Master's Thesis

# Configurable Schema-Aware RDF Data Input Forms

Dávid Konkoly

April 2017


Albert-Ludwigs Universität Freiburg
Department of Computer Science
Chair of Databases and Information Systems

**Candidate**
Dávid Konkoly
**Matr. number**
3757311
**Working period**
18. 10. 2016 – 18. 04. 2017
**Examiner**
Prof. Dr. Georg Lausen
**Supervisor**
Victor Anthony Arrascue Ayala

# Abstract

Abstract in English

# Kurzfassung

Kurzfassung auf Deutsch

# Contents

**Abstract** **II**

**Kurzfassung** **III**

**List of Tables** **VIII**

**1 Introduction** **1**

1.1 Initial goal and contributions . . . . . 1

1.2 Thesis outline . . . . . 1

**2 Preliminaries** **2**

2.1 Semantic Web . . . . . 2

2.1.1 RDF . . . . . 2

2.1.2 RDF Schema . . . . . 4

2.1.3 OWL . . . . . 5

2.1.4 SPARQL . . . . . 7

2.2 Applied Ontologies . . . . . 8

2.2.1 Foundational Model of Anatomy - *FMA* . . . . . 8

2.2.2 Ontology for Biomedical Investigations - *OBI* . . . . . 10

2.3 Web applications . . . . . 11

2.3.1 Client-sever architecture . . . . . 11

2.3.2 Data driven web applications . . . . . 12

2.3.3 Applications with RDF Data . . . . . 15

**3 Problem Statement** **19**
3.1 Challanges of the RDFBones project . . . . . 19
3.1.1 Human skeleton . . . . . 19
3.1.2 Study Design Execution . . . . . 20
3.1.3 Ontology Extensions . . . . . 21
3.2 RDF Data input . . . . . 23
3.2.1 Multi dimensional form . . . . . 23
3.2.2 Form dependencies . . . . . 24
3.2.3 Instance browsing . . . . . 24
3.2.4 Validation . . . . . 24
3.2.5 Editing form data . . . . . 25
3.2.6 Saving data . . . . . 26

**A Glossary** **28**

**B Appendix** **33**
B.1 Something you need in the appendix . . . . . 33

# List of Figures


2.1 Main structure of the RDFS vocabulary . . . . . 4
2.2 RDFS domain and range definition . . . . . 5
2.3 RDFS domain and range definition . . . . . 5
2.4 A subset of OWL vocabulary . . . . . 6
2.5 OWL object properties . . . . . 7
2.6 Properties for qualified cardinalities . . . . . 7
2.7 Ontology structure for skeleton . . . . . 9
2.8 Client server communication . . . . . 11
2.9 HTML document is interpreted by the browser . . . . . 11
2.10 Navigation through the web application . . . . . 13
2.11 Data flow . . . . . 13
2.12 Flow of information from DB to client . . . . . 14
2.13 SQL query with parameter . . . . . 14
2.14 Links to data items . . . . . 15
2.15 Form layout and HTML document . . . . . 15
2.16 Request with parameters . . . . . 15
2.17 Example Java routine for data storage . . . . . 16
2.18 Ontology and data . . . . . 16
2.19 VIVO Profile page . . . . . 17
2.20 Triples to store . . . . . 17

3.1 RDF Triple representation of a skull . . . . . 19
3.2 RDF Triple representation of a skull . . . . . 20
3.3 Glabella and its expressions . . . . . 20

3.4 Study Design Execution Data Structure . . . . . 21
3.5 Ontology extension for Glabella . . . . . 21
3.6 Skeletal Inventory Data Structure . . . . . 22
3.7 Ontology extension scheme . . . . . 22
3.8 Multi dimensional form layout . . . . . 23
3.9 Subform dependencies . . . . . 24
3.10 Navigator example . . . . . 24
3.11 Example data model . . . . . 25

# List of Tables

3.1 SPARQL Result . . . . . 26

Chapter 1

# Introduction

Introduction.
You can reference the only entry in the .bib file like this: [2]

