users, who will use developing system.

3.2.1 Users

This sub-chapter describes roles of the users, who will use the system. From our point of view, the users can be divided into two categories, look at the figure 3.3. Of course, one user can represent both roles.

3.2.1.1 Domain Expert

This user role is mostly interested in the administration of the overview of the domains and particular entities, which belong to given domains. Furthermore, it administrates links between particular entities, so it defines, which entities describe the same objects in the different datasets.

3.2.1.2 Dataset analyst

User role Dataset analyst represents ordinary user, who needs, for instance, to process the analysis of his own dataset (to find out percentage representation in the particular domains), to view the analysis of already existing dataset according to the default or his own setting. He may, also need to compare two datasets.

3.2.2 Show list of domain and entities

This use case diagram contains the ways of the displaying of the domains and entities. The user can view through all particular domains and entities in the database or he can also use searching tool on the page for the quick finding the domain or entity. The user can download and shown the whole list of domains, entity groups and entities in the format N-Triples, CSV or Excel. This diagram is shown in the figure 3.4 and it contains functional requirements F1 and F3.

The main scenario:

1. Scenario of this use case starts, when the user needs to know, which domains and entities can be used for the calculation of the analysis of datasets.

2. After the click on the page "Domain list", the list of domain will be shown. Each domain in the list contains default hidden list of entities, which are in the domain.

3. After finding of the searched domain, the user will unpack it and he will see the list of the entities, which are in the domain.

4. After searching of the particular entities, the user can hide this domain (but he does not have to) and he can continue in searching for the other domain (item number 3).

5. In the case, that the user needs to search the concrete domain or entity, he can use the searching tool. In the case of agreement, matched domains and matched entities and its domains will be shown to the user.

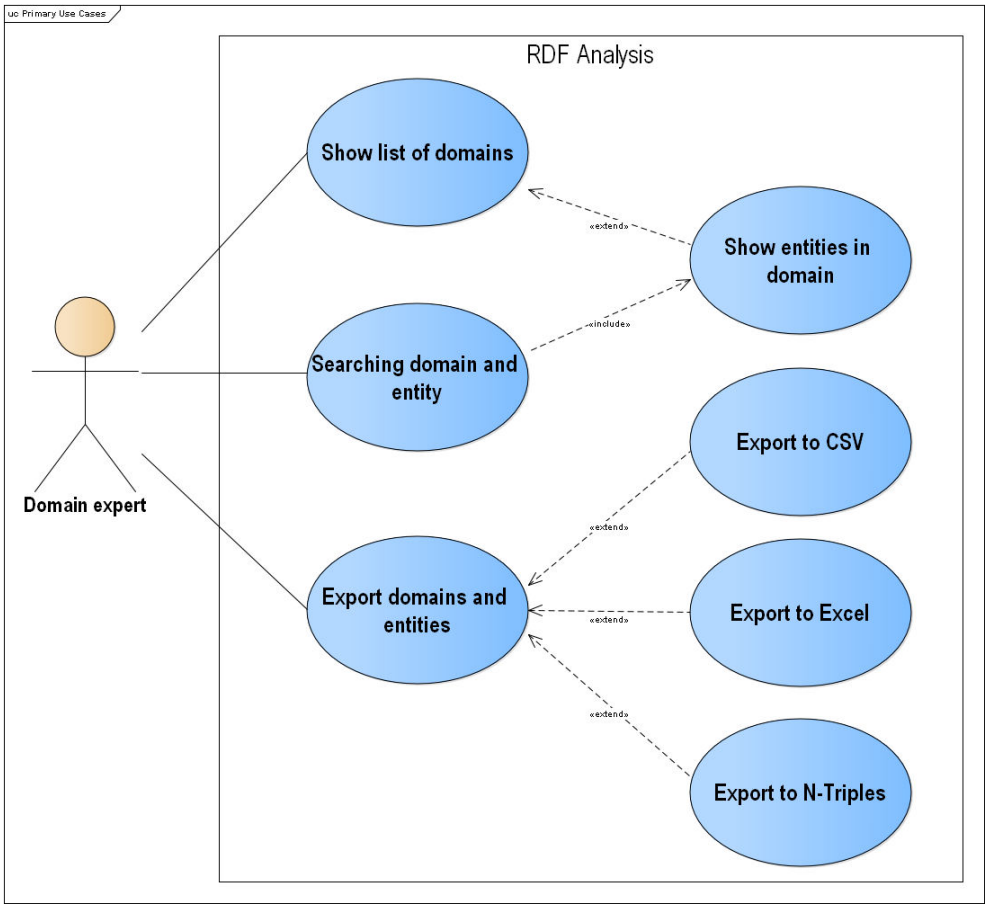


Figure 3.4: Use Case Diagram for showing list of domains and entities

6. The user also does not need to search domains one by one, but he can export this whole list to the CSV, Excel or N-Triples format.

7. This use case ends, if the user had found all the needed information about entities or domains.

3.2.3 Editing of the domains and entities

This use case diagram contains possibilities, by which the user can edit current list of domains and entities. Within the domain, the user can add the new domain, to edit the title an path of the existing domain, and also to remove the domain. However, the user has to pay attention, because when he removes the domain, he also removes the groups of entities and the particular entities in the chosen domain.

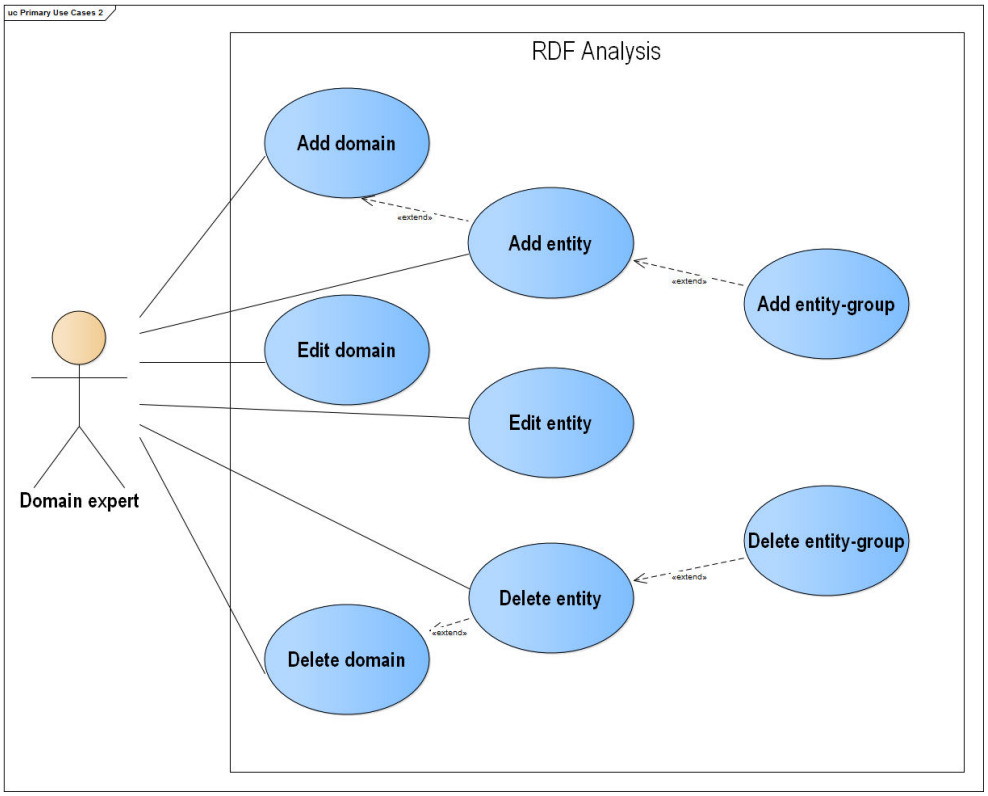


Figure 3.5: Use Case Diagram for editing of list of domains and entities

Within the entities, the user can also add the new entity. Within the addition of the entity, the user will pick either already existing group of entities, or he creates the new one (by fill nothing to entity group), which the particular entity will be inserted to. Also it is possible to edit the path of the entity. In the case of removal the entity, if it is the last entity in the group of entities, the whole group of entities will be also removed.

Furthermore, the user can import own list of domains and entities in format N-Triples in one time. From this file, firstly will be created domains, secondly entities and finally links between entities.

Use cases for the editing or removal of the entity or domain is simple, therefore there is no need to create for them the scenario. The main scenario for the addition of the entity to the system is described below. This use case diagram is shown in the figure 3.5 and it involves functional requirements F2, F4 and F5.

The main scenario:

1. The scenario of the use case starts, when the user needs to add one or more new entities to the system.

2. User can add domain or entity solo, through particular buttons on the page, or he can import list of domains or entities together.

3. In the case of importing domains and entities, the scenario ends after processing the import and getting response with failed domains or entities from the file. In the case of solo adding particular entries, the scenario continues in item 4.

4. User can add one entity in a time to already existing domain and he continues with the item number 6. However, the user can also create new domain first.

5. In the case, that the user needs to create new domain, he creates it by the usage of form for the inserting of the domain.

6. User continues with the creation of the new entity into chosen domain. He has two options: the creation of the entity when he does not choose the group of entity – together with the creating entity the new group of entities will be created, or he chooses from already existing groups of entities.

7. This use case ends with the successful insert of the new entity to the system.

3.2.4 Editing of datasets

This use case diagram describes the possibilities, by which the user can manage the current list of datasets in the system. There are possibilities like the addition of new dataset, editing of existed dataset, especially the title, the description or the ontology predicate or the removal of dataset. This diagram of the usage is shown in the figure 3.6 and it involves functional requirement F7. Functions as editing or removal of dataset are too simple, so we are not going to write the user scenario. Within the removal of dataset, also the particular analyses of datasets will be removed. The scenario for the addition of dataset to the system is described below.

The main scenario:

1. Scenario of this use case starts, when the user needs to add new dataset to the system and to generate the analysis for it.

2. User can open the form for the addition of the dataset, through the button "New dataset".

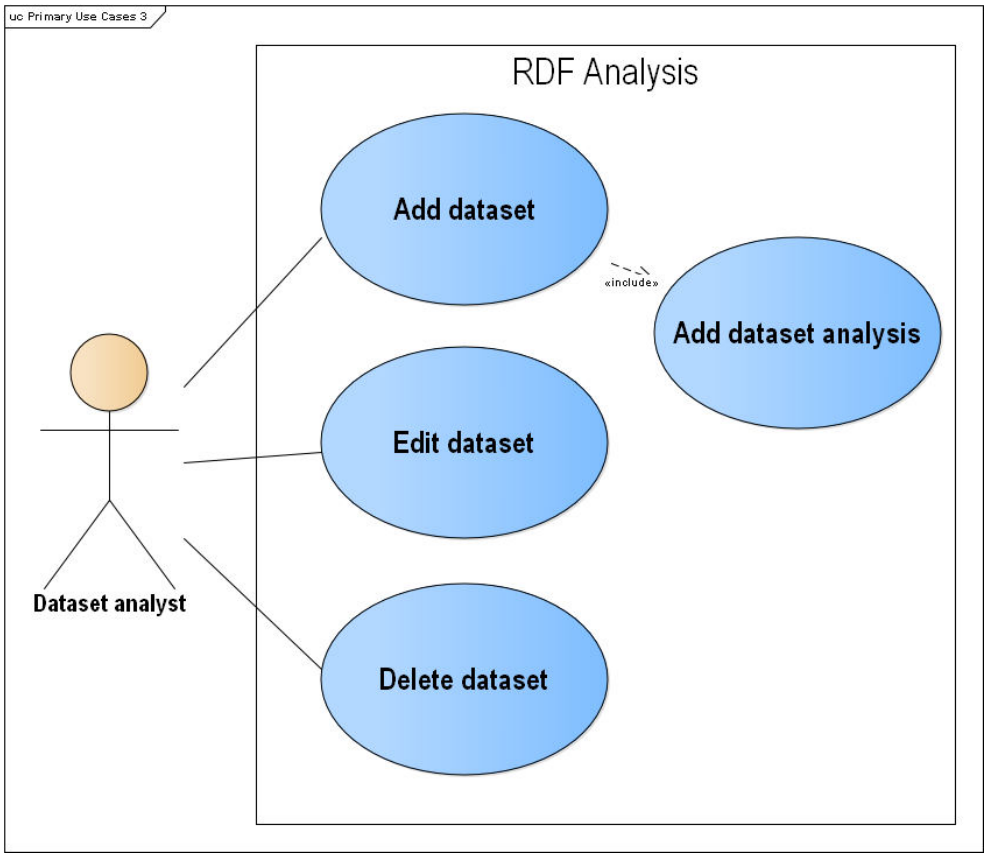


Figure 3.6: Use Case Diagram for editing of dataset

3. In the form, he has to fill-out the title of the dataset, ontology predicate, which is used in concrete dataset for specifying its ontology and path to dataset. This path can be SPARQL endpoint or file from local storage in format HDT. Optional parameter is text box for description of imported dataset. Also, he can choose the type of default calculation, short without calculation of predicates or the long one.

4. After successful submitting of the form, there will be running default dataset calculation for entire list of domain and the user will see notification about it. This will be asynchronous operation, so user won't be blocked.

5. This use case ends after finishing of calculation of new imported dataset.

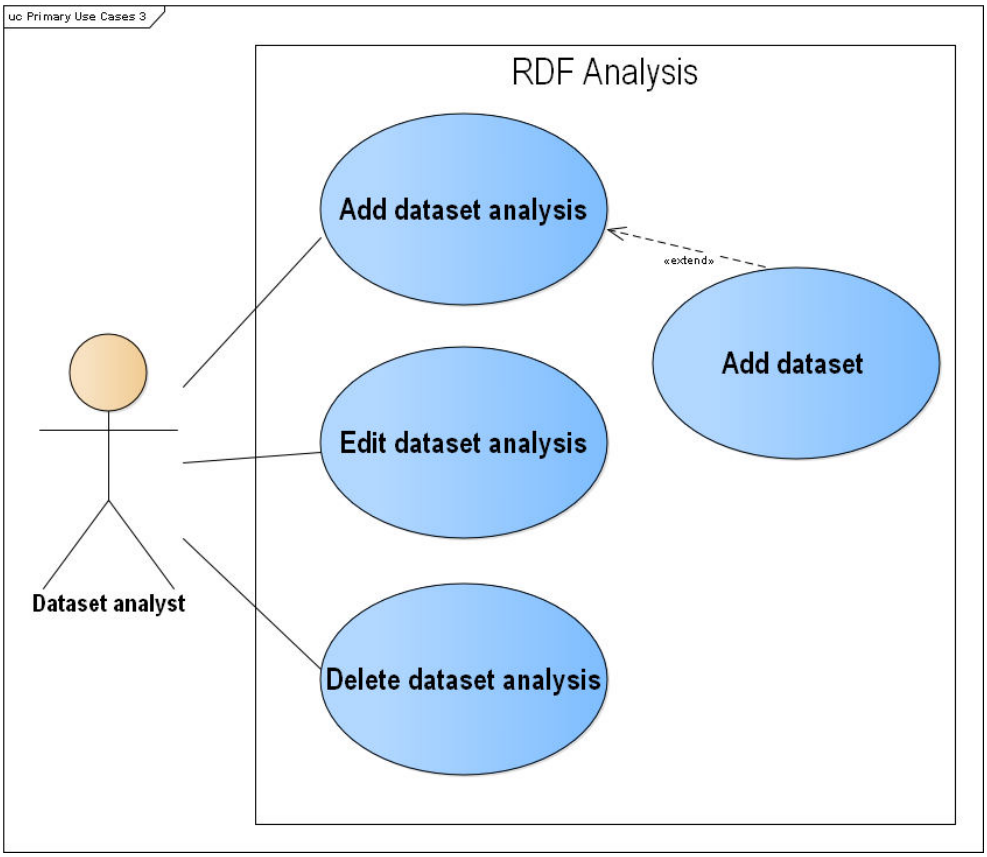


Figure 3.7: Use Case Diagram for dataset analysis editing

3.2.5 Editing list of dataset analysis

This use case diagram describes the possibilities, by which the user can edit the analyses of the particular datasets in the system in some way. They are, concretely, the addition of new analysis, editing of the analysis, especially the title and the description of the analyses or removal of the analysis. The addition of the new analysis can be done in two ways: either by insertion of the new dataset with the calculation of the default analysis or by creation of the new analysis on existing dataset. This use case diagram is shown in the figure 3.7 and it involves functional requirement F8. Functions as editing or removal of analysis are too simple, so we are not going to write the user scenario. The scenario for the addition of analysis of the dataset to the system is described below.

The main scenario:

1. The scenario of this use case starts, when the user needs to calculate new analysis for already existing dataset.

2. User can open the form for the addition of the analysis of the dataset through the button "Add analysis" after conrete dataset was chosen.

3. In the form, he is obligated to fill-out the title of the dataset analysis and specify list of domains, which will be used for the particular calculation. As an optional parameter is to fill-out description of new dataset analysis and the type of calculation, with or without predicates.

4. After successful submitting of the form, there will be running dataset calculation of filled specification and the user will see notification about it. This will be asynchronous operation, so user won't be blocked and he can for instance look at another dataset or analysis.

5. This use case ends after finishing of calculation of the user specified dataset analysis.

3.2.6 Show dataset and its analysis

This use case diagram describes the possibilities, which the user has for displaying dataset and its particular dataset analysis. After the choosing of the particular dataset, user can see list of available dataset analysis described by its name. After the choosing of the particular dataset analysis, user will see calculated result from the database. The result contains: metadata about dataset analysis, graph expressing percentage representation of domains and entitis in the chosen dataset, and detailed participation of the particular entities (number of triples) with the number of participation of the predicates. The result of the analysis can be exported to the format CSV, Excel or N-Triples. The user can also display the analysis of the dataset with the other analysis of the other dataset within the comparative mode. The main scenario for the displaying the analysis of the dataset is described below. This use case diagram is shown in the figure 3.8 and it involves functional requests F6, F9 and F10.

The main scenario:

1. The scenario of this use case starts, when the user needs to display calculated analysis of particular dataset.

2. The user will choose the concrete dataset from the dataset list, for which he want to display calculated analysis.

3. From the offer of the analyses he chooses the concrete analysis, which he wants to overlook. After the unpacking of the analysis, user can look

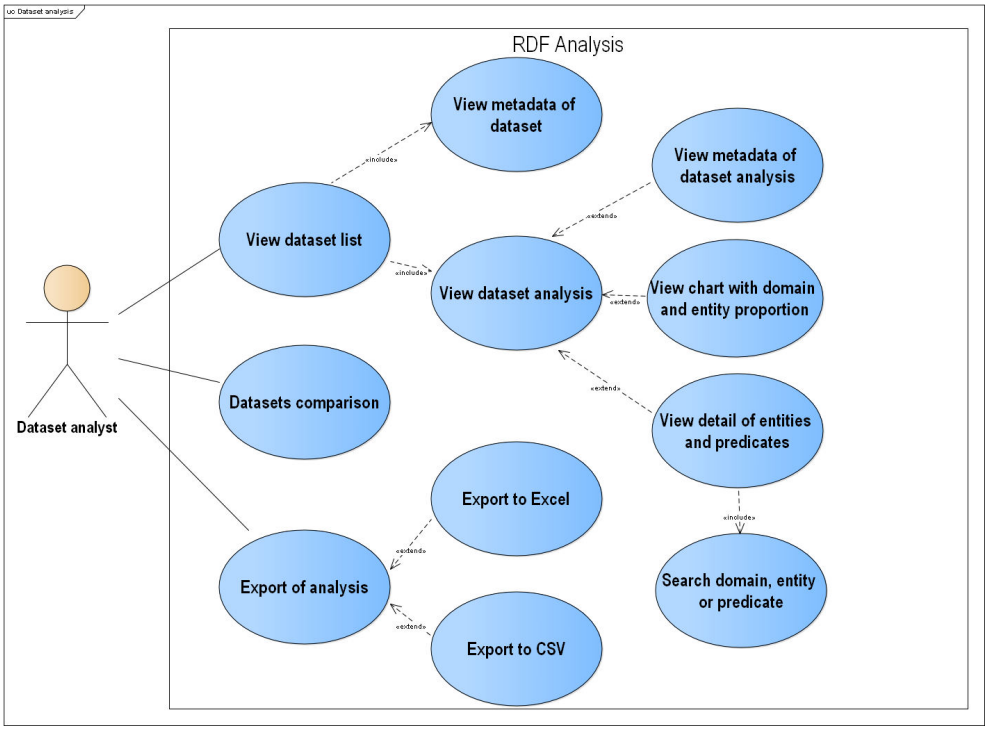


Figure 3.8: Use Case Diagram for view of dataset analysis

at the result. The result involves meta-data about dataset analysis, graph with the percentage division of defined domains and its entities, and the detailed participation of the entities and its predicates (only in case of the long calculation).

4. User can search particular entity or preditaces in entity detail with using search tool.

5. After the finishing of the overview of the result in the application, the user can export the result into the format CSV, to format compatible with MS Excel or to the RDF format N-Triples.

6. The scenario ends after the successful completion of the overview of the dataset analysis result by the user.

3.3 Domain model

The another part of the analysis of the information system or application is also the domain model, in the case, that the developing application is attached

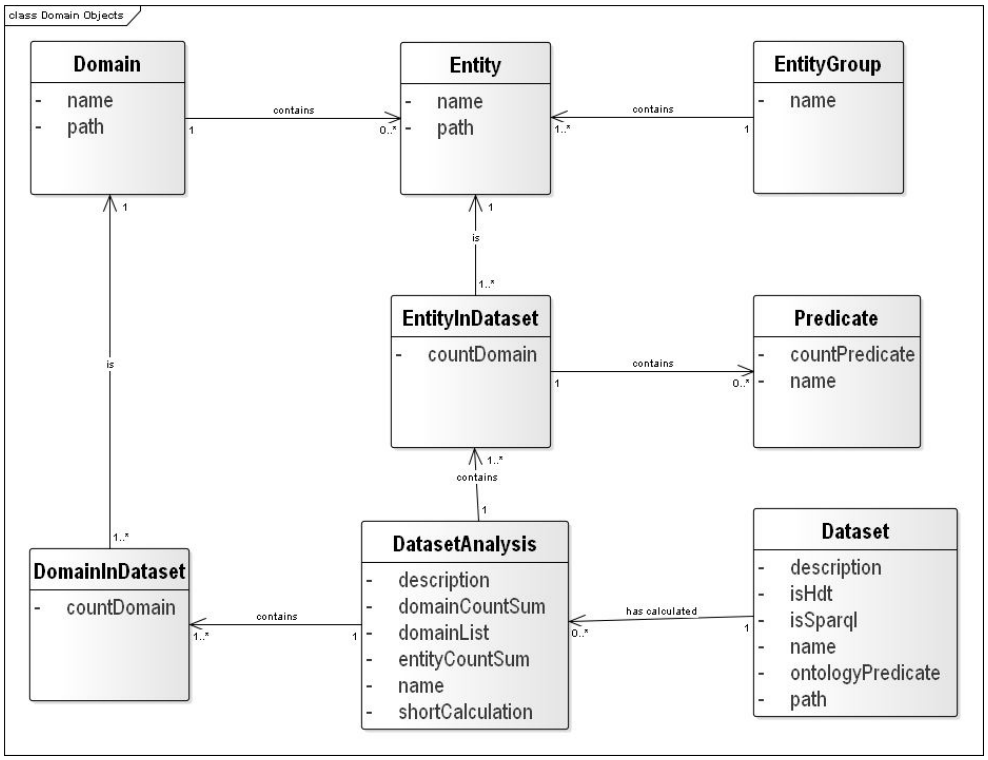


Figure 3.9: Domain model

to the database. This model describes particular classes of the system and it determines, which objects and information will be saved in the application database. It is necessary to realize, that domain model of the classes describes only classes related to the domain areas of the system, that it does not describe classes necessary for the run of the software, for instance system classes. Hence, domain model describes domain classes, relations between them, their multiplicity and also the attributes at particular classes, but only the domain ones, not system (for instance, attribute id or foreign keys in the domain model are not described). Domain model does not include system classes and attributes, not even the concrete implementation methods. Also it does not involve data types of attributes.

Domain model creates first displaying system as unit for the user imagination. Based on the domain model, the database model will be created in design part of software development. Domain model for this application is shown in the picture 3.9. Description of the particular classes is below in subchapters.

3.3.1 Domain

The class represents the domain, which contains entities from some part of life, and it is possible to use it for the calculation of the analysis of the dataset. Attributes of the class *Domain* are following:

- *name* - name of domain,
- *path* - RDF representation of domain, e.g. *http://mydomains.com/domains/SportAndRecration*.

3.3.2 Entity

This class represents the entity, which illustrates some class in particular dataset ontology. It is usually the object in RDF triple, where predicate in this triple is *rdf:type*. Entities are used for the calculation of the analysis of the dataset. Attributes of the class *Entity* are following:

- *name* - name of entity,
- *path* - path to entity in RDF format, e.g. *http://dbpedia.org/ontology/Athlete*.

3.3.3 EntityGroup

This class represents the group of entities, which are semantically the same, but they are described differently in the different datasets ontology. EntityGroup is like over-class of the entity. Attributes of the class EntityGroup are following:

- *name* - name of entityGroup, the title is created by the names of the particular entities in entityGroup divided by comma, e.g. *Athlete,Athletic*.

3.3.4 Dataset

The class represents the dataset, which the user inserts into the system. Then he can calculate the analysis above it. Attributes of the class *Dataset* are following:

- *name* - name of dataset,
- *description* - description of dataset,
- *path* - path to dataset, can be URL to SPARQL endpoint, or file in HDT format,
- *ontology predicate* - RDF predicate which determines specification of entity in dataset ontology,

- *isHDT* - checkbox, that determines if a given dataset is in format HDT,
- *isSPARQL* - checkbox, that determines if a given dataset is entered in the form SPARQL Endpoint.

3.3.5 DatasetAnalysis

This class represents calculated analysis for the particular dataset. Attributes of the class *DatasetAnalysis* are following:

- *name* - name of dataset analysis,
- *description* - description of dataset analysis,
- *domain list* - defined domains for the particular calculation ,
- *domainCountSum* - it represents count of domains, which were used for the analysis,
- *entityCountSum* - it defines the participation of all entities defined within the analysis for the concrete dataset (number of triples),
- *shortCalculation* - represent type of the calculation, short one without predicates or long one with predicates participation.

3.3.6 DomainInDataset

The class represents the information about domain, which was used for the calculation of the concrete analysis of the concrete dataset. It is helped domain and it contains calculated information about particular domain, which is linked directly to the dataset. Attributes of the class *DomainInDataset* are following:

- *domainCount* - it determines the participation of one domain within the dataset, it is calculated from the participation of the particular entities within the domain for the given dataset.

3.3.7 EntityInDataset

The class represents the information about entity, which was used for the calculation of the concrete analysis of the concrete dataset. It contains calculated information, which is connected directly to the dataset. Attributes of the class *EntityInDataset* are following:

- *entityCount* - it defines the participation of one entity within the dataset.

3.3.8 Predicate

The class represents the predicate, which is connected to the entity in concrete dataset. It involves all predicates of subject, which refer to the type of the given entity. Attributes of the class *Predicate* are following:

- *name* - name of predicate,
- *predicateCount* - the number of presence of the particular predicate in triples with the subject which is linked to the entity by ontology predicate.

Chapter 4

Design

Design of web application is next step within the developing of any software. It follows the previous phase in the life cycle of software, which is the analysis, mainly particular functional and non-functional requirements, which describe expected output from the application. Mostly, this phase consists of the class model, database model, architecture design and wireframes of the application. Besides of these particular design model, in this part there is also the possibility of the choice of implementation platform. Concrete details of implementation will be described in the chapter 5.

4.1 The choice of implementation platform

Based on the defined non-functional requirements in the analysis, designed application has to be available through the web interface. Also, it should have responsive design, and also, it has to be transparent and easy-to-use. These are the main points, based on which it is necessary to choose implementation platform for front-end. For the back-end, there are no specified non-functional requirements, so it is important to take a think, what is necessary to implement from the logical point of view, and which libraries needed for this application are available in which programming language.

4.1.1 Front-End

Because it is supposed to be a web application, as the main technology of front-end we use markup language HTML and cascade styles CSS. Because of the manipulation with the particular elements in the websites and the communication with the server, we decided to use script programme language Javascript and its very wide-spread library jQuery. As a base of the graphic style, we use expanded and very easy-to-use framework Bootstrap. We decided, that we do not use framework intended for the developing of front-end,

because it is more useful in the implementation of bigger and more complex projects and we have no skills with any.

4.1.1.1 Javascript and jQuery

Javascript is multiplatform, object-oriented script language, which runs on the side of the client and it react on the events in the browser. We use Javascript, because we used to work with it in the past, it is simple and it supports the technology of AJAX, which we are going to use for the communication of client and server. Javascript is currently the most used object-oriented language, which runs on the side of the client. In this time it is used in 94,2 % of all Internet websites [31].

As our future application is expected to be dynamic and interactive, we also decided to use Javascript library jQuery. jQuery is fast, small and easy-to-use library, which enables easy manipulation with HTML elements, animated effects, managing of events etc. Currently it is the most used JavaScript library, concretely in 71,4 % off all websites, from which 96,3 % create all websites, which use some Javascript library [32].

4.1.1.2 Bootstrap

Bootstrap is one of the most popular front-end framework determined for faster and simple developing of design of front-end of web applications. As according to non-functional requirements in analysis, graphic interface of our web application has to be transparent and responsive, we decided to use Bootstrap as the base of graphic design. Also, we have some experience with it from previous projects. Bootstrap provides many default styles for HTML elements, for instance tables, forms, buttons etc. As well, it offers various components (for example navigation panel, modal windows, progress bars, paging, tabs etc.) and also many own jQuery plugins (carousels, dropdowns, tooltips etc.). Bootstrap contains main elements for the arrangement of website and for responsive design. It has large documentation, thanks to which it is easy-to-use.

4.1.2 Back-end

As the basis of back-end part of our application, we made the decision to use object-oriented programming language Java. For the implementation of the REST application, it will be used framework Jersey. For the database, we will use the technology Sqlite and as ORM technology it will be ORMLite.

4.1.2.1 Java

We use Java as the main programming language of our application, because we have many experience with it from the previous school projects, and also, because there are many libraries written for Java which we need. One of the lib-

raries, which we necessarily need to use for the accomplish of the functional requirements of this application, is the library for the processing of hdt files, which is available only for the language Java and C++ [23]. So, Java has to be chosen, because we did not want to write the application in C++ or implement own library. Furthermore, for this application there is need to use framework Apache Jena, which supports developing of the semantic applications with use Linked Data. So it enables to create RDF graphs, querying about the data with the usage of SPARQL language and also parsing files to RDF formats. This framework is also written in Java language.

4.1.2.2 Jersey

We need the front-end part to communicate in some way with Java back-end, therefore, our application has to create web services in some way. Programming language will be Java and we decided for the restful architecture, so we have to use JAX-RS (Java API for RESTful Web Services). It is, actually API programmed in Java, which supports creation of web services on the base of REST architecture. However, we use Jersey framework, which expands JAX-RS API with additional features and utilities to further simplify RESTful service and client development [33].

Within the usage of rest web services we will need to return objects, so we decided to use Jackson library written also in Java, which supports transmission of objects to JSON formats and in a reverse order.

4.1.3 SQLite

As database technology, we chose SQLite. SQLite is an in-process library, that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects. [34]

Because our application belongs to the simpler ones, speaking about saving data to database, it will have simple database model with approximately 10 tables. It will be useless to apply big database solutions like, for instance, Oracle. SQLite is simple database and it is easy accesible from Java. More important is, that ER model is useless for the implementation. Besides, library ORMLite is also available in Java, which maps object to the relational database SQLite, therefore there is no need to create particular tables by hand.

4.1.4 Maven

For building the project, we will use the tool Apache Maven. Based
on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of informa-

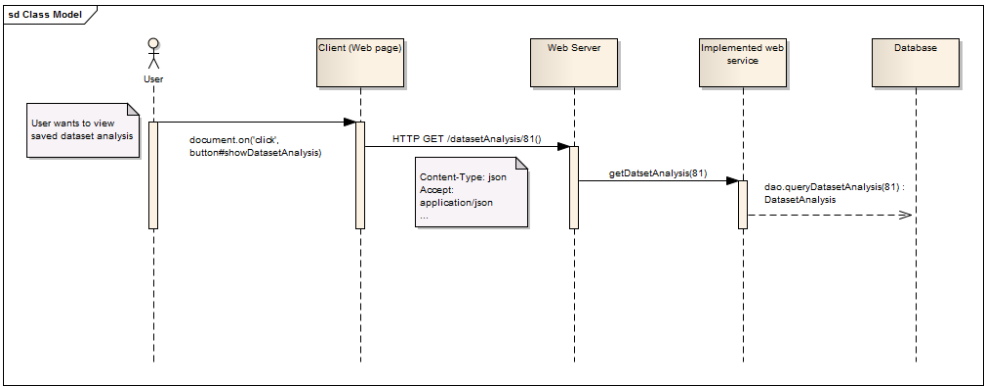


Figure 4.1: Sequence model of processing request for view dataset analysis result

tion [35]. We decided to use Maven, because it enables to download JAR libraries and the other dependencies from central repository. Then, the developer does not need to download and import all needed libraries for his project by himself.

4.2 The Architecture

We decided to use architectonic style REST (representational state transfer) for our web application. It works on the principle of architecture client – server. Client, by the help of protocol HTTP sends the request to the server to the uniquely defined source (e.g. /dataset/) with the HTTP method (GET, POST, PUT or DELETE), which has to be used on the particular resource. Then, the answer will return to it, either in the form of some serialized format of given resource (e.g. JSON, XML etc.), or in the form of type HTTP Response or in another forms. In the figure below, there is shown sequential diagram of one from the main user processes in the application, so the displaying of the analysis of dataset 4.1.

We picked REST architecture, because of simple use, it is easy expandable of other resources, because one object represents one resource and it supports standards as URL, HTTP, JSON, XML, AJAX etc. Also, we have some experience from school with it.

Architecture of our web application is divided into two layers. Presentation layer will be web browser, so web technologies as HTML, CSS, Javascript, JQuery etc. It will contain no classes. The second layer will present data layer, which will include logic of application and, also database access. Designing model of classes will be created only for this second layer.

4.3 Design class model

Class model is build on the base of classes designed in domain model in the analysis, but it should be more precise and detailed, so the programmer will be able to implement the application. Compared to the classes in domain model, furthermore, it contains data types of attributes, identification of private, public or protected attributes/methods, particular methods with parameter and and returns values. Besides of classes of domain model, this model contains also other helpful and configuration classes, which are not defined in domain model.

In our designing class model, we skip classic methods *get()* and *set()*, that the model will not be too disarranged. Basic classes can be divided into some packages, according to their functions. First package will contain classes, which relate to database objects. The second package will include classes for the logic functionality of application, so for the calculation of analysis of dataset. The third package consists of classes, which contain database connection and getting and inserting data to it. Application will probably include also other helpful classes, which will be needed during implementation itself.

4.3.1 Classes as database objects

This package will contain all classes, from which database entities will be created by the deinition of particular annotation by the usage of particular technology ORM. Each class involves attributes shown in the picture, methods get, methods set, empty constructor (needed for the particular ORM) and the constructor with all attributes. Detailed description of particular attributes is described within domain model in the section 3.3.

Among classes *Entity* - *EntityGroup*, *Entity* - *Domain*, *DatasetAnalysis* - *Dataset*, *Predicate* - *EntityInDataset* we used connection of composition, because this connection expresses dependency of one class to another. It means that one class cannot exist without the other one.

In this model, there are illustrated also id value and foreign keys within the particular tables against the domain model. Classes with attributes are shown in the class model in the figure 4.2.

4.3.2 Classes for the calculation of analysis of dataset

This package includes classes with methods, which ensures logic of the application, so the calculation of analysis of particular datasets, which SPARQL queries are part of it. Class model of this package is shown in the figure 4.3.

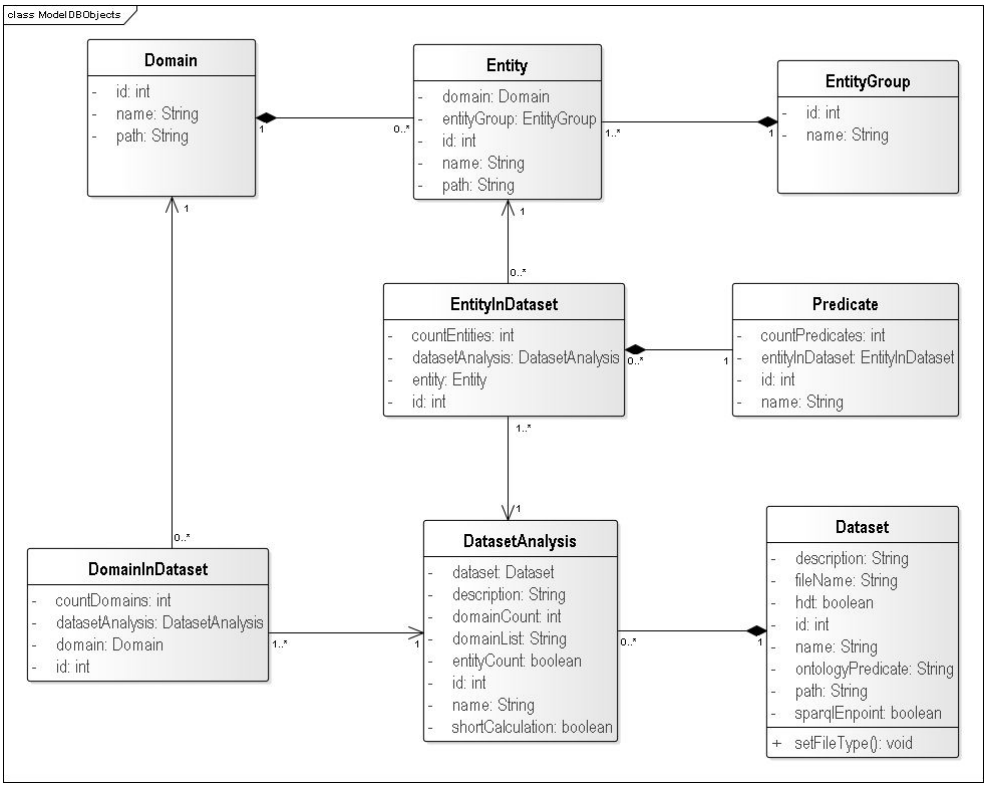


Figure 4.2: Class model for database objects

4.3.2.1 SparqlMethods

This class assures SPARQL communication, to which belongs creation of RDF models in the case of SPARQL endpoint, creation of HDT model in the case of import file in the format HDT, running of queries and the processing of the result. The class will be abstract and it will contain implemented classes *SPARQLEndpoint* and *SparqlHDT* that will have overwrite method *executeQuery* which will be build on the base of given format. Class will have no attributes and it will include methods described in the table 4.1.

4.3.2.2 Calculation

This class will be dealing with the calculation of the analysis of the particular dataset on the base of either SPARQL endpoint, or imported HDT file. Class will be abstract and it will contain implemented class *CalculationHDT* for the calculation of analysis of dataset in HDT format and the class *CalculationSparql* for the calculation of the analysis of the dataset defined by SPARQL endpoint. Class will include attributes described in the table 4.2

Table 4.1: Main methods in the class SparqlMethods

Method name	**Description**	**Parameters**	**Return value**
executeQuery()	Abstract method, which creates execution of selected query on the particular model. It is specific for sparql endpoint and for HDT file	String query, DatasetAnalysis datasetAnalysis, Model model	ResultSet - set with results of query
getResultAsOne Number()	Method, which transform result in ResultSet to number. It will be used in case of queries which returns count of triples.	ResultSet resultset	Integer
getResultAsMap StringInteger()	Method, which transform result in ResultSet to Map<String, Integer >. It will be used in case of queries, which returns string value and number value.	ResultSet resultset	Map<String, Integer>
getResultAsMap StringString()	Method, which transform result in ResultSet to Map <String, String>. It will be used in case of queries, which returns two string values.	ResultSet resultset	Map<String, String>

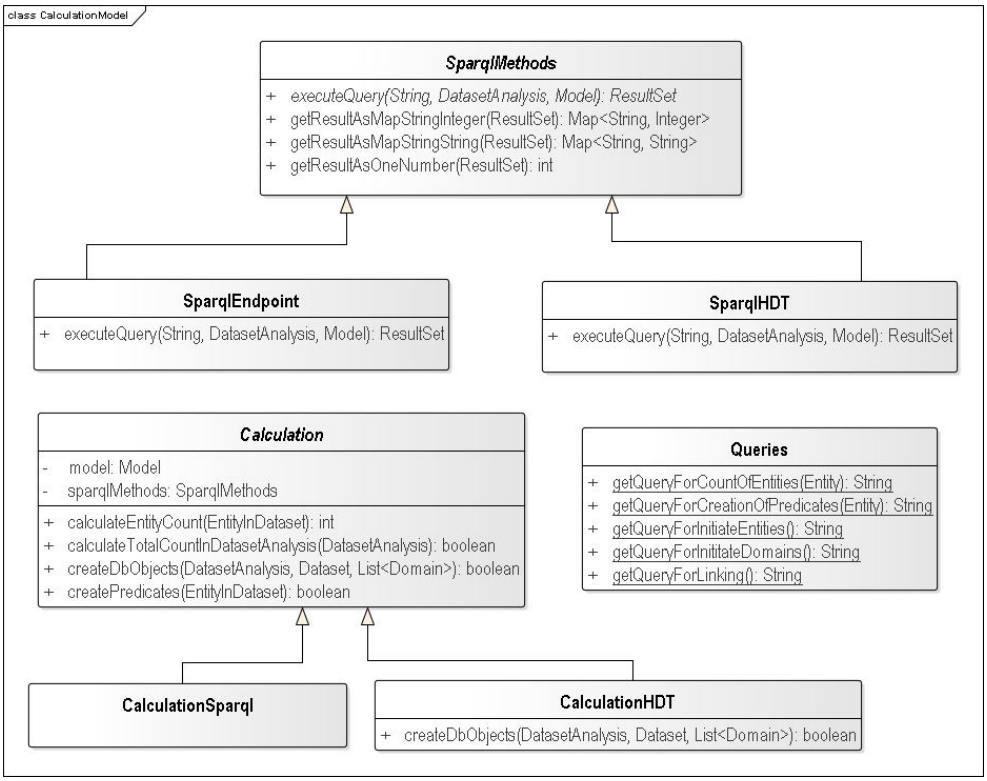


Figure 4.3: Class model for the calculation analysis of dataset

and the methods described in the table 4.3.

.

4.3.2.3 Queries

This class will be static and it will contain only the definition of particular queries. For our application, five basic SPARQL queries are needed, which will be evaluated. Of course, queries within the implementation can increases. Class will minimally include methods described in the table 4.4.

4.3.3 Classes for the manipulation of data in database

This package will contain classes, which will be receiving or editing of data in the database and it will create connection to the database, individual database table, etc. Particular classes of this package are shown in the figure 4.4.

Table 4.2: Attributes in the class Calculation

Attribute name	**Description**	**Datatype**
model	Attribute for persist HDT model. It will be defined only in case of class CalculationHDT	Model
sparq-Methods	Attribute for SparqlMethod. In case of calling class CalculationSparql, it will be set to class SparqlEndpoint(). In case of calling class CalculationHdt, it will be set to class SparqlHdt().	Sparql-Method

Table 4.3: Methods in the class Calculation

Method	**Description**	**Parameters**	**Return value**
calculate Entity-Count()	Method, which calculates count of entities in particular dataset and return that count.	EntityIn- Dataset entityInDataset	Integer
createPredicates()	Method, which creates predicates which are conneted to particular entity in dataset.	EntityIn- Dataset entityInDataset	Boolean
calculateTotal CountInDataset Analysis()	Method, which calculates total count of entitis and domains in dataset, and update particular dataset analysis	Dataset- Analysis datasetAnalysis	Boolean
createDbObjecst()	Method, which creates objects addicted to dataset analysis, like DatasetAnalysis, EntityInDataset, DomainInDataset,Predicate.	Dataset- Analysis datasetAnalysis, Dataset dataset, List<Domain >domainList	Boolean

Table 4.4: Methods in the class Queries

Method name	**Description**	**Parameters**	**Return value**
getQueryForCount-OfEntities()	Static method for getting query for get count of entities in particular dataset according to entity path.	Entity entity	String
getQueryForCreationOfPredicates()	Static method for getting query for get name and count of predicates in particular dataset according to entity path.	Entity entity	String
getQueryForInitiateDomains()	Static method for getting query for get domains from import file.	-	String
getQueryForInitiateEntities()	Static method for getting query for get entities from import file.	-	String
getQueryFor-Linking()	Static method for getting query for get links betwen entities from import file.	-	String

4.3.3.1 DbAccess

This class will help with the creation of database and with the whole communication with it, so it includes classes for receiving, editing, creation or deleting of data. Class will contain attributes described in the table 4.5. All attributes will set in the constructor of the particular class. Main methods of the class for the opening of the connection are described in the table 4.6. Class will be consist of four heritable classes, in which each class will concern with one database edit. It will be either delete, create, update, or get. Methods of the particular heritable classes are clear. They are methods for the access to the particular data in the database. They will be added according to the current needs on the base of implementation. Basic methods are shown in the figure 4.4.

4.3.3.2 DB

This package will, furthermore, include class *DB*, which will be static and it will initiate class *DbAccess*. It will be responsible for opening and closing of database lock too. Class is shown on the model of classes of the whole

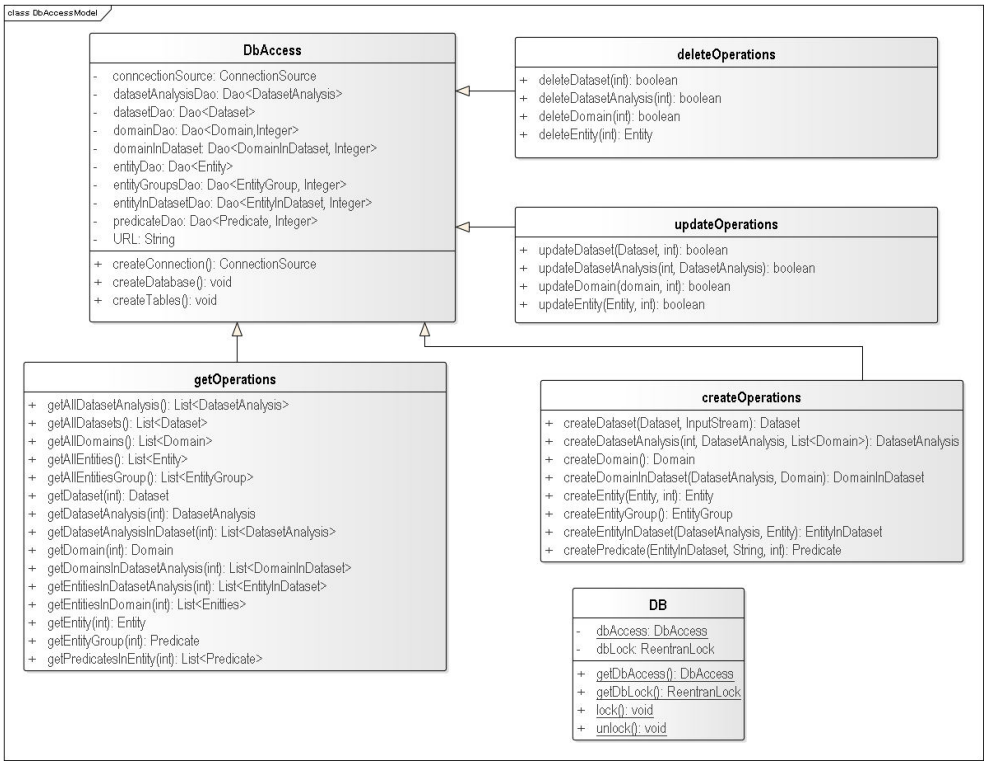


Figure 4.4: Class model for accessing to the database

package, look at the figure 4.4.