## 1.1 Initial goal and contributions

## 1.2 Thesis outline

Chapter 2

# Preliminaries

## 2.1 Semantic Web

### 2.1.1 RDF

In RDF, abbreviation for Resource Description Framework, the information of the web is represented by means of triples. Each triple consists of a subject, predicate and object. The set of triples constitute to an RDF graph, where the subject and object of the triples are the nodes, the predicates are the edges of the graph. An RDF triple is called as well statement, which asserts that there is a relationship defined by the predicate, between subject and the object. The subjects and the objects are RDF resources. A resource can be either an IRI (Internationalized Resource Identifier) or a literal or a blank node (discussed later). A resource represents any physical or abstract entity, while literals hold data values like string, integer or datum. Basically there are two types of triples, the one that links two entities to each other, and the other that links a literal to an entity. The former expresses a relationship between two entities, and the latter in turn assign an attribute to the entity. Common practice is to represent IRI with the notation prefix:suffix, where the prefix represents the namespace, and the expression means the concatenation of the namespace denoted by the prefix, with the suffix. This convention makes the RDF document more readable. The namespace of RDF is the http://www.w3.org/1999/02/22-rdf-syntax-ns#, whose prefix is in most cases "rdf". This is defined on the following way:

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

Literals are strings consisting of two elements. The first is the lexical form, which is the actual value, and the second is the data type IRI. RDF uses the data types from XML schema. The prefix (commonly xsd) is the following :

@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

So a literal value in RDF looks as follows:

"Some literal value"^^xsd:string

The RDF vocabulary provides some built-in IRIs. The two most important are, the rdf:type property, and the rdf:Property class. The meaning of the triples, where the predicate is the property rdf:type is that the subject IRI is the instance of the class denoted by the object. Therefore the following statement holds in the RDF vocabulary:

rdf:type rdf:type rdf:Property.

It is maybe confusing that an IRI appears in a triple as subject and predicate as well, but we will see by the RDFS vocabulary that it is inevitable to express rules of the language. To be able to represent information about a certain domain, it is necessary to extend the RDF vocabulary with properties and classes. The classes will be discussed in the next section, but here it is explained how custom properties can be defined. The namespace of the example is the following:

@prefix eg: <http://example.org#>.

The example dataset intends to express information about people, which university they attend and how old are them. To achieve this two properties are needed:

eq:attends rdf:type rdf:Property .
eq:age rdf:type rdf:Property .

The actual data about a person:

eg:JanKlein eq:attends eq:UniversityOfFreiburg .
eg:JanKlein eq:age "21"^^xsd:integer .

### 2.1.2 RDF Schema

The previous section gave an insight into RDF world by showing how can information stored by means of triples. However the explanation did not mention that each RDF dataset has to have scheme, which is also called ontology. The ontology describes the set of properties and classes and how are they are related to each other. RDFS provides a mechanism to define such ontologies using RDF triples. The most important elements of the RDFS vocabulary can be seen on the following image.

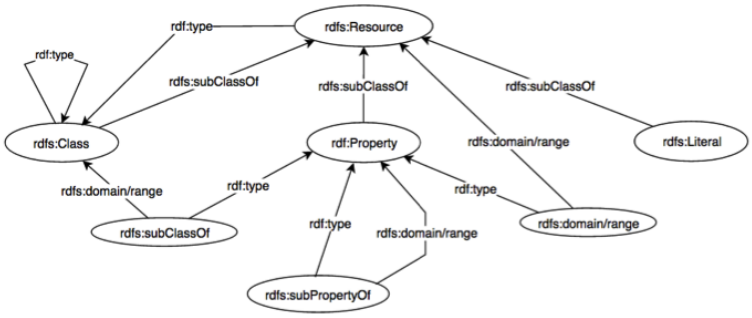


Figure 2.1: Main structure of the RDFS vocabulary

The two most important classes in the RDFS vocabulary is the rdfs:Class and the rdfs:Resource. The rdfs:Class is class, because it is the instance of itself, and the same way the rdfs:Resource is a class. The rdf:Propery and the rdfs:Literal are both classes as well. The rdfs:domain, rdfs:range, rdfs:subPropertyOf and rdfs:subClassOf are properties. Important to note that these properties are subjects and predicates in the same time in the RDFS vocabulary graph. Also they describe themselves like rdf:type. The properties rdfs:domain and rdfs:range describe for the property the type of the subject and object respectively, which with it can build a triple as predicate. The following image illustrates their meaning:

Since both the rdfs:domain and rdfs:range are properties themselves, they have as well their domain and range, which is the class rdfs:Resource. The property rdfs:subClassOf expresses subclass relationship between classes. It means if a class B is a subclass of class A, and resource R is the instance of class B, then resource R is the instance of class A as well. Since it describes the relationship between two classes its both domain and range is

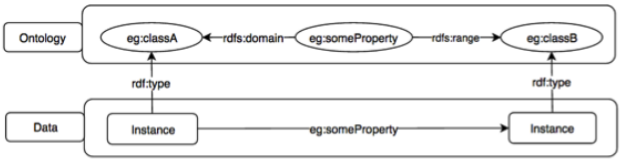


Figure 2.2: RDFS domain and range definition

the class rdfs:Class. The property rdf:subPropertyOf expresses the relationship between two properties. If property P2 is sub property of P1 and two instances are related by P2 then they are related by P1 as well. Its domain and range is the class rdf:Property. Now everything is given to define the ontology for the example of the previous section.

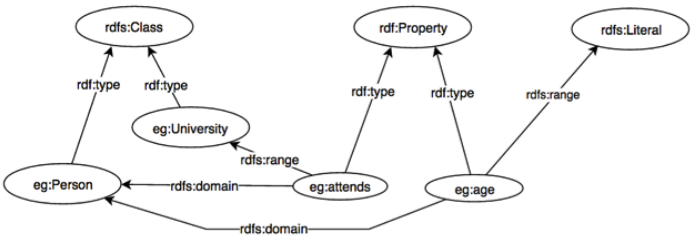


Figure 2.3: RDFS domain and range definition

### 2.1.3 OWL

OWL, abbreviation for Ontology Web Language is an extension of the RDFS vocabulary. OWL allows expressing additional constraints on the data, above the range and domain definitions. These constraints are called restrictions. Restrictions are conventionally expressed by blank nodes. Blank nodes do not have IRIs, but it is defined through the triples in which they participate as a subject. For example a restriction stating that the instances of the class eg:FootballTeam can build a triple through the eg:hasPlayer property only with the instances of eg:FootballPlayer class can be expressed the following way:

```
eg:FootballTeam rdfs:subClassOf [
  rdf:type owl:Restriction ;
  owl:onProperty eg:hasPlayer ;
```

```
  owl:allValuesFrom eg:FootballPlayer .
]
```

Listing 2.1: OWL restriction in N3 format

owl:Restriction is class and owl:onProperty and owl:allValuesFrom are properties. It can be seen that class, on which the restriction applies is the subclass of the restriction blank node. Furthermore OWL is capable of expressing qualified cardinality restriction. For example the statement that a basketball team has to have exactly five players, look as follows in OWL:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX eg: <http://example.org>