4.4 Database model

Because for the application will be used relation database, database model in this case will represent relation data model. We decided to use object-relation mapping for our application, so we are not going to create particular tables, but they will be created according to defined objects with the usage of the technology ORMLite. Database model contains classes, which are involved in the data model in analysis and also in the class model of database objects. Because of it, content of the particular classes and relations among them are not necessary to describe in more details. On the other hand, there is need to define primary keys, foreign keys and database properties of the particular columns. This features are described below. We did not draw a database model, because it is very similar to the domain model 3.9 or to the class model of database objects 4.2.

Table 4.5: Attributes in class DbAccess

Attribute name	**Description**	**DataType**
URL	Path to database	String
connectionSource	Created connection to database	ConnectionSource
domainDao	Object for handling database object Domain	Dao<Domain, Integer >
entityDao	Object for handling database object Entity	Dao<Entity, Integer >
entityGroupsDao	Object for handling database object EntityGroup	Dao<EntityGroup, Integer >
datasetDao	Object for handling database object Dataset	Dao<Dataset, Integer >
datasetAnalysisDao	Object for handling database object DatasetAnalysis	Dao<DatasetAnalysis, Integer >
entityInDatasetDao	Object for handling database object EntityInDataset	Dao<EntityInDataset, Integer >
domainInDatasetDao	Object for handling database object DomainInDataset	Dao<DomainInDataset, Integer >
predicateDao	Object for handling database object Predicate	Dao<Predicate, Integer >

Table 4.6: Methods in class DbAccess

Method name	**Descrtiption**	**Parameters**	**Return value**
createDatabase()	Method, which creates database if does not exist. It will be called with inititate DbAccess()	-	void
createTables()	Method, which creates tables if does not exists. It will be called with initate DbAccess()	-	void
createConnection()	Method, which creates connection to database. It will be called with initate DbAccess()	-	void

4.4.1 Domain

Table *domain* will keep particular saved domains in database. The user can define the domain independently one by one or by the help of imported file. Path and the name of the domain have to be unique to not to confuse the user and also, the database should not contain duplicated information. Individual columns and their properties are following:

- id - primary key, integer, automatically generated, not null, unique,
- name - varchar, not null,
- path - varchar, not null, unique.

4.4.2 EntityGroup

Table *entityGroup* will keep particular saved groups of entities. It is not possible to create it by user and the user will not see it in the system. It is creating by the creation of the entity. Particular columns and their properties are following:

- id - primary key, integer, automatically generated, not null, unique,
- name - varchar, not null, unique.

4.4.3 Entity

Table *entity* will keep particular saved entities. Each entity has to belong to one domain and to one entity group. Entity can be defined by the user

individually one by one or by the help of imported file. Path of the entity should be unique, so the database does not contain the same information several times. Individual columns and their database properties are following:

- id - primary key, integer, automatically generated, not null, unique,
- path - varchar, not null, unique,
- entityGroup_id - foreign key, integer, not null,
- domain_id - foreign key, integer, not null.

4.4.4 Dataset

Table *dataset* will keep particular saved datasets. It is the main table of the application. Dataset can be defined by the user and its name has to be unique. Also, there should be unique path to the SPARQL endpoint or HDT file. Not null has to be name, filename and ontology predicate on which base it will be calculated analysis. Particular columns and their database properties are following:

- id - primary key, integer, automatically generated, not null, unique,
- name - varchar, not null, unique,
- description - varchar,
- fileName - varchar, not null, unique,
- ontologyPredicate - varchar, not null,
- isSparqlEndpoint - boolean,
- isHDT - boolean.

4.4.5 DatasetAnalysis

Table *datasetAnalysis* will keep particular saved analyses of dataset. Analysis of dataset can be defined by the user, but default analysis is created together with the creation of dataset. Each analysis of dataset belongs to one dataset and it cannot exist without it. Particular columns and their database properties are following:

- id - primary key, integer, automatically generated, not null, unique,
- name - varchar, not null, unique,
- description - varchar,

- domainList - varchar,
- domainCount - integer, default value is 0,
- entityCount - integer, default value is 0,
- dataset_id - foreign key, integer,
- shortCalculation - boolean.

4.4.6 DomainInDataset

Table *domainInDataset* is helpful and it keeps calculations of domains within the particular analysis. Calculations have to be saved for the reason of slow calculation of the whole analysis, to be enough to calculate particular analysis once. Each *domainInDataset* belongs to the one *datasetAnalysis* and to one *domain*. Particular columns and their database properties are following:

- id - primary key, integer, automatically generated, not null, unique,
- domain_id - foreign key, integer,
- datasetAnalysis_id - foreign key, integer,
- countDomain - integer, default value is 0.

4.4.7 EntityInDataset

Table *entityInDataset* is helpful and it keeps the calculation of entities within the domain in the analysis. Calculations have to be saved for the reason of slow counting of the whole analysis, to be enough to calculate particular analysis once. Each *entityInDataset* belongs to the one *entity* and to one *datasetAnalysis*. Particular columns and their database properties are following:

- id - primary key, integer, automatically generated, not null, unique,
- entity_id - foreign key, integer,
- datasetAnalysis_id - foreign key, integer,
- countEntities - integer, default value is 0.

4.4.8 Predicate

Table *predicate* is helpful and it keeps calculations of predicates within the entity in the particular analysis. Calculations have to be saved in the reason of slow calculation of the whole analysis, to be enough to calculate particular analysis once. Each *predicate* belongs to one *entityInDataset*. Particular columns and their database properties are following:

- id - primary key, integer, automatically generated, not null, unique,
- name - varchar,
- entityIndataset_id - foreign key, integer,
- count - integer, default value is 0.

4.5 Wireframes of the application

With the creation of web application it is necessary to design the composition of the particular elements in the right way, because the application should has a clear design. Also it should provide asked functionality. On the base of design is then possible to identify, which elements are really important for the user, which are, on the other hand, not important and it is possible to remove them or to move them to the background. Design of the user interface is possible to divide on lo-fi, that is wireframe, which displays, in fact, only the composition of the particular element, without color, without interaction and without any other functions. On the base of wireframe, there is next created wi-fi design, which displays also the graphics. It is concrete front-end, for instance, with the validation of data, but without computing logic. In this part of paper, we concentrate only on the lo-fi design, because wi-fi design will be already the part of the implementation.

The fastest way to create wireframe is usage of pencil and paper. Furthermore, there exist many tools, also interactive, which we can possible use for this design. We decided to use the tool called Balsamiq [1]. Every section of our web application will be divided into three main parts, the header, body of the page and the footer.

4.5.1 Header

Header of the page will contain the logo with the name of the application on the left side, which will be simultaneously the reference to the landing page. On the right side, there will be located navigation of the page (horizontal), which will include three items: Dataset Analysis, Domain List and About.

4.5.2 Footer

Footer will content information about author's laws and it will be places in the bottom of the page.

[1] Available from the: https://balsamiq.com/download/

4.5.3 Body

Body will be different in each section, therefore we will describe every page separately. However, all pages will have the same type of displaying the forms for the editing of the data. They will be displayed by the help of pop-up modal windows, which will have buttons *Close* and *Save*.

4.5.3.1 Domain List

The first element of the page of Domain List will be a title element, which will create the heading of the page on the left side and control buttons with the import, adding of domain and export domains on the right side. Heading will, actually, create every page besides the Landing page.

The main element for Domain List will be drop-down panel list, which will be in the form of accordion, so after user clicks on one element, the previous displayed one will pack up. We chose form of accordion, in order to display only current searching information. Also, the user should not be too confused by the big amount of data. Particular panels will display particular domains, which, on the right side will contain control buttons for the editing, deleting or addition of entity to the given domain. Entities will be shown in the form of table and on the right side, they will include also control buttons for the editing of concrete entities, eventually deleting.

In the end of the page, there will be shown a pie chart, which will represent percentage representation of particular domain, so how many entities each domain contains. One of the wireframe of this page is shown in the figure 4.5. The other wireframes of this page are included in the addition C.

4.5.3.2 Dataset Analysis

This page is the main page of our application. Page will be divided into two tabs - dataset analysis and comparison. Both tabs will be on the page located in the top left corner.

First tab, that is dataset analysis, will contain in the left part vertical menu, which will include the list of saved datasets in the particular application. As the first element of this vertical menu will be the button with the addition of the new dataset. The rest content of the page will create dataset detail. In the above, there will be the heading of chosen dataset or Overview (in the case of chosen no dataset) with the control buttons on the right side. Every displaying of the dataset will be composed of the table with the main information about dataset and also, it will contain the list of analysis. Analyses of the dataset can be more and probably will contain lot of information, so analyses will be shown by the form of panel of accordion type, so after the displaying of one analysis, the other one will pack up, previously unpacked analysis. Analysis of dataset will consist of three more elements. First one will show main information about analysis in the form of table, the second part

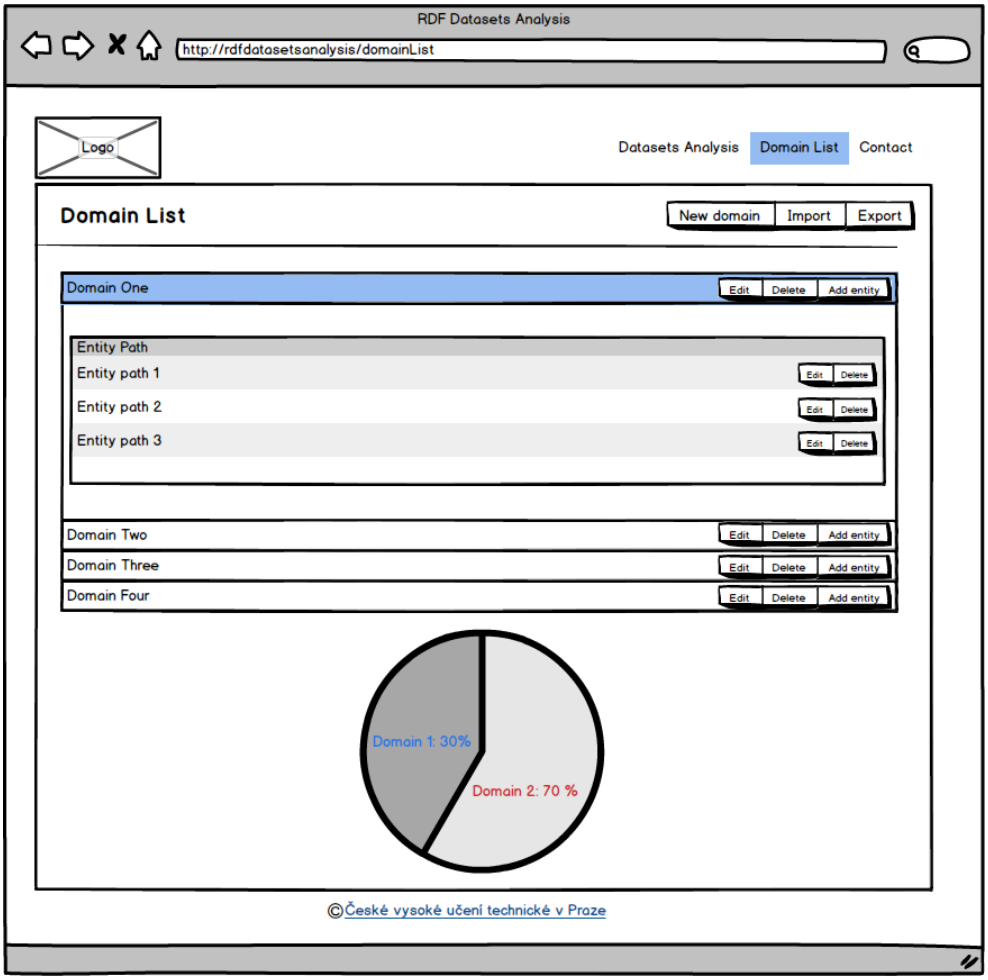


Figure 4.5: Wireframe - Example of the page Domain List

will contain the chart with the domain and entities representation in the particular dataset and the third part will include detailed table with occurrence of particular entities divided between domains. All control buttons of the page will be located on the right side.

The second tab will present comparative mode. In the beginning of the page, there will be, of course, the title of the page. After that, there will be three columns in the form of table. In the head of table, the first column will be empty. The second one will contain combobox for the choice of dataset number 1 and combobox for the choice of the analysis of dataset number 1. The third one will include also comboxes for the choice of the second dataset and the second analysis. Individual lines of the table will create domains and entities, and their representation in the first and second dataset. After

the clicking on the domain, belonging entities will unpack.

One of the wireframe of this page is shown in the figure 4.6. The other wireframes of this page are included in the addition C.

4.5.3.3 Landing page

Landing page, against to other pages, will contain bigger logo and bigger navigation menu without navigation to the contact and helpful information. It is because this information will be included directly on the Landing page, together with the welcome text, with picture and the instruction for use. From Landing Page, the user can get directly to the analyses of datasets or to domain list. Wireframe is not necessary, because Landing Page will contain the same navigation menu and only text elements.

4.5.3.4 About

This page will include contact information, on which the user can apply to, in the case of problems or in the case of idea on improvement. This page will aslo contains source of the application and some helpful information about using the application. For this page, the wireframe is not necessary, because it contains the same navigation menu as the other pages, and it includes only static text information.

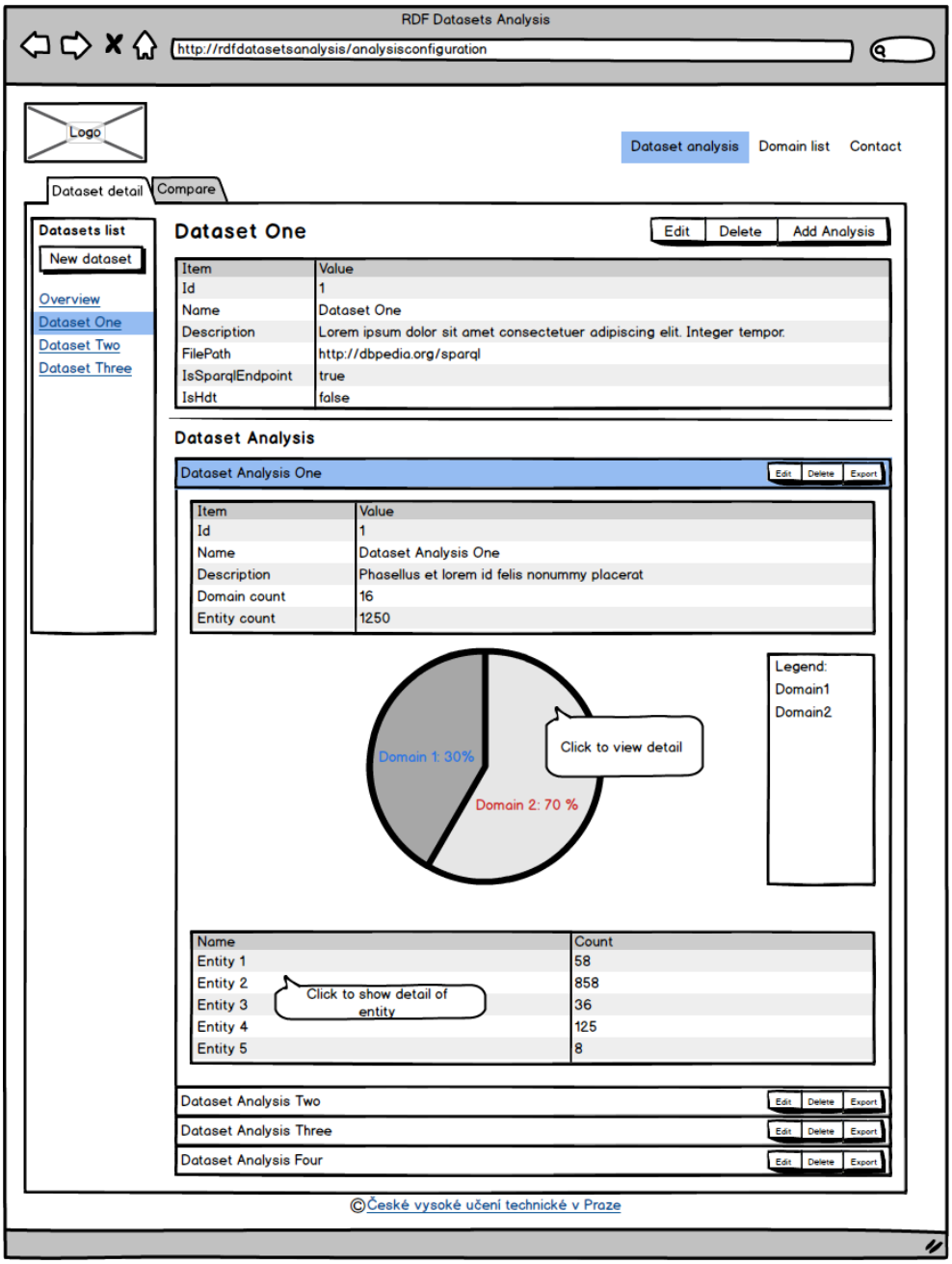


Figure 4.6: Wireframe - Detail of dataset analysis

Chapter 5

Implementation

After the introduction with the issue, processing the analytical requirements and inventing the design, it is necessary to move to the almost last part in the life time of application and that is implementation. During the implementation, we were trying to accomplish in order that the result application fulfills all functional and non-functional requirements. Also, it was very important that the application follows designed model of classes, database model and designed architecture.

In this chapter, we will pay attention mainly on the concrete particular functions of the application. We mention only the most interesting ones. Screenshots of the application are shown in the addition D.

5.1 Creation of user interface

We decided already in the design part, that for the creation of the user interface we will use mostly front-end framework Bootstrap. As according to the requirements in the analysis that application should be responsive and easy-to-use, Bootstrap really helped us with these conditions. It is because of its design, which is simple and clear and it contains many useful consistent components. We used mainly its responsive grid system, which scales 12 columns according to the current width of the window. For the responsiveness, besides of this grid system, we also used jQuery function *resize()* for the adjusting of the design depending on the changing width of the window, for instance, editing of the scale of the graph, font, buttons, etc. or function *width()* for the receiving of the current width of the window.

For the simple and intuitive controlling of elements, we used Javascript library jQuery, which we use mainly on the hiding/showing of the particular HTML elements, handling of the HTML elements, setting of attributes and processing of the event.

In the case of the implementation of the responsiveness, we did not have wireframes designed on smaller screens because the application is mainly

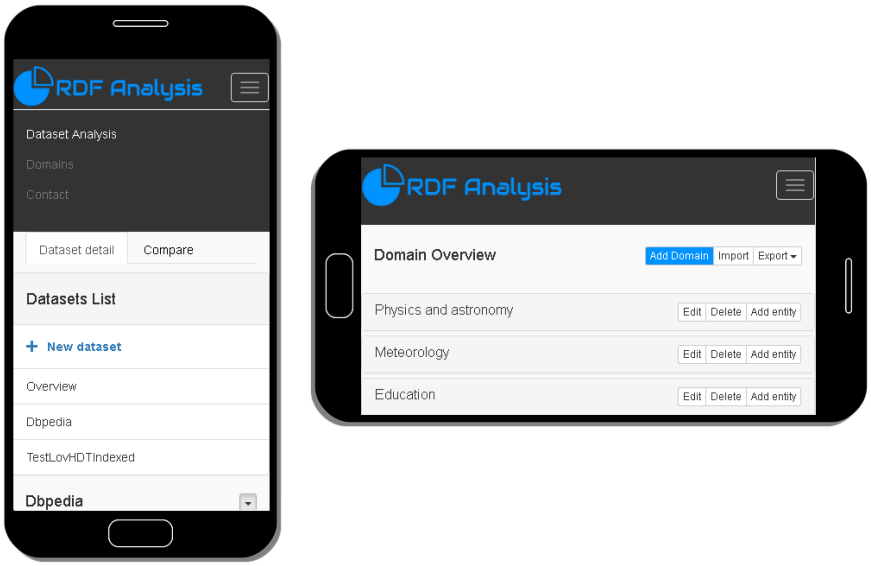


Figure 5.1: Example of the responsiveness of the application

determined for the Internet browser, primarily in the computer. We kept the main standards, for instance packing the title menu to the button (which is spread among mobile application) or removal of the big amount of the buttons. Instead of them we created one buttons with the drop-down list, etc. Example of responsiveness is shown in the figure 5.1.

5.2 Data validation

Within the implementation, we decided to validate entered data from the user mainly in the front-end, before the sending of data to the server. It means, that the button which is responsible for the sending and processing of data, firstly check the correctness of all data and after the fulfillment of the condition, data will be sent to the server for processing. In the case of failure, the user is informed, what is necessary to fulfill first. Validation of inserted data is running not only on the button, but already during the fulfilling of validated field. In the moment of correctly entered data, the background color of the filed will make green, in the reverse case, it will be red color. The user is informed with the obligatory input fields by the displaying of the used symbol "*". Every field contains help note which will be displayed by click on the small button with the symbol "?". Form validation is shown in the figure 5.2.

For the validation on the client side, we used the library Parsley, version 2.6.0 [2]. Parsley is the open source Javascript library for form validation. It is easy to build it to already existing forms, and for the graphic interface, so e.g. for the emphasizing correctly and incorrectly entered inputs and for the displaying of the alert messages it is used also framework Bootstrap. Library contains big amount of implemented validations, which the user is able to parameterize, e.g. obligation of the input, maximum/minimal length, email validation, URL validation etc. Besides the default validation, we had to use own validators, so, for instance, to the uniqueness of the domain path, dataset name etc. Implementation of the own validator included the definition of the condition for the evaluation of the validation and the error message. Library supports yet English and French language, what was enough for our application.

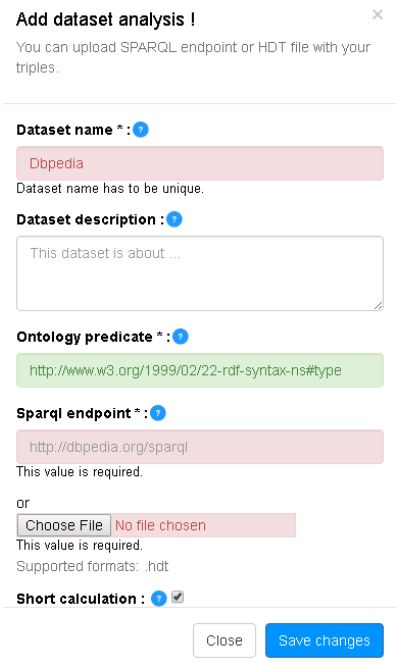


Figure 5.2: Example of the form validation in the application

We had to make decision, what library to choose, Parsley, jQuery-FormValidator [3], or jQueryValidationPlugin [4]. We had problem with the jQueryFormValidator with running validation of form on button out of the form and also with interactive validation and requirements fields. About the library jQueryValidationPlugin, we did not like mainly design, and it has weaker documentation with the examples in the comparison to the rest of the other libraries.

5.3 User information about processing of the calculation in application

In the moment of the correct/wrong processing of data or in the case of the processing of asynchronous long request, is the user informed by alert in the form of the colorful element dev, which is places in the beginning of the page. Because we send the data to the server via AJAX, and after the successful in-

[2] http://parsleyjs.org/#
[3] http://www.formvalidator.net/
[4] https://jqueryvalidation.org/

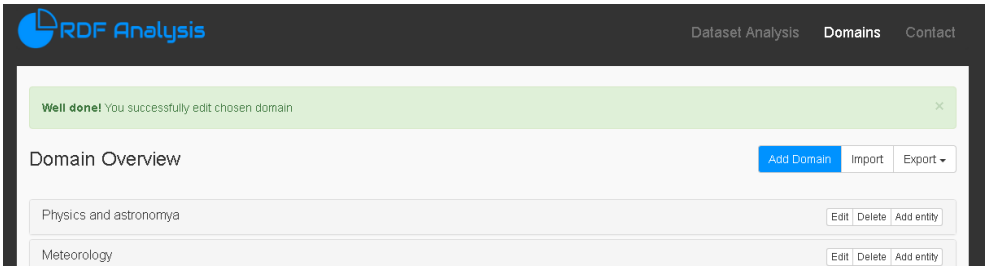


Figure 5.3: Information about successful operation of editing dataset

sertion, editing or deleting of data is necessary to refresh the page, information about processing of data we saving to *local.storage*, which is called immediately after running the script. After using this information (displaying to user), we remove it. Example of successfully processing form is shown in the figure 5.3.

At big operations, as, for instance, processing the analysis or importing of dataset, the user is notified also in modal after the confirmation of the form.

5.4 Chart drawing

For chart drawing we decided to use library D3.js [5]. It is the Javascript library oriented to data visualization by usage of language HTML, SVG and CSS. This library is available on Github and it containts large amount of examples, simple ones like column chart, donut chart, pie chart, etc., but also difficult professional charts of the real data visualization in existed projects. It is enough to choose any chart and adjust it to concrete data and needed visualization.

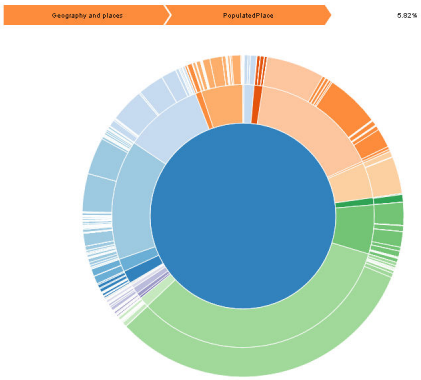


Figure 5.4: Chart of dataset analysis

There is a large amount of Javascript or jQuery libraries for drawing nice and interactive charts, for instance Google Chart [6] or Charts.js [7]. Furthermore,in both of these libraries is build responsiveness which could be very useful. The main problem was, that these libraries contains only simple charts, not suitable for our application, when ve wanted to use hierarchical pie chart. So, be-

[5] https://d3js.org/
[6] https://developers.google.com/chart/
[7] http://www.chartjs.org/docs/

cause of this issues, we decided to use library d3.js, which that type of chart offers.

For visualization of dataset analysis, we decided to use hierarchical zoomed pie chart. It means that chart has two levels. The first level displaying proportion of domains in the particular dataset. If user click on some domain, there will be displayed the second level which represents proportion of entities in the chosen domain. In the beginning of chart is also displayed actual path, which shows the domain and the entity, on which user stay. Example of this hierarchical pie chart is displayed in the picture 5.4.

For visualization of domains and entities in page *Domain List*, we decided to use simple pie chart, which displays proportion of domains in database on the base of entity count in each domain. For this chart we could use another simpler library, but we decided to honor the same design.

These charts are not responsive, so we has to make own responsibility by usage of Javascript function *windows.width()*.

5.5 Export

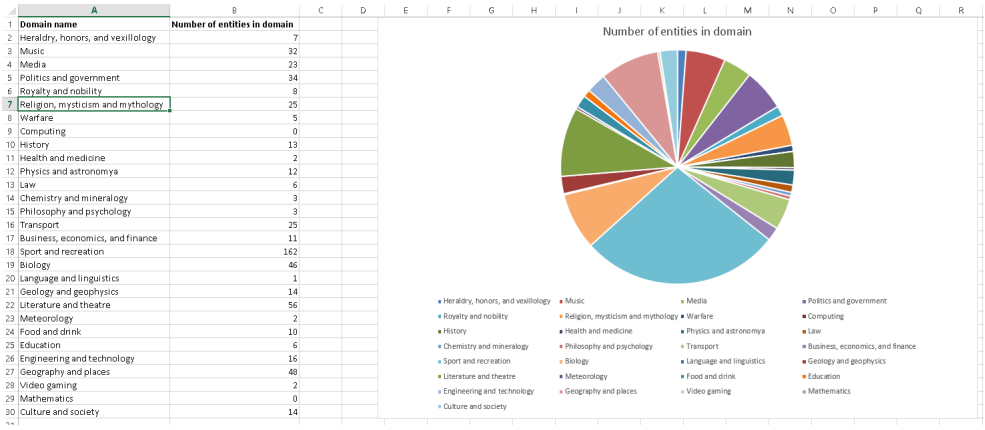
We made the decision to add new unplanned functionality into functional requirements because of large volume of data our application works with. This new functionality is export of domains and mainly dataset analysis to the external files. It makes data more readable and it is possible to use that for further processing.

We decided to implement functionality that user will be able to export dataset analysis and overview of domains and entities to Excel, CSV or also to RDF format, specifically N-Triples.

Every particular exported file is saved on the server with unique name with the help of java function *File.createTempFile()* which adds random number sequence to the end of filename. Files are deleted from the server after a certain time. We created new special objects containing export attributes in order to make data output easier. These objects are used for nothing but data export.

Export of domains contains list of domains, entity groups and entities. Furthermore, in the excel file there is pie chart based on exported data. In N-Triples format file are domains displayed in the same way as in the case of domain import.

In case of export of the dataset analysis detail, there is detailed list of domains, entities, predicates and their representation in the particular dataset in CSV and Excel format. In Excel is also a pie chart with the domain representation on the first sheet. Exported file in format N-Triples do not contain predicates, it contains triples with domain and entity representation with the usage of the predicate *void:triples*.



	Domain name	Number of entities in domain
2	Heraldry, honors, and vexillology	7
3	Music	32
4	Media	23
5	Politics and government	34
6	Royalty and nobility	8
7	Religion, mysticism and mythology	25
8	Warfare	5
9	Computing	0
10	History	13
11	Health and medicine	2
12	Physics and astronomya	12
13	Law	6
14	Chemistry and mineralogy	3
15	Philosophy and psychology	3
16	Transport	25
17	Business, economics, and finance	11
18	Sport and recreation	162
19	Biology	46
20	Language and linguistics	1
21	Geology and geophysics	14
22	Literature and theatre	56
23	Meteorology	2
24	Food and drink	10
25	Education	6
26	Engineering and technology	16
27	Geography and places	48
28	Video gaming	2
29	Mathematics	0
30	Culture and society	14

Figure 5.5: Domains export to the Excel file

In the case of export to CSV file we used classic way of writing into the file with the help of class *Writer* and method *writeLine* which writes data line by line with the specified column separator. In the case of export to excel file we decided to use *XSSFWorkbook* java library which provides possibility to create Workbook and particular Sheets and finally provides writing into concrete cells. Because of the using of this library we had to use also *OPCPackage* library and its method *open()* we used for opening already existing workbook stored on server. So we have two templates stored on the server – one template for domain export and one template for analysis export. In templates are defined titles, graphs, text formatting, etc. In the case of export to N-Triples format we used classic way of writing data into file with the help of class *PrintWriter* and method *println()*. Example of the exported file to Excel is shown in the picture 5.5.

5.6 Domain import

For domain import we used basic processing of the file in Java. We had to exactly specify the format of particular RDF triple for different using, for instance inserting new domain, adding entity to existing domain or specify link between entities. For user, this format is described in the form for import domains and also in the page *About*. Validation of triples is running during import and after finishing, user will be informed about failed ones. Format with concrete examples for particular operations available in import is described below.

Examples of particular operations according to table 5.1:

- **Inserting of new domain**

Table 5.1: Format for importing domains and entities

Operation	**Subject**	**Predicate**	**Object**
create new domain	domain name in URI format	rdfs:label	value, e.g "Sport"
add new entity to domain	entity path in URI format	purl:subject	domain object
define link between entities	entity path in URI format	owl:sameAs	entity path in URI format

```
<http://mydomains.com/domains/SportAndRecreation> <
    http://www.w3.org/2000/01/rdf-schema#label> "
    Sport and recreation".
```

- **Add new entity to domain**

```
<http://dbpedia.org/ontology/SportsSeason> <http://
    purl.org/dc/terms/subject> <http://mydomains.com
    /domains/SportAndRecreation> .
```

- **Link between entities**

```
<<http://dbpedia.org/ontology/SportsEvent> <http://
    www.w3.org/2002/07/owl#sameAs> <http://dbpedia.
    org/ontology/SportsSeason> .
```

The same format of RDF triples is used in case of exporting domains to N-Triples.

5.7 Jersey 2

We already decided in the design part, that this application will be build on REST architecture, so the way of communication between server and client is providing with JAX-RS technology. Already in design part we made the decision to use framework Jersey to provide this kind of architecture, which improving this technology and also it is easier to use. By the help of Jersey, in every web service there will be specified several issues by concrete annotation, like:

- HTTP method (@GET,@PUT,@POST, @DELETE),
- path to server (@Path),
- format of data, which web service consumes (@Consumes),
- format of data, which web service produces (@Produce),

- parameter filled in the path (@PathParam), for instance id of dataset.

After some research, at the beginning of the implementation, we were using technology Jersey 1, because we supposed, that will be more stable as its the newest version Jersey 2 and that Jersey 2 does not extend Jersey 1. The most of data forms we serialize to JSON format and send it to web service through AJAX. In case of the form with one input, we send directly this one value in string format. So we are using this types of annotation for specifying consuming data: *MediaType.APPLICATION_JSON* or *MediaType.PLAIN_TEXT*. In case that web service producing data we are using for annotation of data *MediaType.APPLICATION_JSON*. In case that web service does not return any data, we are using *MediaType.PLAIN_TEXT* annotation and web service returns instance of the class *Response*.

One problem with using Jersey 1 showed up when we need to send file from client to server, for instance in case of domain or dataset import. We discovered that Javascript is safe enough, that in case processing file from client is path to file in the form *//fakepath/domains.nt*. In Jersey 1 we could process file only in one way, which is getting binary data from the file and send it to the server, where data would be processed by the help of class *InputStream*.

Another problem in Jersey 1 was, that we were not able to handle asynchronous operation, which was needed in case dataset importing or dataset analysing calculation, which can be time consuming.

Finally we decided to use Jersey 2 instead of Jersey 1, which is able to handle of mentioned issues. Jersey 2 supports class *FormData*, which can process entire form including file and send it to server. In this web service in annotation *@Consumes* has to be used format of data *MediaType.MULTIPART_FORM_DATA*. Instead of annotation (@PathParam), there has to be used annotation *@FormDataParam*, according to which server recognizes, which parameter are from the form. In case that the form includes also the file, this annotation could be used more than once for the same parameter. For instance, we used it ones for getting input stream from file, and second one for getting of detailed information about the file by usage of class *FormDataContentDisposition*. This class provides useful information about file, for instance name of file, size of file or format.

Next advantage of using Jersey 2 is, that it can handle and process asynchronous operation in simple way by using of annotation *@Suspended* with *AsyncResponse* param in this method. It provides receiving of information about processed request after finishing it, but user is not blocked, he just can not refresh the page and he has to stay on it. For providing of the notification about data processing, the user will see displayed div with this information with blue color.

We are very satisfying with the switch from Jersey 1 to Jersey 2, because it includes lot of great functionality, it is not unstable and it supports everything

we need for the implementation of this application.

5.8 Performance optimization

Because it takes a lot of time for processing of huge amount of data, we tried to improve it in some way. The first speed improvement is that we used threads in Java by usage of the method *Executors.newFixedThreadPool()*. This method helps to create particular threads. Another class we had to use for it is the heritable class *Callable()*, which contains method *call()*, where is implemented logic, which will be running in threads. We used threads only in two sections of calculation, which could run parallel, so independently of each other. These sections are: calculation of entity proportion in dataset and calculation of predicates(which belong to the particular entity) proportion in dataset. Obviously, we had to use database lock with implementation of threads in application. This database lock is provided by Java class *ReentrantLock*, specifically by implementation of static class with static methods *getDbLock()*, *lock()* a *unlock()*.

In the case of reading HDT file, there could be another speed improvement, by the usage of method *HDTManager.mapHDT()*. Instead of the method *HDTManager.loadHDT()*, this method does not load everything
into the memory. The main advantage is faster loading and processing data from the file. Disadvantage is slower searching, but only the first time. Anyway, we could not use the method *HDTManager.loadHDT()* because of processing huge amount of data like DBpedia or Geonames, which caused lack of memory.

With displaying data to the user, data of the specific dataset are loading after first click on it. It means, that initial loading page is faster, and first loading of the specific dataset is slower.

5.9 Conclusion

During the implementation we tried to follow design and designed class model which was almost fulfilled. Finally we didn't divide class *DbAccess* to several subclasses and all database operations are involved in one class. Obviously, by the implementation of specific parts of the application, there shows up some issues, for which we had to create new classes, for instance class for logging exceptions, classes for testing rest client with method Get or method Post, etc. The biggest problems during implementation were in size of the data and lack of memory, where we had to increase Java memory.

Source of the application is stored in CD and also in Github in the address https://github.com/jcabaiova/RDFDataAnalyser.

Chapter 6

Testing

Testing is the last phase in the life time cycle of the software application. We decided to write automatic integrate test as the base of the testing. These tests are focused to test the basic user functionalities. This chapter also includes described application use cases and demonstration scenarios of the application usage.

6.1 Integrated automatic tests

The best way for creation of integrated tests for testing simple basic functionality of the application is to create these test automated. Because our application is in the web, we decided to use program Selenium IDE [8] for creation of automated tests. Selenium IDE is the plugin for the web browser Mozila Firefox. Its primary goal is creation of the integrated automatic tests for front-end testing. It supports simple recording of particular commands, but these command can be also written by hand. It contains lot of testing methods, for instance assert commands, verify commands, etc. For the selection of particular elements on the web page, it is possible to use id's of elements, css selectors or javascript selectors. This application supports creation of test suites which contains test cases. Furthermore, it is very simple for debugging of particular test case.

For our application, we decided to create only one test suite, which contains specific test cases for the page *Dataset Analysis* and also for the page *Domain List*. In these test cases we mainly focused on the form validation, showing or hiding particular elements and on verification of server response for our particular request. In case of these tests we do not check concrete update of data in the database, but the database should not contain the incorrect data based on the good quality of front-end validation.

[8] http://www.seleniumhq.org/projects/ide/

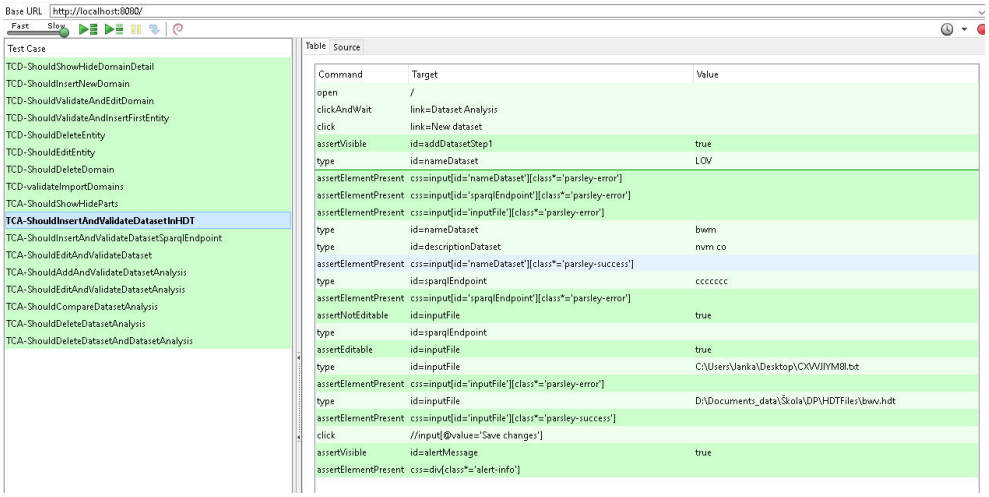


Figure 6.1: Automating integration tests in Selenium IDE

We made the special instance of the database for the running these tests, which contains several prepared domains, entities and datasets. Because of that, we are capable of testing not only adding new items, but also editing already existing items, deleting items and also validation of the unique names. This test database is the part of source code.

There are examples of questions which should answer to individual test cases:

- Was the form validation correct in case of incorrect input? Did application refuse to send request to the server?
- Was the form validation correct in case of correct input? Did application allow to send request to server?
- Did the application show right information about update data in database?
- Did the application show right title of the element after updating data in database?
- Did the application show or hide right HTML element after clicking on the particular part of the application?

We tried to write these tests continuously during the creation of application. We always tried to run it after the implementation of the bigger change. Tests together with source code of the application are stored in CD, which is the part of the paper.

6.2 Testing among different web browsers

We made manually testing of individual functionalities in the most used web browsers according to research from November 2016 from the portal *w3school* [36]. According to this portal, the most used web browsers are:

1. Chrome - 73,8 %,
2. Mozila Firefox - 15,3 %,
3. Internet Explorer, resp. Edge - 5,2 %,
4. Safari - 3,5 %,
5. Opera - 1,1 %.

Testing among different web browsers was simple, without any bigger differences. So the application is tested on these versions of the particular web browsers:

- Chrome - from the version 55.0.2883.75,
- Mozila Firefox - from the version 50.0.2,
- Microsoft Edge - from the version 38.14393,
- Safari - from the version 5.1.7,
- Opera - from the version 41.0.2353.69.

6.3 Application Use Cases and Demonstration Scenario

This application can be used in different cases. The most used case is, that user has some dataset, and he needs to know, what this dataset contains in sense of the representation in particular domains. Another case can be comparison of two already existing datasets. Particular application use cases are described in the next sections.

6.3.1 How can be defined a domain

The user has some RDF dataset, and if he wants the analysis, he has to define domains and entities first, if they do not already exist. He has to get ontology classes of the concrete dataset, create entities from them and added them manually to domains. If an expected domain doesn't exist, he has to create it too.

One domain in this application is represented by a triple, in which subject is RDF link to domain, predicate is *rdfs:label* and object is a literal with the

name of the domain. One entity in this application is represented by a triple, in which subject is RDF link to entity, predicate is *purl:subject* and object is the RDF link to domain. Entity or domain can be defined in two ways: importing of RDF file in N-Triples format with triples represented domains or entities, or inserting one entity or one domain by using the applicable form.

6.3.2 How can be used a domain to analyse a dataset

In the application are defined domains and entities about the particular dataset and dataset is already imported in the application. Then the user can run a new analysis over the dataset, where he defines domains (from existed ones) which will be used for the calculation of analysis. The analysis is calculated on the base of number of instances of the particular entities belongs to chosen domains in the dataset.

6.3.3 How a dataset can be imported

If the user has a dataset, which it is not imported in the application, he can import a new one, where he has to fill-out name of dataset, SPARQL endpoint to dataset or path to HDT file, ontology predicate, which is the predicate used in the particular dataset ontology and chose between short or long calculation. With the importing of new dataset, the default dataset analysis over all actual domains is calculating too.

6.3.4 How datasets can be compared

User can compare two dataset analysis over two different datasets. He chose first dataset, first dataset analysis, second dataset, second dataset analysis and comparison table will show. This table shows the summary representation of the domains and also the representation of the particular entities. In case of the entities in different dataset are linked with the predicate owl:sameAs (defining by chosing the same entityGroup during creation of entity), they are shown in one row for the better comparison.

6.3.5 Demonstration Scenario

Here is the demonstration scenario of the application usage. The user has RDF dataset in HDT format or link to its SPARQL endpoint (for instance DBpedia) and he needs to see insights. Any domains or entities are not created yet, so it is an empty database.

1. The user needs to define domains.

2. The user needs to define entities according to dataset ontology.

3. The user can import particular dataset and wait for the default calculation over all domains in database.

4. When the calculation is over, the user can view the result.

5. If he thinks, that for the calculation of the analysis were used too many domains, he can run another calculation with the specification of domains which to use. Or, he can also edit list of domains and entities.

6. Next, the user has another dataset, for instance GeoNames, and he wants to compare this one with the already imported dataset DBpedia.

7. So, he has to define entities from the GeoNames ontology, and also specify links between entities in DBpedia and GeoNames.

8. After that, he can import another dataset and wait for the result

9. Finally, the user goes to the compare mode, chose these two dataset and view the comparison table with the representation of domains and entites in these two datasets.

Chapter 7

Experiments and results

According to the work assignment, we need to test this application on real big Linked Open Data datasets, like DBpedia or GeoNames.

7.1 Domain initialization

In the first place, we needed to create initiate list of domains. Particular domain should represents some life area, for instance sport, medicine, culture, etc. There exist some domain classification systems which we could use. One of them is for instance WordNet Domains [37]. WordNet Domains represents a language-independent resource composed of 164, hierarchically organized, domain labels (e.g. Architecture, Sport, Medicine). It could be used in our application as initiative domains, but for the testing it contains a lot domains.

For initialization of domain we decided to use domain according to *Wikipedia:Featured Articles* [9]. Featured articles are considered to be the best articles Wikipedia has to offer and these articles are divided to thirty main areas, which we used as the base of our domain list.

7.2 DBpedia ontology and calculation of analysis

We processed DBpedia ontology mostly manually. This process contains identification of particular entities and theirs adding to concrete domain. At first, we create different program in Java, which get every entities from the ontology, so triples with the subject *owl:Class*. These entities we used for manually adding to the particular domain. Because of the big amount of entities, we tried to simplify a little this allocation, so the external program return not only entities, but also subclasses of entities, defined by predicate *rdfs:subClassOf*. In straightforward cases we allocated every entity belonged to particular subclass to the one domain. In numbers, we processed 586 different entities

[9] Available from: https://en.wikipedia.org/wiki/Wikipedia:Featured_articles

in DBpedia. Ontology of DBpedia is described in RDF format RDF/XML. Example of the one entity in this ontology is shown in listing 7.1.

```
<owl:Class rdf:about="http://dbpedia.org/ontology/
    BasketballLeague">
    <rdfs:label xml:lang="de">Basketball-Liga</
        rdfs:label>
    <rdfs:label xml:lang="fr">ligue de basketball</
        rdfs:label>
    <rdfs:label xml:lang="en">basketball league</
        rdfs:label>
    <rdfs:label xml:lang="nl">basketbal competitie</
        rdfs:label>
    <rdfs:label xml:lang="el">&#x39F;&#x3BC;&#x3BF;&#
        x3C3;&#x3C0;&#x3BF;&#x3BD;&#x3B4;&#x3AF;&#x3B1;
        &#x39A;&#x3B1;&#x3BB;&#x3B1;&#x3B8;&#x3BF;&#x3C3
        ;&#x3C6;&#x3B1;&#x3AF;&#x3C1;&#x3B9;&#x3C3;&#x3B7
        ;&#x3C2;</rdfs:label>
    <rdfs:label xml:lang="ja">&#x30D0;&#x30B9;&#x30B1;&#
        x30C3;&#x30C8;&#x30DC;&#x30FC;&#x30EB;&#x30EA;&#
        x30FC;&#x30B0;</rdfs:label>
    <rdfs:label xml:lang="it">lega di pallacanestro</
        rdfs:label>
    <rdfs:comment xml:lang="en">a group of~sports teams
        that compete against each other in~Basketball</
        rdfs:comment>
    <rdfs:subClassOf rdf:resource="http://dbpedia.org/
        ontology/SportsLeague"/>
    <prov:wasDerivedFrom rdf:resource="http://mappings.
        dbpedia.org/index.php/
        OntologyClass:BasketballLeague"/>
</owl:Class>
```

Listing 7.1: One class in file DBpedia Ontology (in our application it represents one entity)

We tried to find another already existing allocation of entities to domains. We found project Umbel [10] (Upper Mapping and Binding Exchange Layer), in which entities from DBpedia (and from other datasets too) were mapped to their own ontology. Unfortunately, this Umbel ontology was described in more details than we needed fo allocation to our only 30 defined domains.