eg:BasketballTeam rdfs:subClassOf [
  rdf:type owl:Restriction ;
  owl:onProperty eg:hasPlayer ;
  owl:onClass eg:Player ;
  owl:qualifiedCardinality "5"^^xsd:nonnegativeInteger
] .
```

Listing 2.2: OWL restriction in N3 format

These two examples cover the thesis related features of OWL. The next image depicts the OWL vocabulary.

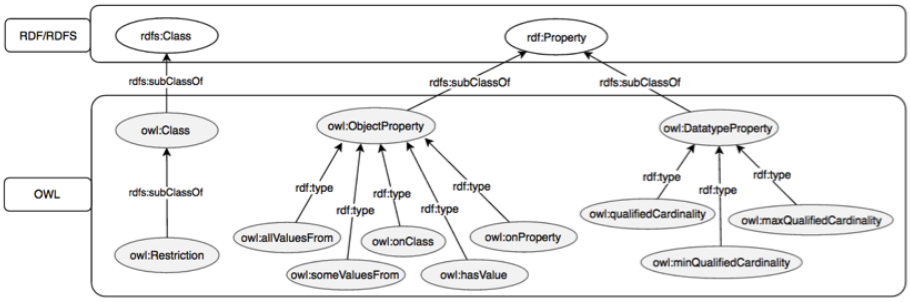


Figure 2.4: A subset of OWL vocabulary

There are two new class types are the owl:Class and the owl:Restriction. The rdf:Property has two subclasses, the owl:ObjectProperty and owl:DataTypeProperty. owl:ObjectProperty represent the properties that links instances to instances, and the owl:DataTypeProperty is those that link instances to literals. The

following two images shows the domain and range definitions of the OWL properties used to describe restrictions.

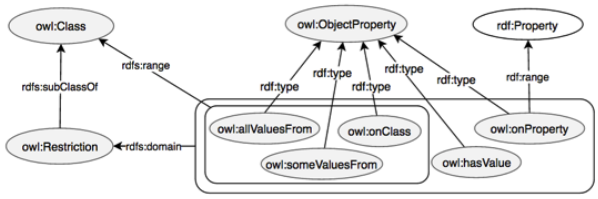


Figure 2.5: OWL object properties

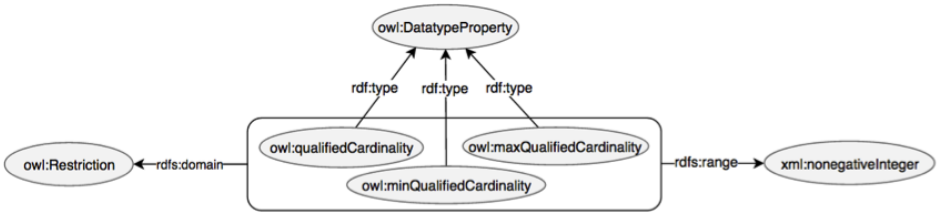


Figure 2.6: Properties for qualified cardinalities

### 2.1.4 SPARQL

SPARQL is a query language for querying data in RDF graphs. A SPARQL query is a definition of a graph pattern through variables and constants. The following example query returns all IRIs that represent a football player:

```
SELECT ?player
WHERE {
  ?player   rdf:type   eg:FootballPlayer .
}
```

Listing 2.3: SPARQL Query I.

In the example the query consist of only one triple. The subject is a variable and the predicate and the object are constant. Therefore the triple store looks all the triples and checks the predicate is rdf:type and the object is eg:FootballPlayer. It is well possible to not just ask the IRI of the players but further information by adding additional triples to the query in order to ask the name for example of the player:

```
SELECT ?player ?name
WHERE {
  ?player   rdf:type   eg:FootballPlayer .
  ?player   eg:name    ?name .
}
```

Listing 2.4: SPARQL Query II.

The result table in this case will contain two columns, one with the IRI of the person and one with their name. Important that it is as well possible to query blank nodes by introducing a variable for it. So if we want to list all the instances that are coming into question as player to a football team we can formulate the following query:

```
SELECT ?person ?name
WHERE {
  eg:FootballTeam rdfs:subClassOf   ?restriction .
  ?restriction    rdf:type          owl:Restriction .
  ?restriction    owl:onProperty    eg:hasPlayer .
  ?restriction    owl:allValuesFrom ?playerType .
  ?player         rdf:type          ?playerType .
  ?player         eg:name           ?name .
}
```

Listing 2.5: SPARQL Query III.

## 2.2 Applied Ontologies

Ontologies are used to describe types, relationships and properties of objects of a certain domain. It is a common practice to used already defined ontologies rather developing an own. The first reason is, that the development of an ontology is a complex and a tedious process, and requires a lot of resource. Secondly, it is reasonable to use standardized vocabularies, in order to make data from same domain but different sources inter-operable.

### 2.2.1 Foundational Model of Anatomy - *FMA*

The foundational Model of Anatomy ontology is an open source ontology written in OWL. FMA is a fundamental knowledge source for all biomedical domains, and it provides a declarative definition of concepts and relationships

of the human body for knowledge based applications. It contains more than 70 000 classes, and 168 different relationships, and organize its entities into a deep subclass tree [4]. All types of anatomical entities are represented in FMA, like molecules, cells, tissues, muscles and of course bones. In our project we use only the subset of the FMA. The taken elements are the subclasses of the following two classes and the three properties:

- Classes

  *Subdivision of skeletal system* - fma:85544

  *Bone Organ* – fma:5018

- Properties

  *fma:systemic_part_of*

  *fma:constitutional_part_of*

  *fma:regional_part_of*

The class *Bone Organ* is the superclass of all bones in the human skeleton. Each bone belong to a skeletal subdivision and a skeletal subdivision can be a part of another skeletal subdivision. This relationship in both cases is expressed by the property *fma:systemic_part_of*. To define which bone organ belongs to which skeletal subdivision FMA uses OWL restrictions (see Figure 2.7). The properties *fma:constitutional_part_of* and *fma:regional_part_of* *fma:constitutional_part_of* are discussed later.

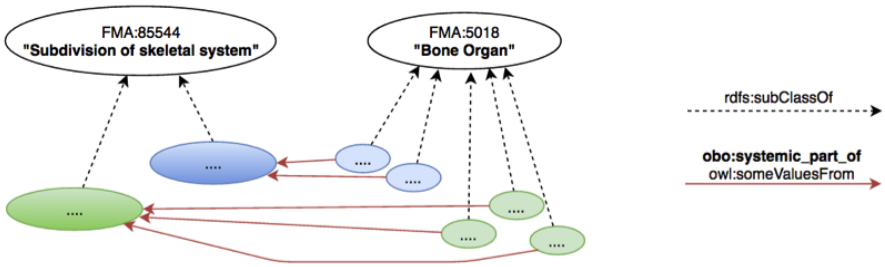


Figure 2.7: Ontology structure for skeleton

Finally the advantage of using the FMA ontology is that, if in the future further elements of the human body have to be addressed by the research

processes, i.e. muscles, then these classes can be easily integrated to the currently applied subset.

### 2.2.2 Ontology for Biomedical Investigations - *OBI*

The aim of OBI ontology, is to provide the formal representation of the biomedical investigation in order to standardize the processes among different research communities. It is a result of a collaborative effort of several working groups, and it continuously evolving as new research methods are being developed. Its main function to describe the rules how biological and medical investigations have to be performed. OBI reuses terms from BFO *Basic Formal Ontology* IAO *Information Artifact Ontology* and OBO *Open Biological and Biomedical Ontologies*[3]. To define processes OBI uses the following three general classes:

- *Information Content Entity* - obo:IAO_0000030
- *Material Entity* - obo:BFO_0000040
- *Process* - obo:BFO_0000015

Information Content Entity represent results of a specific measurement, while Material Entity stands for the objects, on which the meausurements have been performed. The Process could mean any kind of step within an investigation, from the planning, through execution till the conclusion.

- *Planning* - obo:OBI_0000339
- *Study Design Execution* - obo:OBI_0000471
- *Drawing a conclusion* - obo:OBI_0000338

In our project the following three properties are used:

- *has part* - obo:BFO_00000051
- *has specified input* - obo:OBI_00000293
- *has specified output* - obo:OBI_00000299

## 2.3 Web applications

This chapter contains practical information about how web applications work. In section 2.3.1 the basic mechanism of data driven applications are discussed, like navigation between page, data display and creation. Section 2.3.3 then focuses on the applications that are using semantic technologies, and addresses what kind of architectural changes that means.

### 2.3.1 Client-sever architecture

A web application is program that runs on a machine, which is accessible through the web. The machine is called server, because its main purpose is to server request that are coming from the web browser. Web browsers are as well programs, but they run on personal computers, tablets, etc, and they are capable of sending request through web to the servers. The response to these requests are HTML document, which can be displayed by the browser.

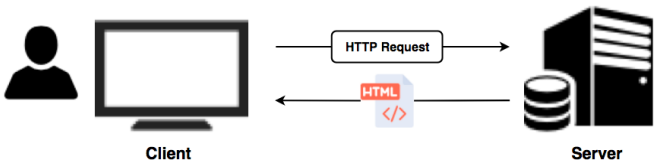


Figure 2.8: Client server communication

An HTML document contains definition of the elements of the pages, such as tables, buttons, etc. It contains as well so called CSS docuements (Cascading Style Sheet), which is responsible for the definition of the style of the elements. Moreover to make the web pages more interactive, JavaScript (JS) de can be embedded to HTML as well.

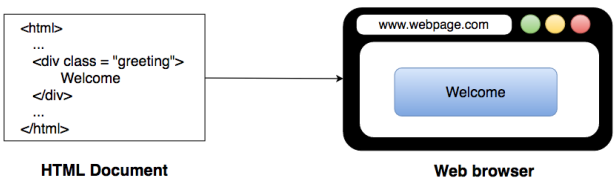


Figure 2.9: HTML document is interpreted by the browser

Initially web pages were static, which means that their only function was

to show certain set of information. These applications usually web applications do not consist of one single page, but of several different pages. Like a web page for news, have normally a main page, and different sub pages for the particular topics. In order to navigate between the pages of the application, the HTML document contains links that trigger further HTTP requests. Links in HTML can be defined by means of the <a/> tag. The most important parameter of this tag is href, whose value contains the URL of the HTTP request. Let assume that an application's main page is accessible through the URL http://newsPortal.com. Common practice that subpages of the application can be called through various url-mappings, which means the main URL is extended with a keyword that denotes the page to be requested.