After successful import of domains and entities (the imported file is added in CD) we imported dataset DBpedia in format SPARQL endpoint for testing this possibility of importing the big data. Example of chart from DBpedia

[10] Available from: https://github.com/structureddynamics/UMBEL

analysis is shown in the figure 7.1. Finally, the analysis shows, that the biggest representation is created by domain Media with 31,9 %, followed by the domain Geography and places with 15,2% and the third place belongs to domain Sport and recreation with 14 %. The most three represented entities in particular domains are shown in the table 7.1. Representation of domains in dataset DBpedia is shown in chart 7.2.

Domain / Entity name	Count
Media	**2913106**
http://dbpedia.org/ontology/Image	2753948
http://dbpedia.org/ontology/TelevisionShow	36979
http://dbpedia.org/ontology/Broadcaster	29256
Geography and places	**1388140**
http://dbpedia.org/ontology/PopulatedPlace	505557
http://dbpedia.org/ontology/Settlement	472250
http://dbpedia.org/ontology/Village	166959
Sport and recreation	**1289395**
http://dbpedia.org/ontology/Athlete	331680
http://dbpedia.org/ontology/SportsTeamMember	323111
http://dbpedia.org/ontology/SoccerPlayer	110947
Biology	**840137**
http://dbpedia.org/ontology/Eukaryote	294585
http://dbpedia.org/ontology/Animal	226263
http://dbpedia.org/ontology/Insect	131244
Music	**524083**
http://dbpedia.org/ontology/MusicalWork	199640
http://dbpedia.org/ontology/Album	135329
http://dbpedia.org/ontology/MusicalArtist	50978
Literature and theatre	**443572**
http://dbpedia.org/ontology/Film	106613
http://dbpedia.org/ontology/Artist	79353
http://dbpedia.org/ontology/WrittenWork	64259
Culture and society	**377645**
http://dbpedia.org/ontology/OrganisationMember	323111
http://dbpedia.org/ontology/SocietalEvent	51298
http://dbpedia.org/ontology/Museum	5341
Art,architecture, and archaeology	**373429**
http://dbpedia.org/ontology/ArchitecturalStructure	188490
http://dbpedia.org/ontology/Infrastructure	87966
http://dbpedia.org/ontology/Building	78246
Transport	**137492**
http://dbpedia.org/ontology/Ship	29485
http://dbpedia.org/ontology/RouteOfTransportation	28999

Domain / Entity name	Count
http://dbpedia.org/ontology/Road	20384
Business, economics, and finance	**137123**
http://dbpedia.org/ontology/Company	67544
http://dbpedia.org/ontology/OfficeHolder	63693
http://dbpedia.org/ontology/Convention	2104
Politics and government	**109817**
http://dbpedia.org/ontology/Politician	40260
http://dbpedia.org/ontology/AdministrativeRegion	24409
http://dbpedia.org/ontology/Election	9315
Education	**107657**
http://dbpedia.org/ontology/EducationalInstitution	53584
http://dbpedia.org/ontology/School	32186
http://dbpedia.org/ontology/University	19777
Engineering and technology	**93151**
http://dbpedia.org/ontology/Software	32334
http://dbpedia.org/ontology/AutomobileEngine	27699
http://dbpedia.org/ontology/Scientist	23373
Warfare	**70324**
http://dbpedia.org/ontology/MilitaryPerson	29007
http://dbpedia.org/ontology/MilitaryUnit	17591
http://dbpedia.org/ontology/MilitaryConflict	13660
Heraldry, honors, and vexillology	**66894**
http://dbpedia.org/ontology/WorldHeritageSite	968
http://dbpedia.org/ontology/Monument	564
Geology and geophysics	**65791**
http://dbpedia.org/ontology/BodyOfWater	42274
http://dbpedia.org/ontology/Mountain	16752
http://dbpedia.org/ontology/MountainRange	2493
Religion, mysticism and mythology	**41444**
http://dbpedia.org/ontology/Cleric	15485
http://dbpedia.org/ontology/ChristianBishop	7715
http://dbpedia.org/ontology/ReligiousBuilding	4349
History	**36714**
http://dbpedia.org/ontology/HistoricPlace	22194
http://dbpedia.org/ontology/HistoricBuilding	8413
http://dbpedia.org/ontology/Historian	766
Video gaming	**20149**
http://dbpedia.org/ontology/VideoGame	20419
Royalty and nobility	**20296**
http://dbpedia.org/ontology/Royalty	11000
http://dbpedia.org/ontology/Noble	5244

Domain / Entity name	Count
http://dbpedia.org/ontology/Monarch	2454
Physics and astronomy	**11349**
http://dbpedia.org/ontology/Planet	3690
http://dbpedia.org/ontology/Star	2869
http://dbpedia.org/ontology/Asteroid	1962
Chemistry and mineralogy	**11294**
http://dbpedia.org/ontology/ChemicalCompound	10026
http://dbpedia.org/ontology/Mineral	1268
Law	**8867**
http://dbpedia.org/ontology/Judge	2915
http://dbpedia.org/ontology/SupremeCourtOfTheUnitedStatesCase	2724
http://dbpedia.org/ontology/LegalCase	2724
Food and drink	**3920**
http://dbpedia.org/ontology/Restaurant	1158
http://dbpedia.org/ontology/Beverage	911
http://dbpedia.org/ontology/Chef	564
Health and medicine	**3449**
http://dbpedia.org/ontology/Hospital	3019
http://dbpedia.org/ontology/Medician	430
Philosophy and psychology	**1768**
http://dbpedia.org/ontology/Philosopher	1768
Mathematics	**0**
Computing	**0**
Language and linguistics	**0**
Meteorology	**0**

Table 7.1: DBpedia analysis - three most occurring entities in every domain in LOD dataset DBpedia

7.3 GeoNames ontology and calculation of analysis

In the processing of GeoNames ontology we used two kinds of files from source GeoNames [11]. The first one was the ontology in RDF/XML format. The main problem with processing this ontology was in naming of the entities. It was not understandable by human, because the name consist of only some shortcut. So, we decided to map Umbel ontology to GeoNames ontology for the recognition and insertion to right domain, because umbel ontology adds to every entity human understandable name. Another way for this we could follow predicate *prefLabel*. We processed this file in different program and manually

[11] http://www.geonames.org/ontology/documentation.html

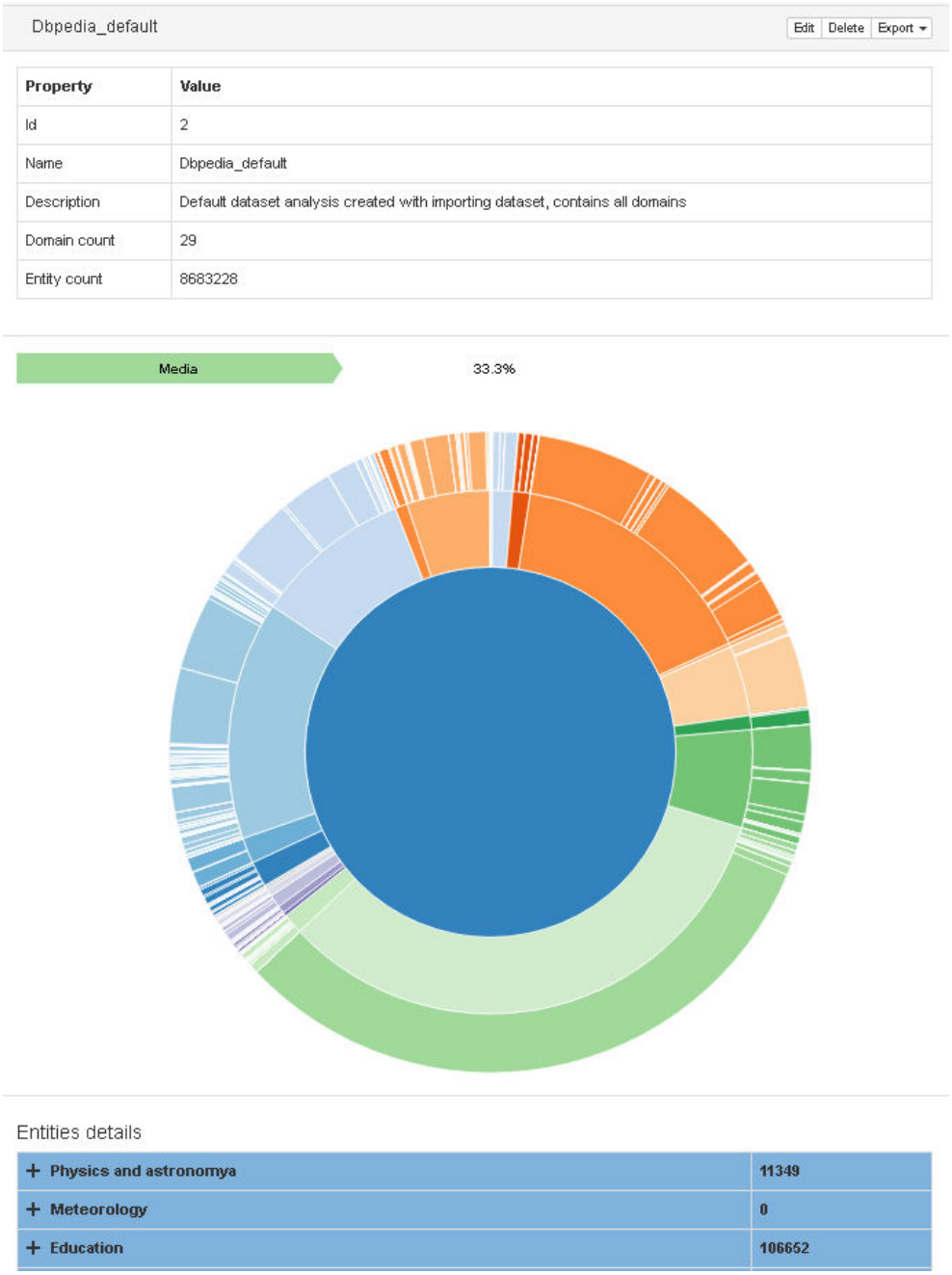


Figure 7.1: DBpedia analysis - chart in the application showing the representation of domains and entities in DBpedia

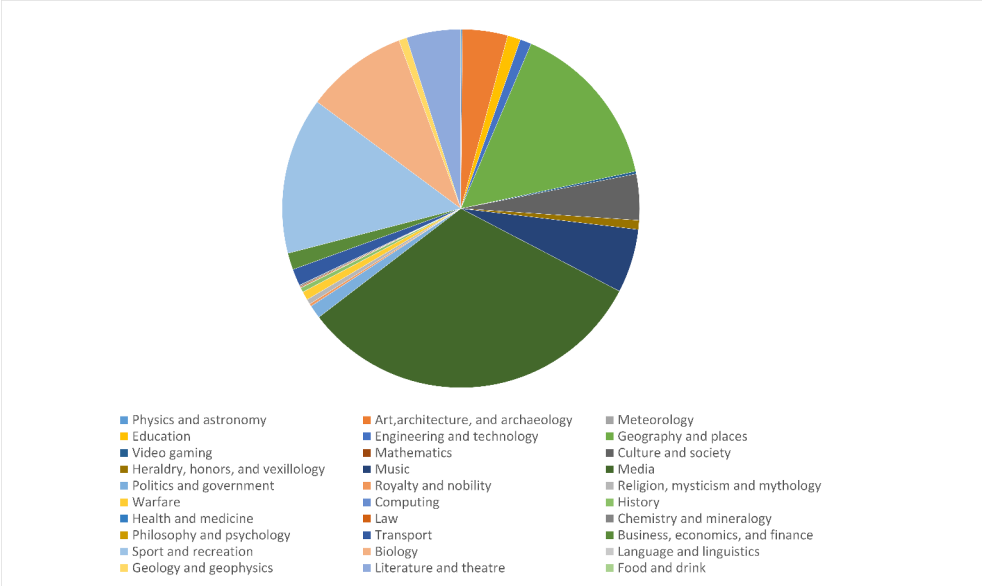


Figure 7.2: Domain representation in DBpedia

inserted it to concrete domains. Example of one entity in GeoNames ontology is shown in listing 7.2.

```
<gn:Code rdf:about="#S.VETF"
                    skos:notation="S.VETF">
    <skos:definition xml:lang="en">a building or camp at
        ˜which veterinary services are available</
        skos:definition>
    <skos:inScheme rdf:resource="#S"/>
    <skos:prefLabel xml:lang="en">veterinary facility</
        skos:prefLabel>
  </gn:Code>
```

Listing 7.2: One class in the file GeoNames Ontology

Another file we used from source GeoNames was mapping. This file is also in format RDF/XML and it contains some mapping between GeoNames and DBpedia or between GeoNames and LinkedGeoData. One mapping example in this file is shown in the listing 7.3. We used every mapping from this file to our application, not only between dataset DBpedia and Geonemas.

```
<owl:Class rdf:about="http://dbpedia.org/ontology/
    Theatre">
        <rdfs:label xml:lang="en">Theatre</rdfs:label>
        <owl:equivalentClass rdf:resource="http://
            linkedgeodata.org/ontology/Theatre"/>
```

```
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www
                    .geonames.org/ontology#featureCode"/>
                <owl:hasValue rdf:resource="http://www.
                    geonames.org/ontology#S.THTR"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="http://www.
            geonames.org/ontology#Feature"/>
</owl:Class>
```

Listing 7.3: One class in the GeoNames mapping file

After successful import of entities and linking between entities (the imported file is added in CD) we imported dataset GeoNames in format HDT for testing also this possibility for the big data. Finally, the analysis shows, that the biggest representation is created by domain Geography and places with 61,9 %, followed by the domain Geology and Geophysics with 17 % and the third place belongs to domain Religion, mysticism and mythology with 5,34 %. The most three represented entities in particular domains are shown in the table 7.2. Representation of domains in dataset GeoNames is shown in the chart 7.3.

Domain / Entity name	Count	Umbel label
Geography and places	**5110941**	
http://www.geonames.org/ontology#P.PPL	2853189	PopulatedPlace
http://www.geonames.org/ontology#H.STM	763142	Watercourse
http://www.geonames.org/ontology#H.LK	249677	Lake
Geology and geophysics	**1407207**	
http://www.geonames.org/ontology#T.MT	316148	Mountain
http://www.geonames.org/ontology#T.HLL	174753	Hill
http://www.geonames.org/ontology#H.STMI	120878	Watercourse
Religion, mysticism and mythology	**440764**	
http://www.geonames.org/ontology#S.CH	237203	Church-Building
http://www.geonames.org/ontology#S.CMTY	136377	Graveyard
http://www.geonames.org/ontology#S.MSTY	25126	Monastery-Residence
Art,architecture, and archaeology	**299843**	
http://www.geonames.org/ontology#S.BLDG	118192	Building
http://www.geonames.org/ontology#H.WLL	78496	Well
http://www.geonames.org/ontology#P.PPLQ	32078	Abandoned Construction Artifact

Domain / Entity name	Count	Umbel label
Politics and government	**286760**	
http://www.geonames.org/ontology#A.ADM4	93067	FourthOrder Administrative Division
http://www.geonames.org/ontology#A.ADMD	64132	Administrative District
http://www.geonames.org/ontology#A.ADM3	56375	Third Order Administrative Division
Education	**265184**	
http://www.geonames.org/ontology#S.SCH	250662	School Building
http://www.geonames.org/ontology#S.LIBR	11620	Library Space
http://www.geonames.org/ontology#S.SCHC	1915	School Campus
Transport	**218604**	
http://www.geonames.org/ontology#S.RSTN	52844	Railway
http://www.geonames.org/ontology#S.PO	42417	Post Office Building
http://www.geonames.org/ontology#S.AIRP	23263	Airport Physical
Engineering and technology	**73190**	
http://www.geonames.org/ontology#P.PPLA4	23965	Fourth Order Seat
http://www.geonames.org/ontology#H.RSVT	20277	Water Tank
http://www.geonames.org/ontology#S.AIRF	7954	Airfield
Biology	**34537**	
http://www.geonames.org/ontology#V.FRST	20605	Forest
http://www.geonames.org/ontology#H.MRSH	6212	Marsh
http://www.geonames.org/ontology#L.RESF	4334	Forest Reserve
Sport and recreation	**34140**	
http://www.geonames.org/ontology#S.CMP	23659	Campsite
http://www.geonames.org/ontology#S.RECG	7246	GolfCourse
http://www.geonames.org/ontology#S.STDM	905	Stadium
Health and medicine	**27281**	
http://www.geonames.org/ontology#S.HSP	18686	Hospital Building
http://www.geonames.org/ontology#S.HSPD	5317	Medical Care Facility
http://www.geonames.org/ontology#S.HSPC	2750	Medical Care Facility
Business, economics, and finance	**23840**	

Domain / Entity name	Count	Umbel label
http://www.geonames.org/ontology#S.EST	16646	Estate-Legal Entity
http://www.geonames.org/ontology#S.MKT	2195	Marketplace Object
http://www.geonames.org/ontology#S.ESTX	1743	Estate-Legal Entity
History	**10710**	
http://www.geonames.org/ontology#S.MUS	5050	Museum Structure
http://www.geonames.org/ontology#S.ANS	2561	Ancient Site
http://www.geonames.org/ontology#S.HSTS	1339	Historical Site
Culture and society	**8380**	
http://www.geonames.org/ontology#L.TRB	6176	Tribal Region
http://www.geonames.org/ontology#L.AGRC	1645	Colony
http://www.geonames.org/ontology#S.COMC	316	Communications Facility
Warfare	**6592**	
http://www.geonames.org/ontology#S.INSM	2701	Military Facility
http://www.geonames.org/ontology#S.FT	2698	Military Base Grounds
http://www.geonames.org/ontology#S.PSTP	633	Military Post
Food and drink	**3707**	
http://www.geonames.org/ontology#V.OCH	1543	Orchard
http://www.geonames.org/ontology#S.REST	1343	Restaurant Space
http://www.geonames.org/ontology#S.MLSG	192	SugafMill Manufacturing Facility
Meteorology	**1587**	
http://www.geonames.org/ontology#S.STNM	1587	Weather Station
Heraldry, honors, and vexillology	**796**	
http://www.geonames.org/ontology#S.MNMT	796	Monument
Law	**714**	
http://www.geonames.org/ontology#L.DEVH	437	Subdivision RealEstate
http://www.geonames.org/ontology#S.CTHSE	187	Courthouse
http://www.geonames.org/ontology#S.STNI	49	Inspection Facility
Physics and astronomy	**392**	

Domain / Entity name	Count	Umbel label
http://www.geonames.org/ontology#S.ASTR	208	Astronomical Station
http://www.geonames.org/ontology#S.OBPT	124	Observation Point
http://www.geonames.org/ontology#S.OBS	41	Astronomical Observatory
Literature and theatre	**299**	
http://www.geonames.org/ontology#S.THTR	299	Theater Space
Computing	**142**	
http://www.geonames.org/ontology#S.ITTR	142	Research Institute
Media	**139**	
http://www.geonames.org/ontology#S.STNR	139	RadioStation
Chemistry and mineralogy	**15**	
http://www.geonames.org/ontology#S.MLM	8	Mineral Ore Refinery
http://www.geonames.org/ontology#S.MFGLM	4	LimKiln
http://www.geonames.org/ontology#S.MFGCU	3	Copper Refinery
Mathematics	**0**	
Music	**0**	
Royalty and nobility	**0**	
Philosophy and psychology	**0**	
Language and linguistics	**0**	
Video gaming	**0**	

Table 7.2: GeoNames analysis - three most occurring entities in domains LOD dataset GeoNames

7.4 Comparison of dataset DBpedia and GeoNames

We made the comparison table 7.3 with domain representation in dataset DBpedia and GeoNames. From this table it is obvious, that entities which make the biggest difference are Geography and places, Geology and geophysics, which have bigger representation in dataset GeoNames. The most different domains which have bigger representation in DBpedia are Media, Sport and recreation and Biology. This result proves that dataset GeoNames is focused on geographic places, while dataset DBpedia is cross-domain dataset.

Table 7.3: Comparison of domain representation in datatses: DBpedia vs GeoNames

Domain name	**DBpedia**	**Geonames**
Physics and astronomy	11349	392
Art,architecture, and archaeology	373429	299843
Meteorology	0	1587
Education	107657	265184
Engineering and technology	93151	73190
Geography and places	1388140	5110941
Video gaming	20437	0
Mathematics	0	0
Culture and society	377645	8380
Heraldry, honors, and vexillology	75429	796
Music	524083	0
Media	2913106	139
Politics and government	109817	286760
Royalty and nobility	20296	0
Religion, mysticism and mythology	41444	440764
Warfare	70324	6592
Computing	0	142
History	36714	10710
Health and medicine	3449	27281
Law	8867	714
Chemistry and mineralogy	11294	15
Philosophy and psychology	1768	0
Transport	137492	218604
Business, economics, and finance	137123	23840
Sport and recreation	1289395	34140
Biology	840137	34537
Language and linguistics	0	0
Geology and geophysics	65791	1407207
Literature and theatre	443572	299
Food and drink	3920	3707

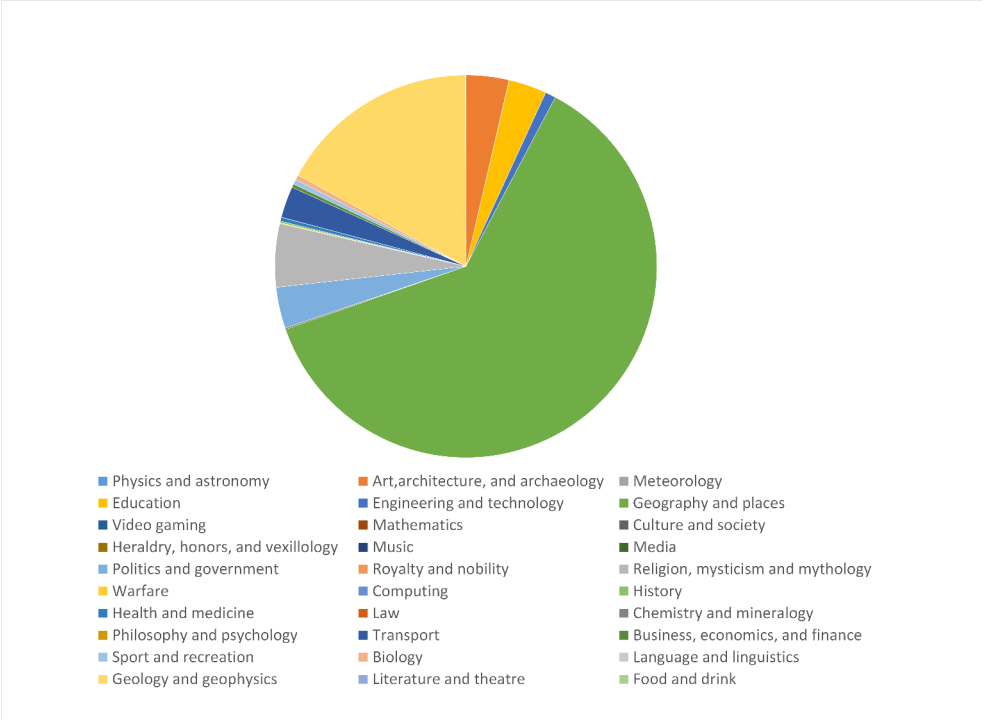


Figure 7.3: Domain representation in GeoNames

7.5 Conclusion about testing of the application on real data

During the testing of this application we found out, that not every dataset has human understandable name of entities, so we definitely could inserted not only path to entity, but also some human understandable name of entity, for instance use value in *pref:label* in ontology for it.

Within the this testing, the main problem was dataset size and a slow calculation. There occurs main problem with the entity *http://www.geonames.org/ontology#P.PPL* in dataset GeoNames, which is represented by 2,853,189 triples, and every subject in this triple has a lot of predicates. This entity causes fall of the application during the calculation of predicates (a long calculation) because of the lack of memory. We tried to fix it in several ways, for instance by dividing a query for predicates calculation to several smaller queries, but it did not help. Because this application is only a school work, we decided to remove this entity from the calculation of predicates. Anyway we hope we will find solution for this bug in the future.