```
<a href="http://newsPortal.com/politics"> Politics </a>
<a href="http://newsPortal.com/sport"> Sport </a>
```

Listing 2.6: Example link definitions

If the user clicks on of these link (with the label 'Politics' and 'Sport') then these request will be sent to the news portal page. Each such request has to be served, differently to each mapping some routine has to be assigned. For example by Java web applications, the classes of the server that process the request are called servlets. On the next image it is shown, how the XML file defines, which class is responsible for the the mapping '/politics'.

```
<servlet-mapping>
  <url-pattern>/politics</url-pattern>
  <servlet-class>servlets.PoliticsController</servlet-class>
</servlet-mapping>
```

Listing 2.7: Java servlet mapping definition

Then the responsibility of the class *servlets.PoliticsController* is to respond the corresponding HTML page for the client. Figure 2.10 show the main structure of the applications, where the rectangles on the client side represent the different pages of the application.

### 2.3.2 Data driven web applications

This section aims to present the fundamentals of the web technologies that allows to build application for browsing and creating data. Modern web ap-

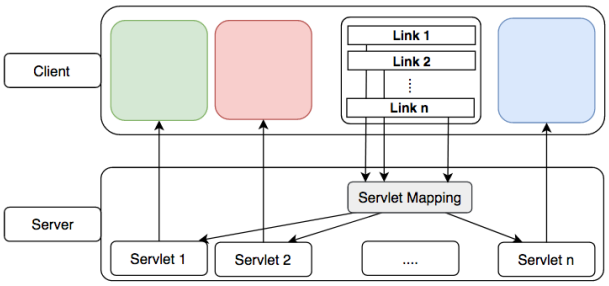


Figure 2.10: Navigation through the web application

plications do not store the information in HTML documents. So the page loading process is not just the sending the HTML document, but a retrieval of a particular dataset, and the substitution into a web page. First of all the task of responding requires a query that retrieves that data from the database. By applications using relational data model, the tables and attributes are always modeled by classes of the used object oriented programming (OOP) language. So the data retrieval is the instantiation of the classes in scope.

Let assume that articles of a news portal is stored in a table with the attributes, id, type, title, summary and text. Then there has to be a class defined in the server code with the same attributes. To instantiate instances of the class, it is necessary to perform an SQL query.

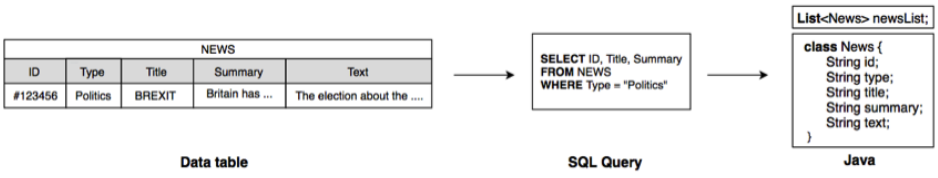


Figure 2.11: Data flow

The query results not only one instance of the News class, but a list (*List<News> newList*). To generate from this a HTML page that shows the articles, normally so called template engines are used. Templateing enables to define the HTML documents parametric, and passing them data, and they generate the result page automatically.