Conclusion

The aim of this thesis was to introduce with the Linked Open Data and their particular principles. Simultaneously, it was necessary to explore the current state of LOD cloud. This aim was accomplished, so the work contains transparent definition and explanation of the particular terms together with the exploration of the current status of LOD and comparison with the previous date.

Besides the main principles of Linked Open Data, it was necessary to study the particular semantic technologies which are the base of Linked Open Data. They are mostly technologies RDF and SPARQL and also different accesses to these datasets by the help of SPARQL endpoint, Linked Data Fragments or HDT. The work explains the main elements of RDF data model, formats of RDF datasets and also the demonstrational possibilities for the creation of SPARQL queries. Also, it deals with the presentation of Linked Data Fragment or HDT format of LOD datasets.

The paper also contains research about existing technologies and tools which are focused on the summarization of LOD datasets in some way.

The other important aim of this work was to create analysis and based on it to design and to implement web application for the summarizing and comparison of Linked Open Data datasets.

Analytical part of the application was fulfilled by writing up the functional and non-functional requirements of the proposed application. Moreover, we processed particular use case diagrams with the concrete scripts, and also the domain model. Design part of the application was completed by the processing of the design models, concretely model of classes and database model. Individual analytical and design models are processed graphically by the help of the program Enterprise Architect. Part of the design was also the choice of the architecture and implementation platform which we accomplished by the summarizing of the particular technologies, which we decided to use for the creation of the proposed application. The other part of the design was the user interface design, which was fulfilled by the drawing wireframes

by the help of programme Balsamiq.

Implementation part was made by programming of the web application, which comes from the above mentioned design. Source code is saved in the CD, which is also the part of the paper. The most important parts of the implementation are also described in the work together with the screenshots in the appendix. Application is also available in the Github.

Test part was fulfilled by the programming of the automatic integrating tests for the testing basic functions of the application and also testing of application within the particular web browsers. The aim of the work was also to validate the application on the real datasets as DBpedia and GeoNames. This was also successfully accomplished and individual results are described in the work in more details.

Bibliography

[1] Cyganiak, R.; Jentzsch, A. The Linking Open Data cloud diagram. Available from: http://lod-cloud.net/

[2] Jakub Klímek, M. N. Introduction to Linked Data. Available from: https://edux.fit.cvut.cz/courses/MI-SWE/_media/lectures/mi-swe-p01-intro-en.pdf

[3] community, M. About microformats. Available from: http://microformats.org/wiki/about

[4] Wikipedia. Linked data. Available from: https://en.wikipedia.org/wiki/Linked_data

[5] Hausenblas, M. 5 star OPEN DATA. Available from: http://5stardata.info/en/

[6] Berners-Lee, T. Linked data. Available from: https://www.w3.org/DesignIssues/LinkedData.html

[7] Heath, T.; Bizer, C. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, first edition, 2011, ISBN 9781608454303. Available from: http://linkeddatabook.com/

[8] DBpedia. About. Available from: http://wiki.dbpedia.org/about

[9] Wick, M. GeoNames Ontology. Available from: http://www.geonames.org/ontology/documentation.html

[10] W3C. Resource Description Framework (RDF). Available from: http://wiki.dbpedia.org/about

[11] Wood, D.; Zaidman, M.; et al. *Linked Data Structured data on the web*. Manning, 2014.

[12] W3C. W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. Available from: https://www.w3.org/TR/xmlschema11-2/#string-derived-types

[13] W3C. RDF Schema 1.1. Available from: https://www.w3.org/TR/rdf-schema/

[14] W3C. OWL Web Ontology Language Reference. Available from: https://www.w3.org/TR/owl-ref/

[15] Wikipedia. Simple Knowledge Organization System. Available from: https://en.wikipedia.org/wiki/Simple_Knowledge_Organization_System

[16] W3C. Describing Linked Datasets with the VoID Vocabulary. Available from: https://www.w3.org/TR/void/

[17] W3C. SPARQL Query Language for RDF. Available from: https://www.w3.org/TR/rdf-sparql-query/

[18] Institute, D. E. R. Namespace lookup for RDF developers. Available from: https://prefix.cc/

[19] OpenLinkSoftware. Virtuoso server. Available from: https://virtuoso.openlinksw.com/

[20] RDF4J, E. The Eclipse RDF4J framework. Available from: http://rdf4j.org/

[21] Jena, A. The Apache Software Foundation. Available from: https://jena.apache.org/

[22] Multimedia Lab, G. U. Linked Data Fragments. Available from: http://linkeddatafragments.org/

[23] DataWeb. Download. Available from: http://www.rdfhdt.org/

[24] Langegger, A. RDFStats Manual for v2.0-beta. Available from: http://rdfstats.sourceforge.net/

[25] Demter, J.; Auer, S.; et al. LODStats—An Extensible Framework for High-performance Dataset Analytics. In *Proceedings of the EKAW 2012*, Lecture Notes in Computer Science (LNCS) 7603, Springer, 2012, p. 10, 29% acceptance rate. Available from: http://svn.aksw.org/papers/2011/RDFStats/public.pdf

[26] Ziawasch Abedjan, A. J. F. N., Toni Gruetze. Profiling and Mining RDF Data with ProLOD++. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE), Demo*, Chicago, IL, 0 2014, p. 4.

[27] Fabio Benedetti, L. P., Sonia Bergamaschi. LODeX: A tool for Visual Querying Linked Open Data. In *LODeX: A tool for Visual Querying Linked Open Data*, 2015, p. 4.

[28] Dudas, M. LodSight - An RDF dataset summarization too. Available from: `http://lod2-dev.vse.cz/lodsight-v2/index.html`

[29] Lalithsena, S.; Jain, P.; et al. (editors). *Automatic Domain Identification for Linked Open Data*, Atlanta, GA: ACM, 2013.

[30] Abele, A. Linked Data Profiling: Identifying the Domain of Datasets Based on Data Content and Metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2016, ISBN 978-1-4503-4144-8, pp. 287–291, doi:10.1145/2872518.2888603. Available from: `http://dx.doi.org/10.1145/2872518.2888603`

[31] W3Techs. Usage of client-side programming languages for websites. Available from: `https://w3techs.com/technologies/overview/client_side_language/all`

[32] W3Techs. Usage of JavaScript libraries for websites. Available from: `https://w3techs.com/technologies/overview/javascript_library/all`

[33] Oracle. Jersey - RESTful Web Services in Java. Available from: `https://jersey.java.net/`

[34] SQLite. About SQLite. Available from: `https://sqlite.org/about.html`

[35] Foundation, T. A. S. Welcome to Apache Maven. Available from: `https://maven.apache.org/`

[36] W3C. Browser Statistics. Available from: `http://www.w3schools.com/browsers/`

[37] Bentivogli, L.; Forner, P.; et al. Revising the Wordnet Domains Hierarchy: Semantics, Coverage and Balancing. In *Proceedings of the Workshop on Multilingual Linguistic Ressources*, MLR '04, Stroudsburg, PA, USA: Association for Computational Linguistics, 2004, pp. 101–108. Available from: `http://dl.acm.org/citation.cfm?id=1706238.1706254`

Appendix A

Acronyms

AJAX Asynchronous JavaScript And XML

API Application Programming Interface

CSV Comma Separated Values

DCAT Data Catalog Vocabulary

DCMI Dublin Core Metadata Initiative

FTP File Transfer Protocol

GUI Graphical User Interface

HDT Header, Dictionary, Triples

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

IBAN International Bank Account Number

JAX-RS Java API for RESTful Web Services

JSON JavaScript Object Notation

LOD Linked Open Data

OWL Web Ontology Language

PDF Portable Document Format

RDF Resource Description Framework

REST REpresentational State Transfer

SKOS Simple Knowledge Organization System

URI Uniform Resource Identifier

URL Uniform Resource Language

XML Extensible Markup Language

VoID Vocabulary of Interlinked Datasets

Appendix B

Content of enclosed CD

- `readme.txt` ..... short description of the CD content
- `src`
  - `impl` ..... source codes of the implemented application
  - `thesis` ..... source codes ot the paper in LaTeX
- `text` ..... text content of the paper
  - `thesis.pdf` ..... text content of the paper in PDF format

Appendix C

Wireframes of the application

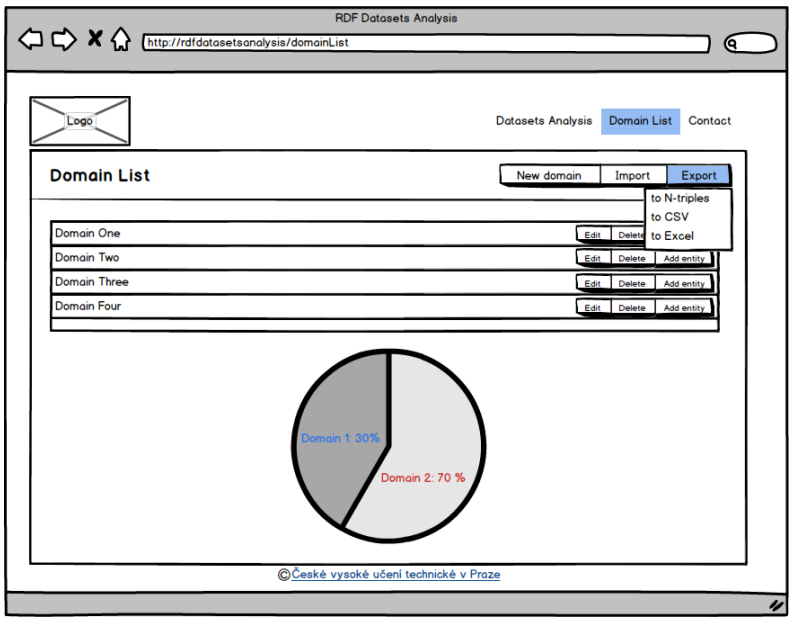


Figure C.1: Wireframe - DomainList with packed domains and with opened button for theexport

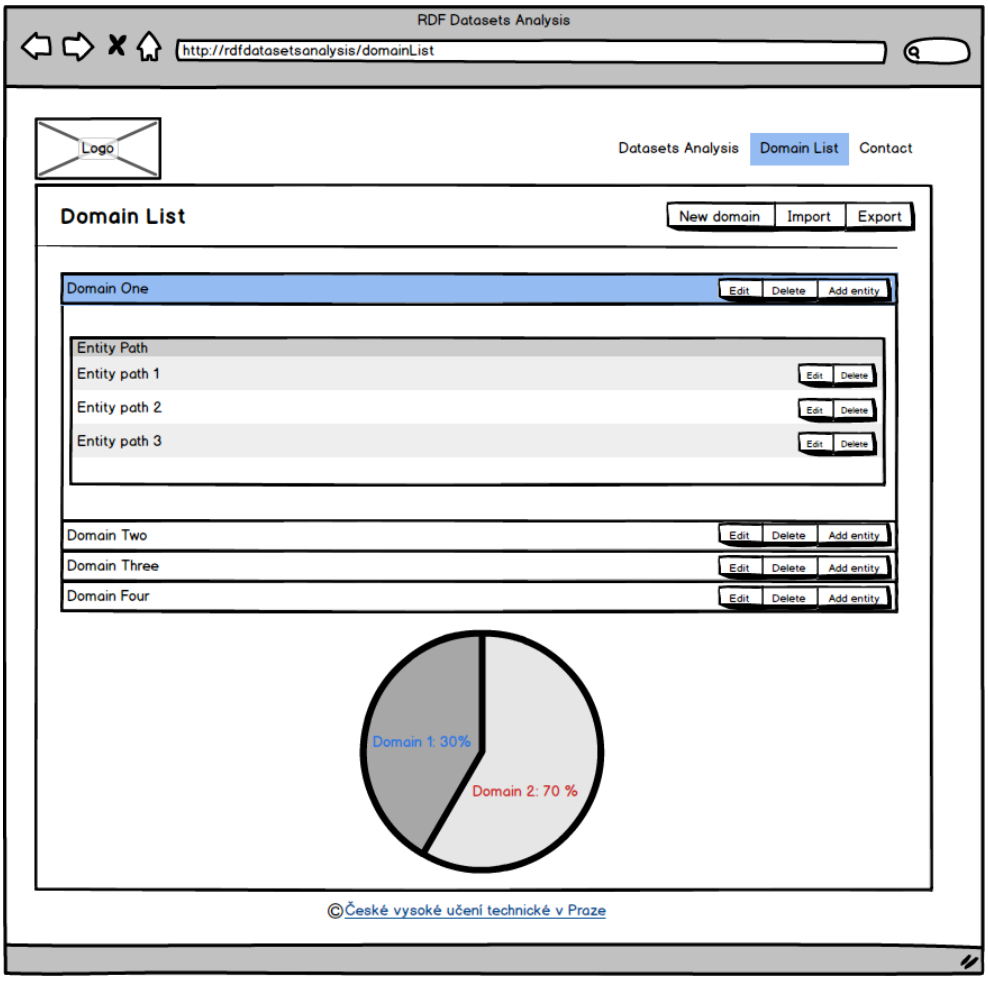


Figure C.2: Wireframe - DomainList with unpacked one domain where the list of entities is displayed

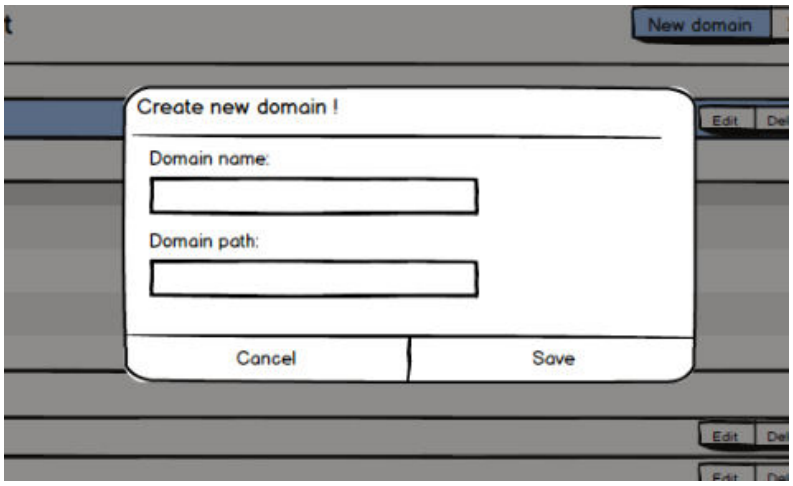


Figure C.3: Wireframe - DomainList with opened form for inserting new domain

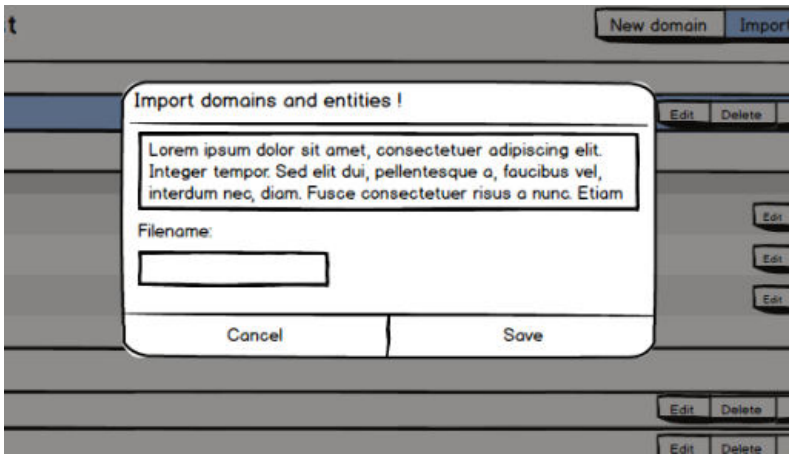


Figure C.4: Wireframe - DomainList with opened form for importing domains and entities

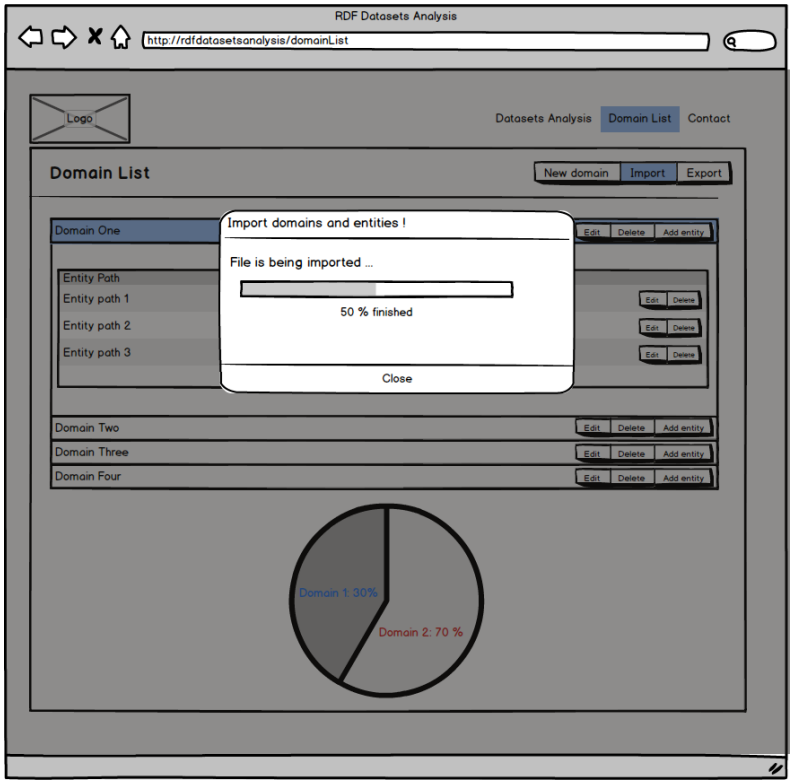


Figure C.5: Wireframe - DomainList after sending data to the server for processing

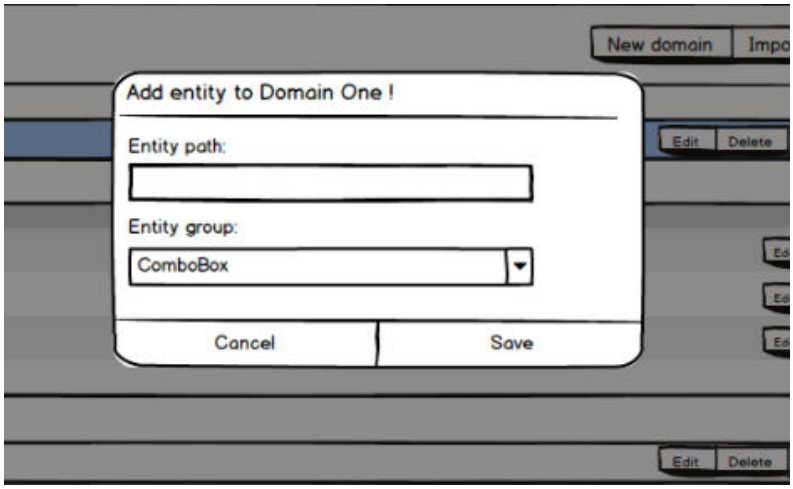


Figure C.6: Wireframe - DomainList with opened form for inserting new entity

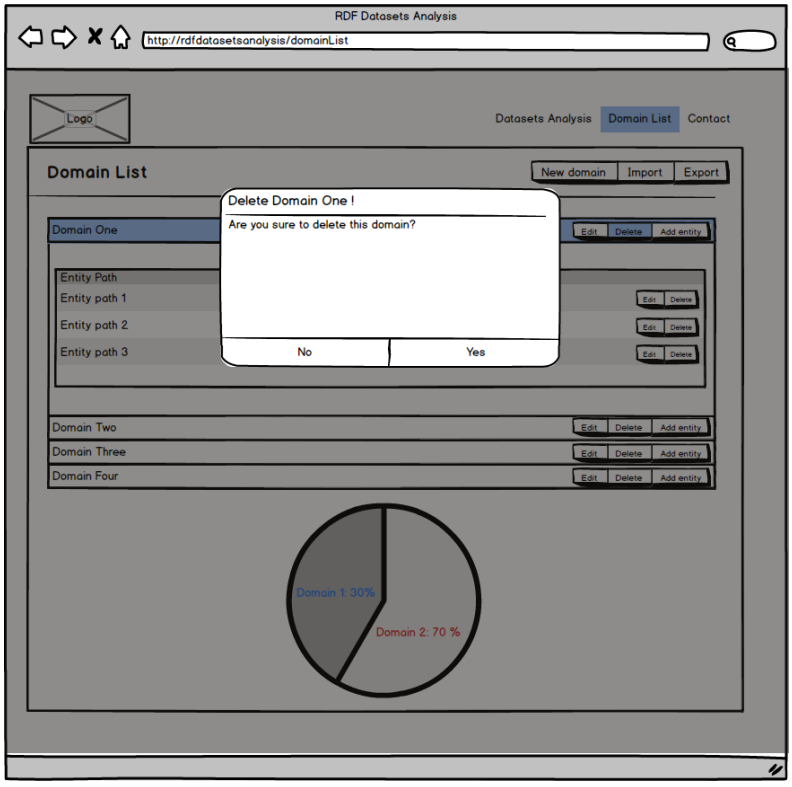


Figure C.7: Wireframe - DomainList with opened form for deleting existing domain

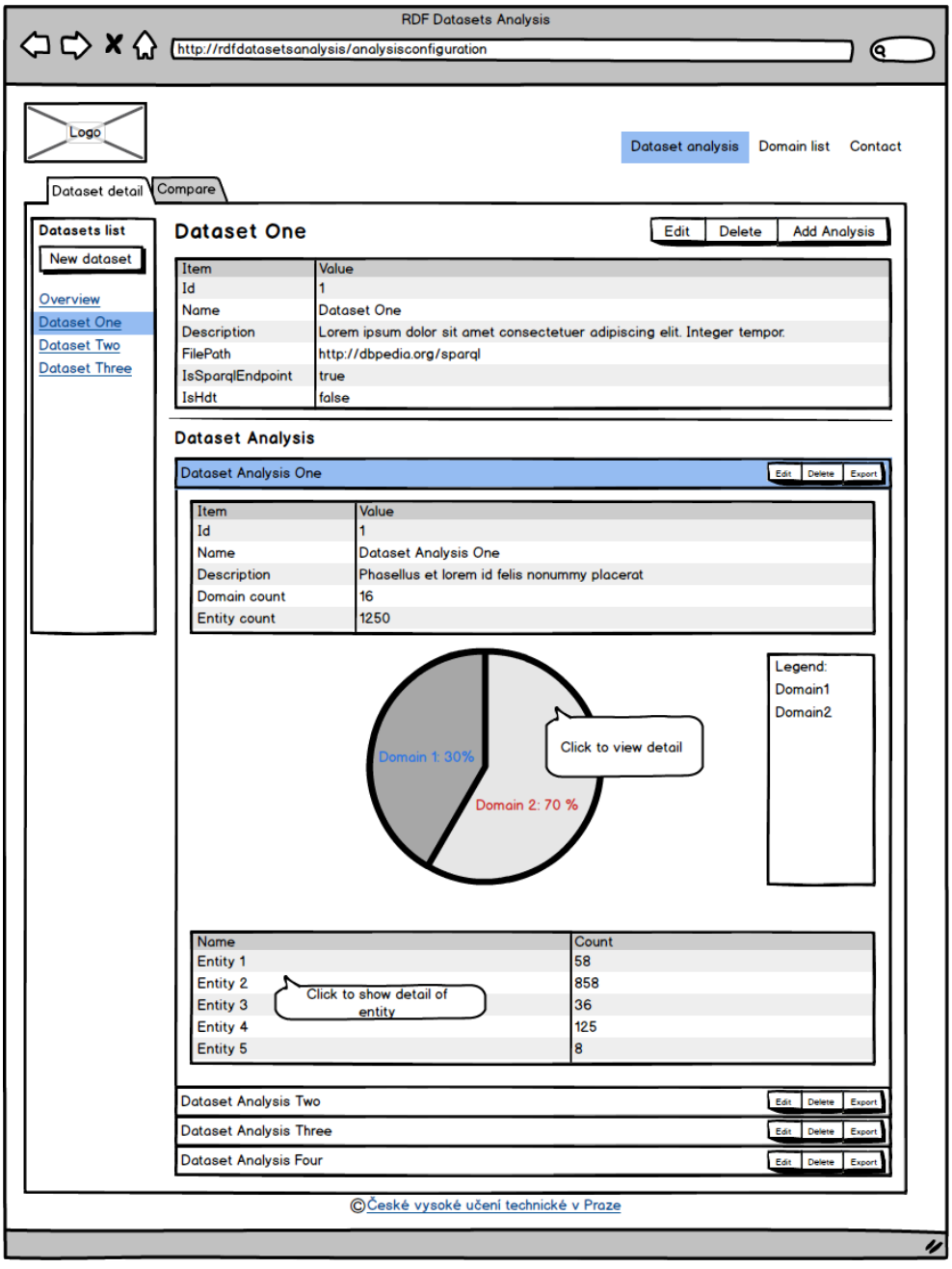


Figure C.8: Wireframe - DatasetAnalysis with opened detail of dataset analysis

Figure C.9: Wireframe - DatasetAnalysis with no datasets imported

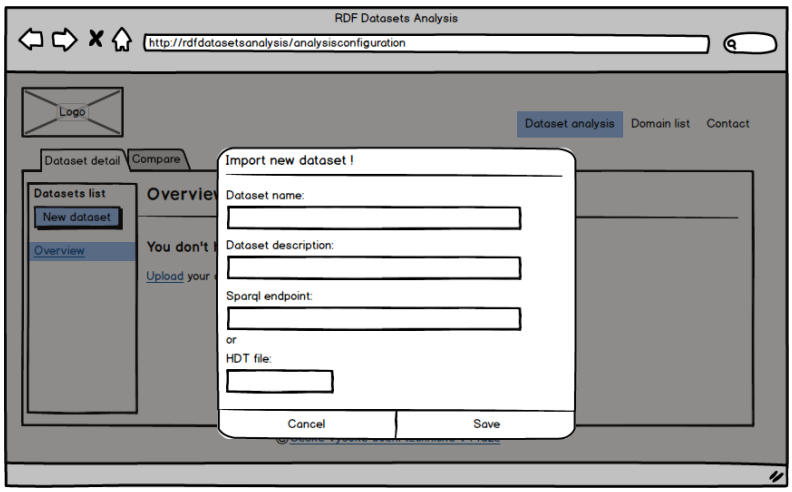


Figure C.10: Wireframe - DatasetAnalysis with opened form form inserting new dataset

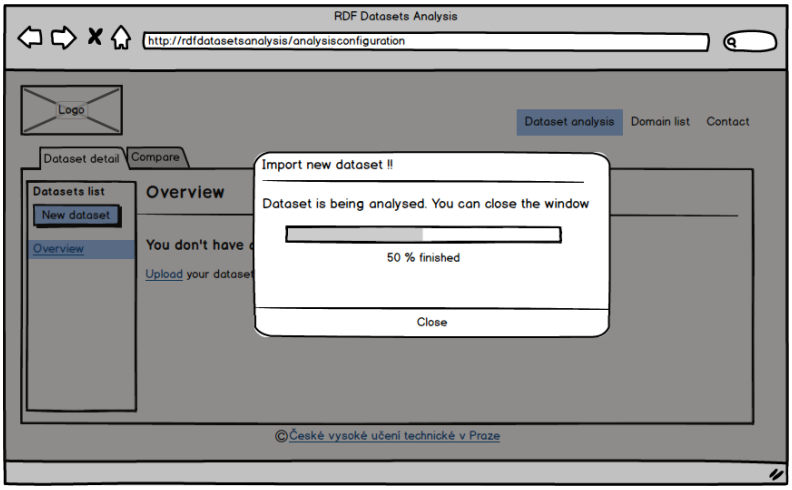


Figure C.11: Wireframe - DatasetAnalysis after sending data to server for processing

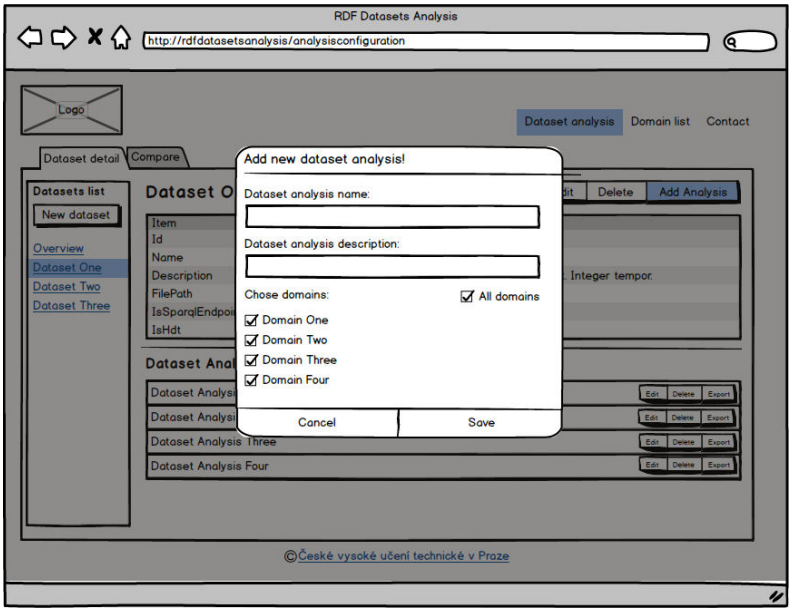


Figure C.12: Wireframe - DatasetAnalysis detail with opened form for inserting new dataset analysis

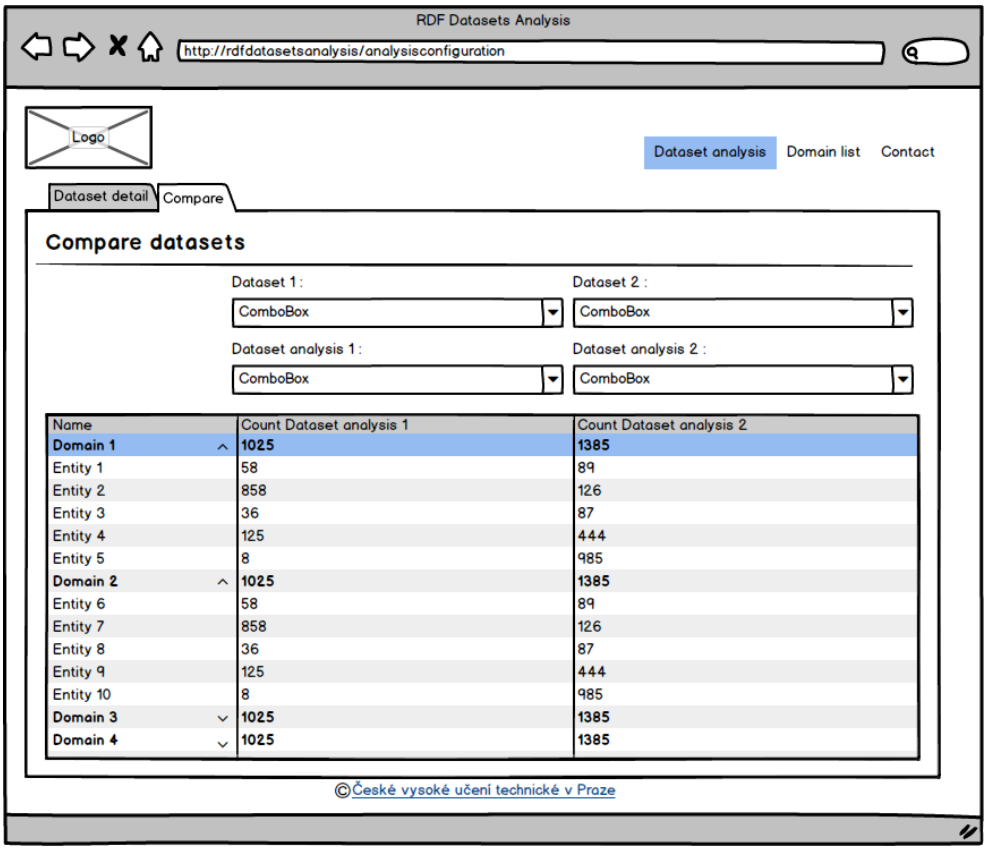


Figure C.13: Wireframe - DatasetAnalysis detail with opened compare page

Appendix D

Screenshots of the application

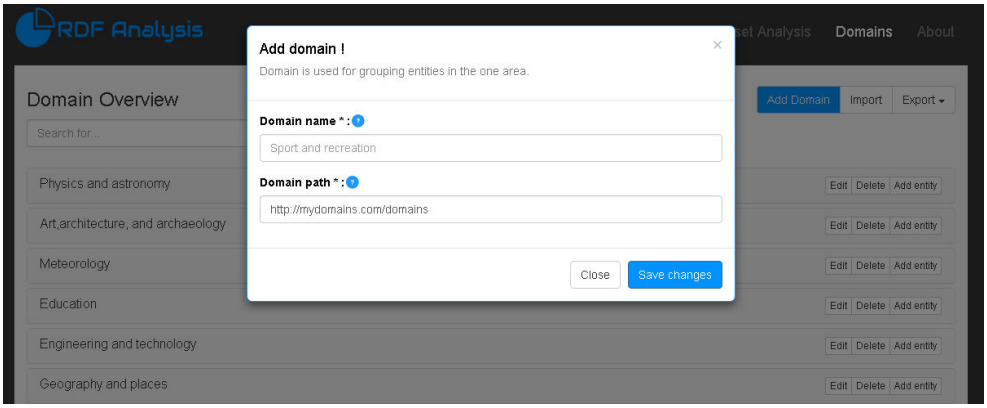


Figure D.1: Screenshot - Form for the creation of new domain

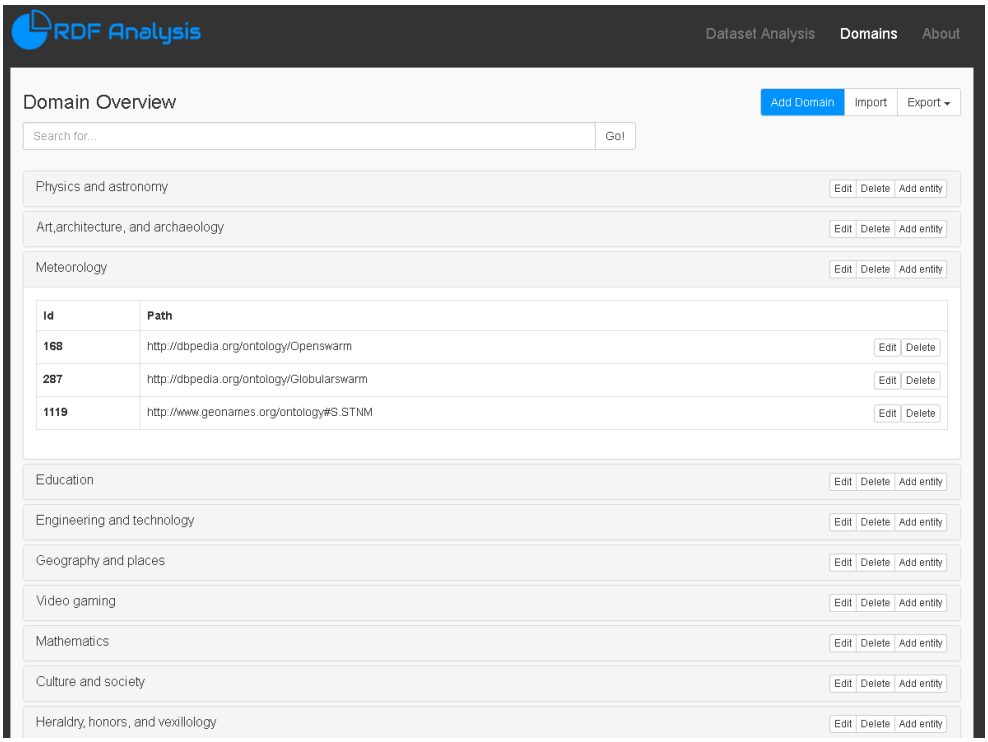


Figure D.2: Screenshot - Domain list with unpacked one domain

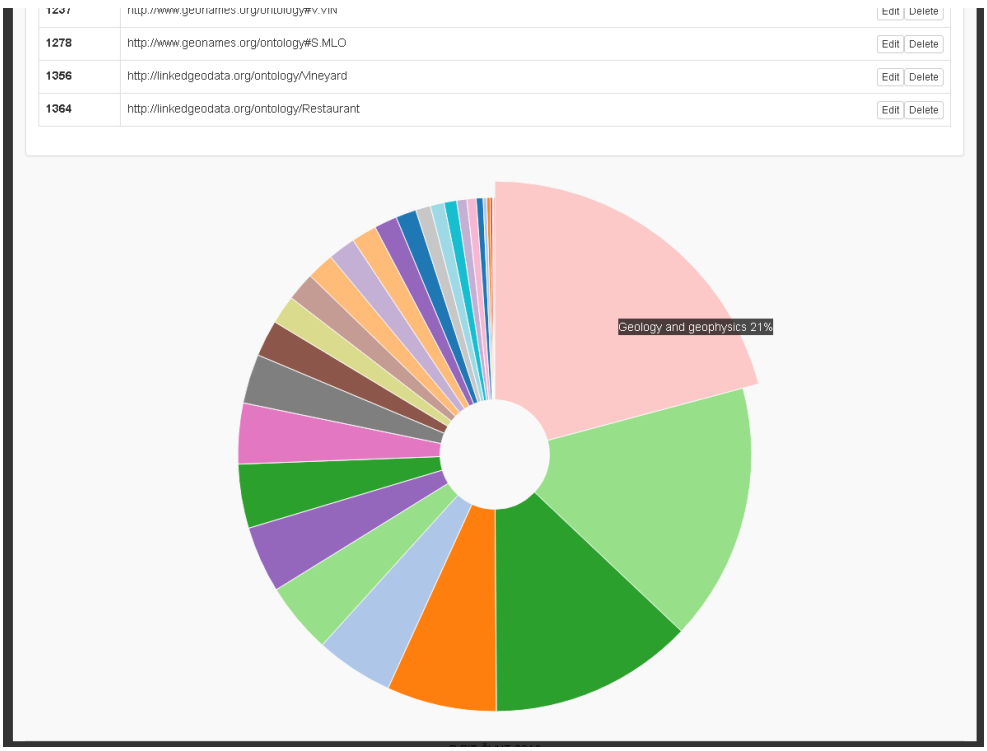


Figure D.3: Screenshot - End of the page Domains with chart shown

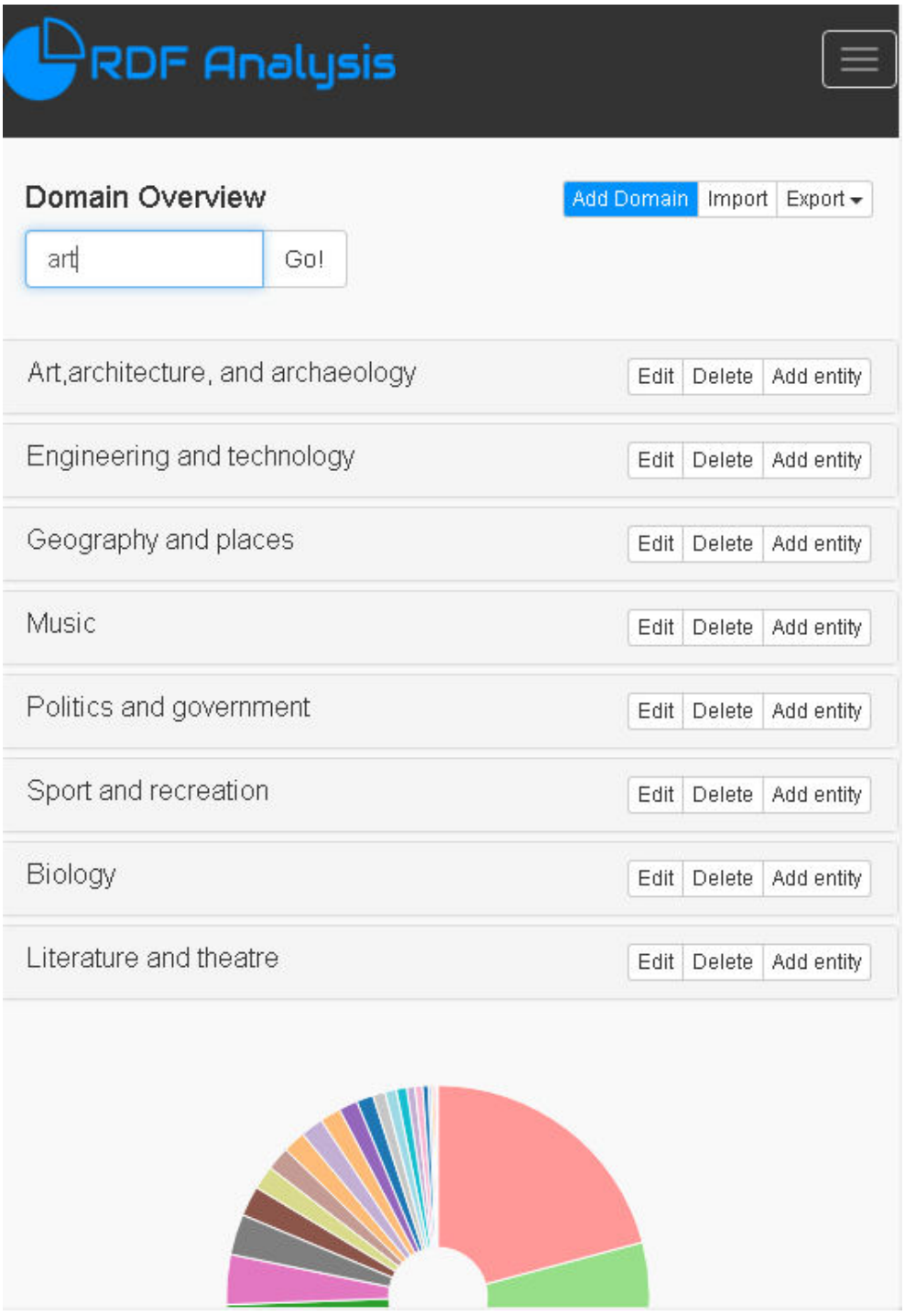


Figure D.4: Screenshot - Overview of domains in the mobile screen with the searching view

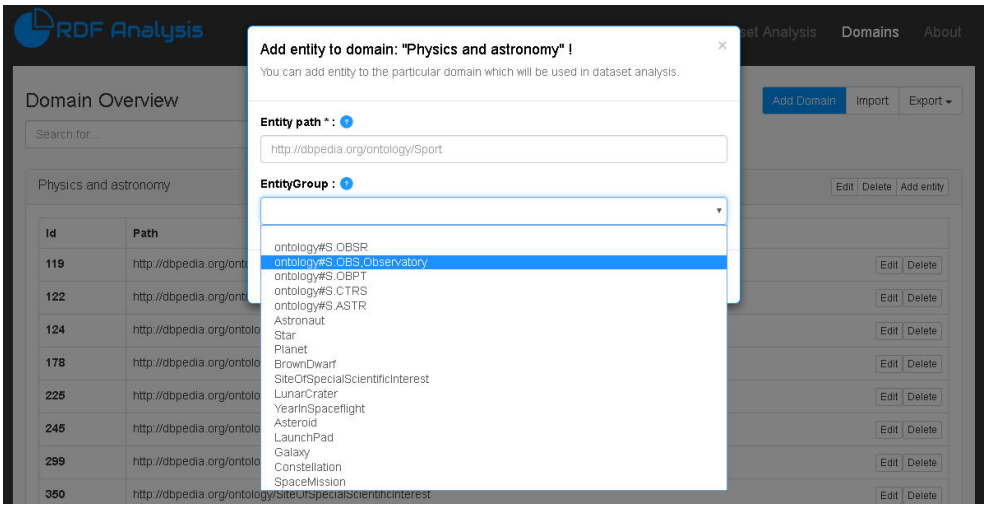


Figure D.5: Screenshot - Form for the creation of new entity

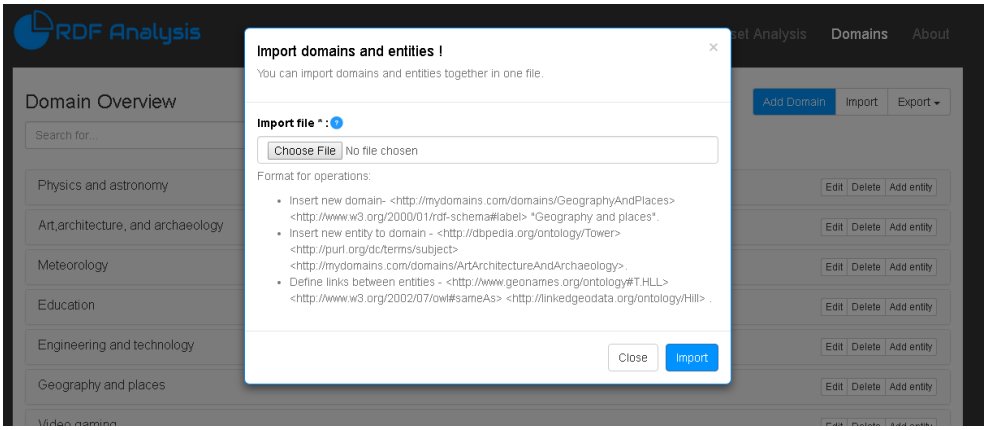


Figure D.6: Screenshot - Form for the import domains and entities

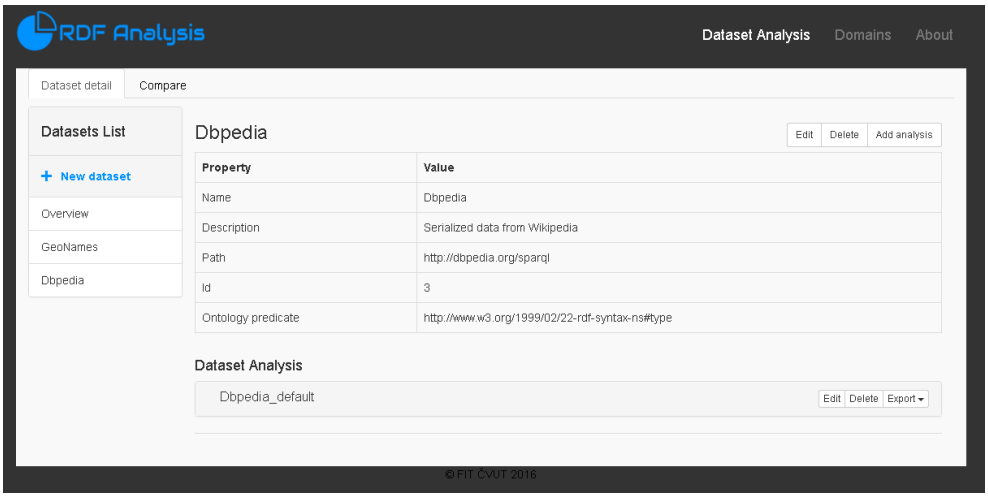


Figure D.7: Screenshot - Dataset detail with packed its analysis

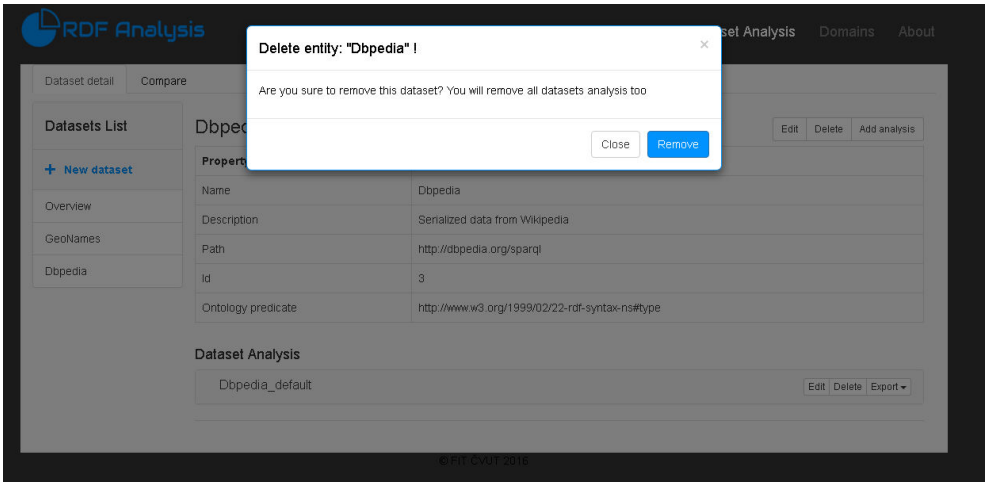


Figure D.8: Screenshot - Form for dataset deleting.

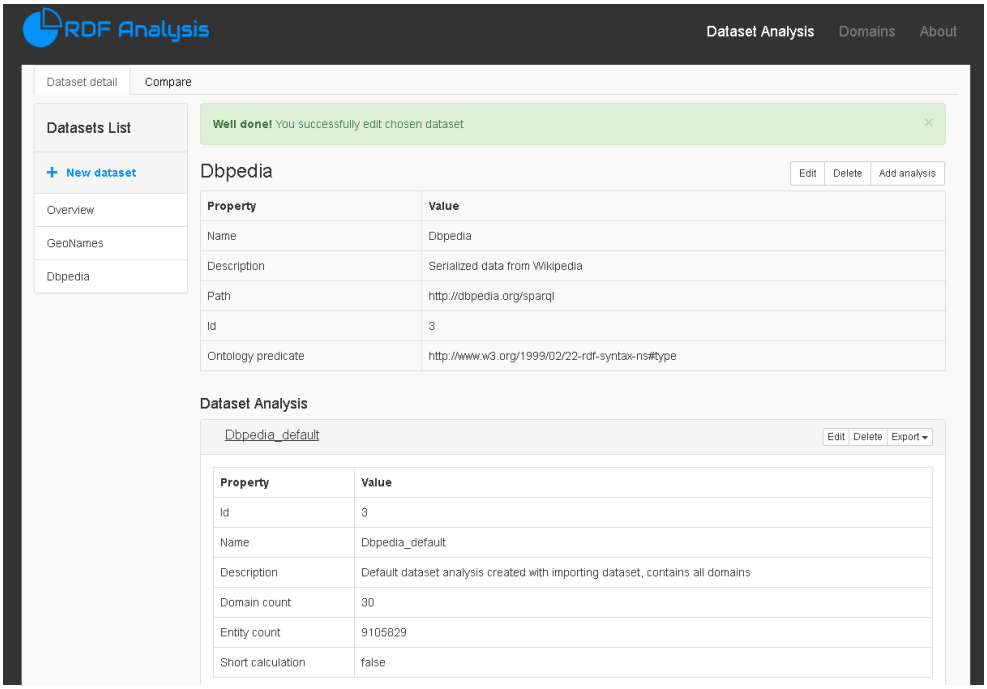


Figure D.9: Screenshot - Dataset detail with unpacked analysis - the first part of analysis

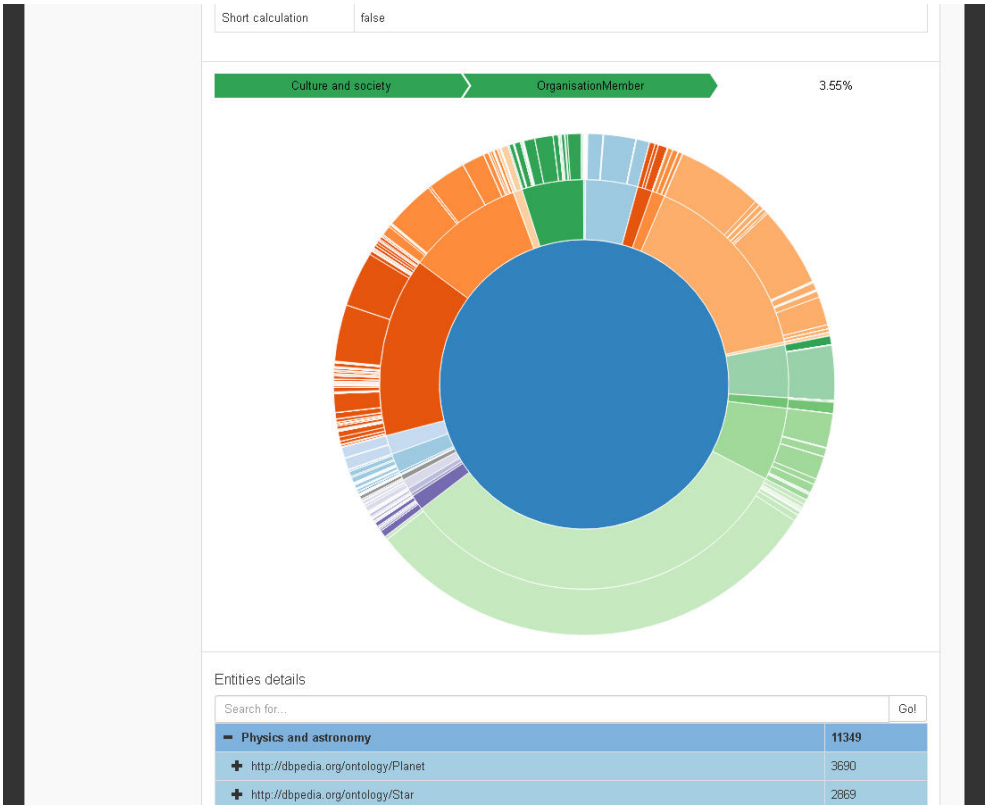


Figure D.10: Screenshot - Dataset detail with unpacked analysis - second and partly third part of analysis

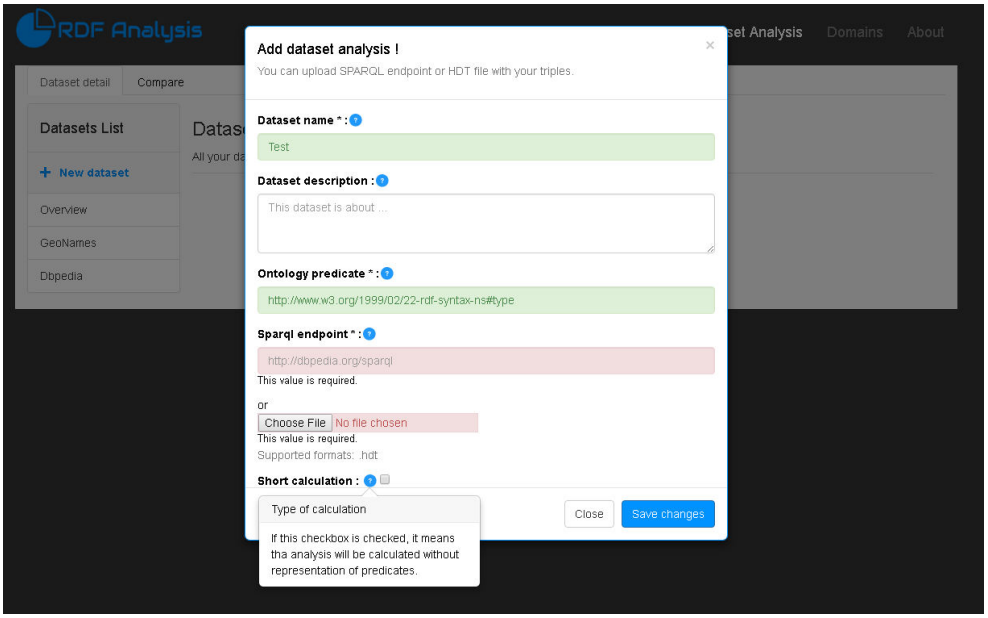


Figure D.11: Screenshot - Form for adding new dataset with the validation and displaying help message

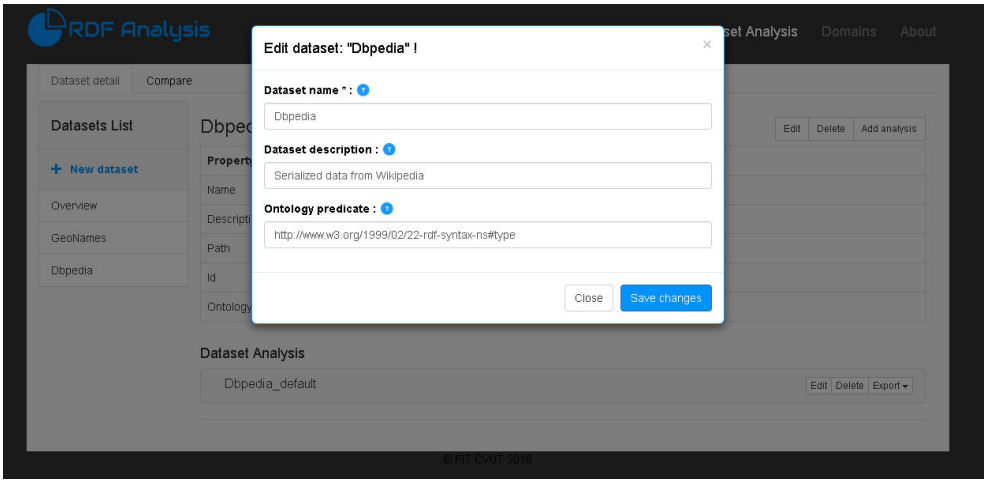


Figure D.12: Screenshot - Form for editing already existed dataset

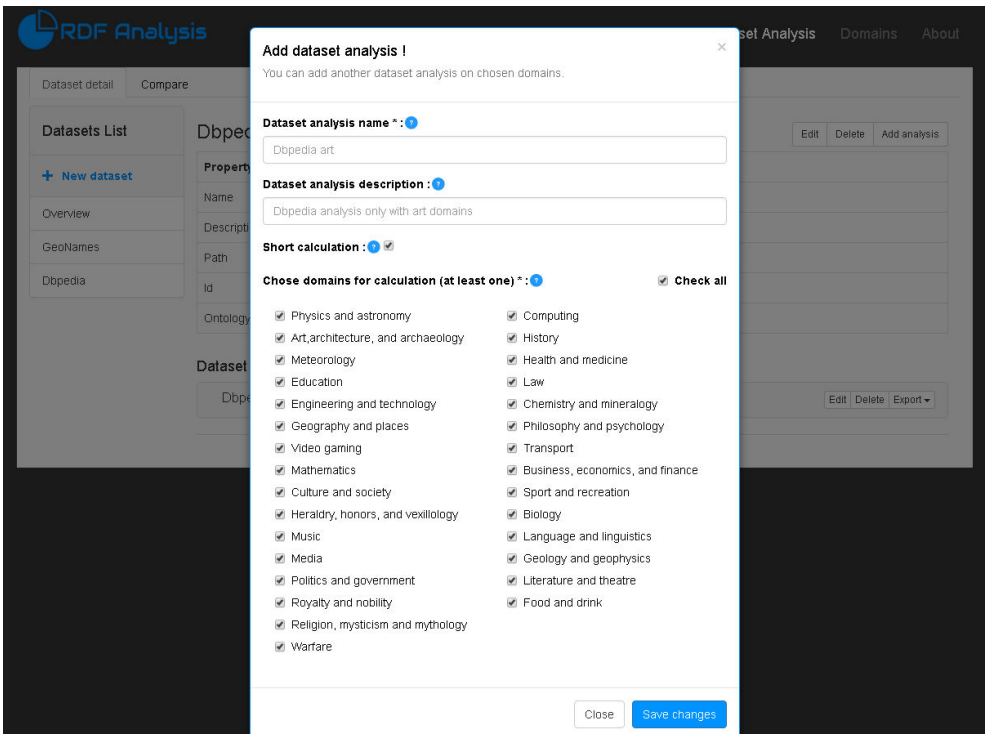


Figure D.13: Screenshot - Form for adding new dataset analysis

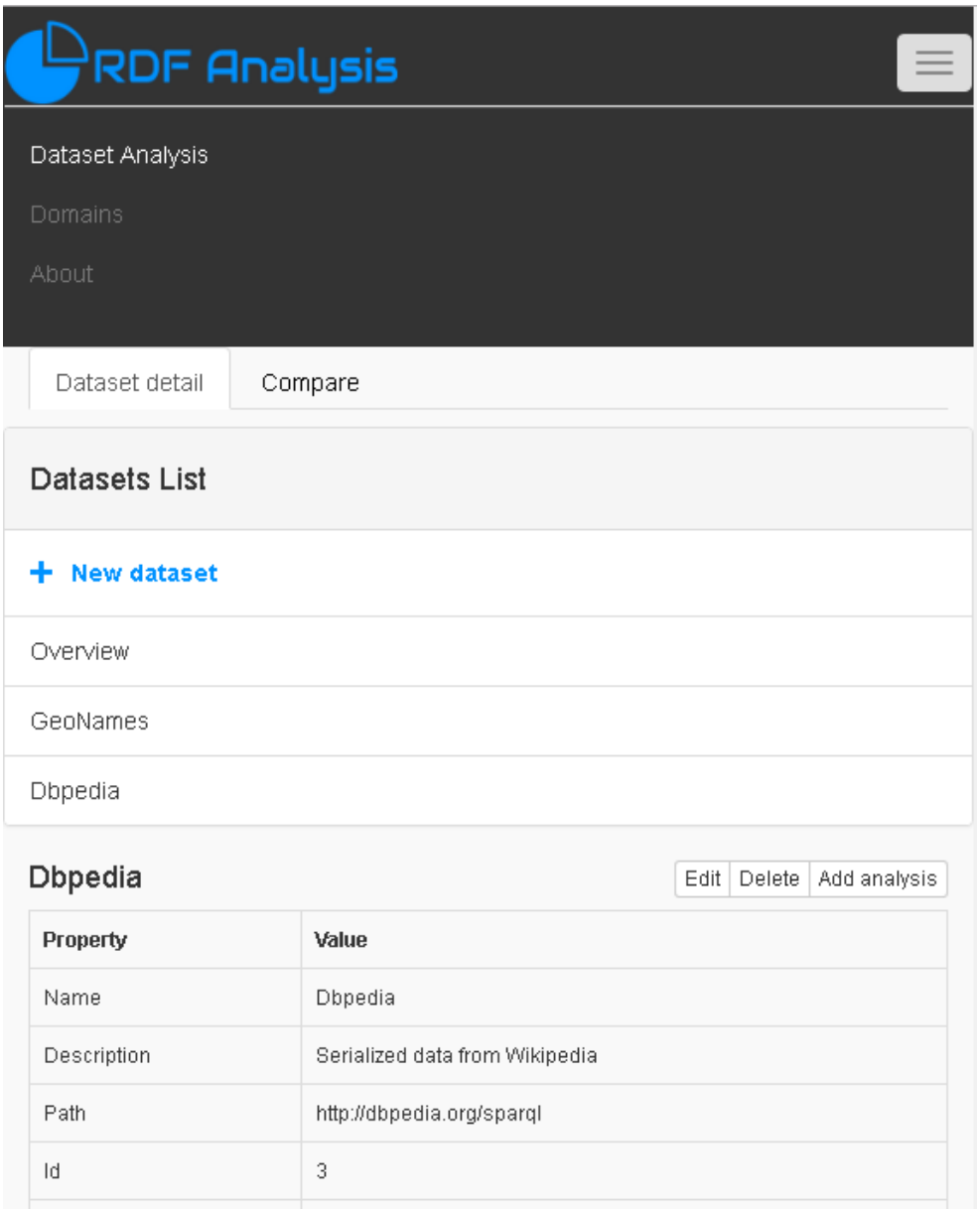


Figure D.14: Screenshot - View page ”Dataset analysis” in the mobile screen

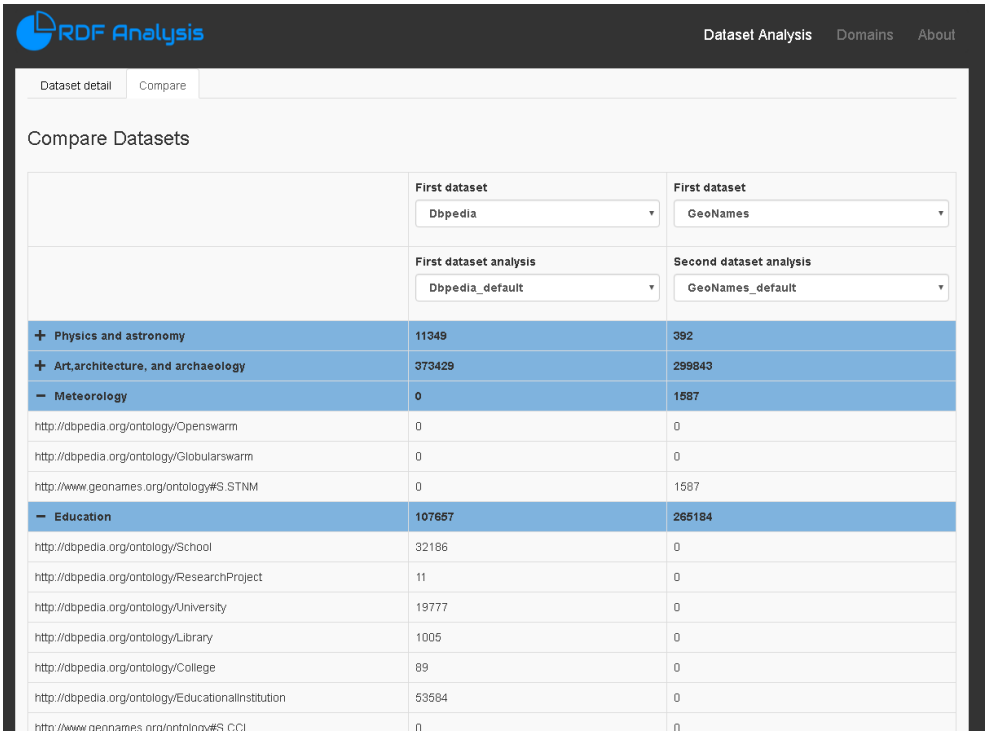


Figure D.15: Screenshot - Open the tab ”Compare” to compare two datasets

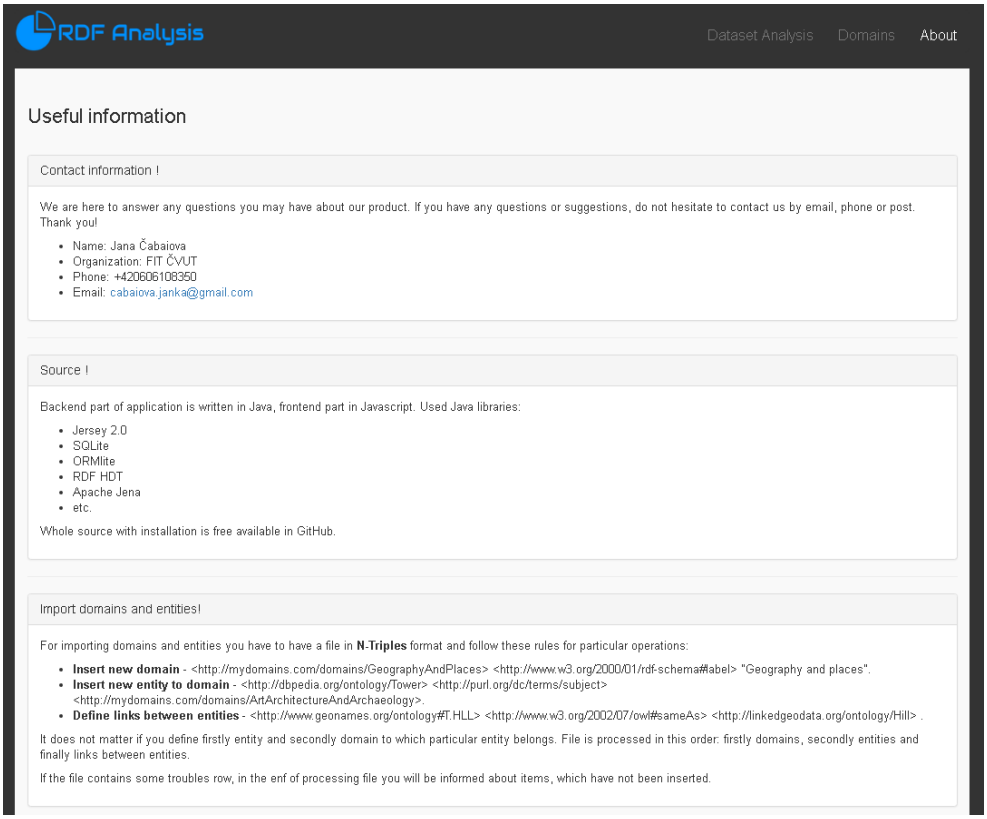


Figure D.16: Screenshot - Page with contact, source and useful information about application functions

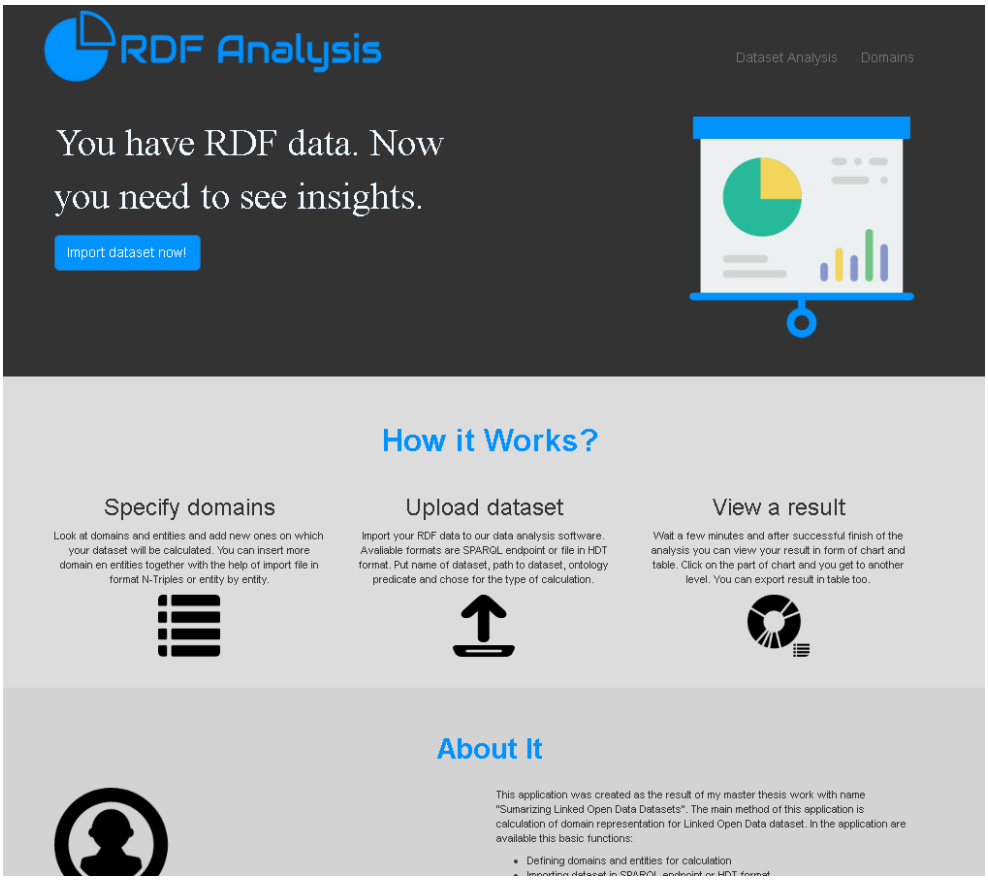


Figure D.17: Screenshot - Landing page of the application