```
<#list  newsList as news>
```

```
  <h3>  ${news.title} </h3>
  <p>  ${news.summary} </p>
  <a  href = "http://newsPortal.com/wholeNews?id=${news.id}">
    Read more
  </a>
</#list>
```

Listing 2.8: Template file example

The template file is a description of how the data has to be converted into HTML document. It can be seen that it is possible for instance to declare a list on the input variable newsList. Then the template engine iterates through the News objects and by accessing its fields (title, summary, id) and generates the HTML for each element. So the complete flow of data from the database to the client looks as follows:

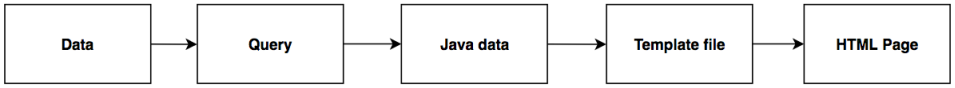


Figure 2.12: Flow of information from DB to client

The template shows only the summary of the article, but offers the following link:

```
http://newsPortal.com/wholeNews?id=${news.id}
```

The new feature is that after the url mapping there is a parameter *id*, and its value will be the database id of the web application. The idea is that this link redirects to the page where the whole article can be seen. So there has to be a servlet class defined to the mapping /wholeNews, which to perform the following query where the id is the input.

```
SELECT Text
FROM NEWS
WHERE ID = ${id}
```

Figure 2.13: SQL query with parameter

Thus it is achieved that different links are programmed to get access not only to different other pages, but to specific data items.

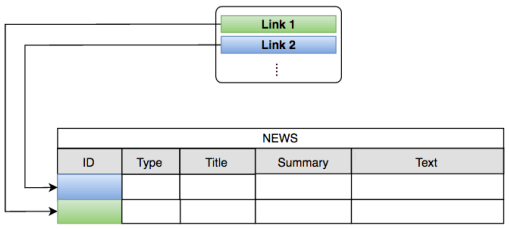


Figure 2.14: Links to data items

Web applications do not only just display existing data, but they allow the users to enter their new data. In HTML the element used for data input is called form. Form is a container, and it consists of particular form elements according to the data to be added.

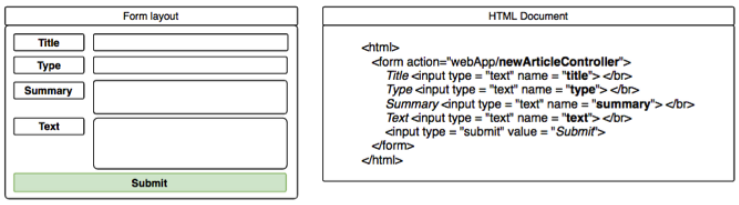


Figure 2.15: Form layout and HTML document

Submitting the form to the server send an HTTP request with multiple parameters, where they are divided through the & character.

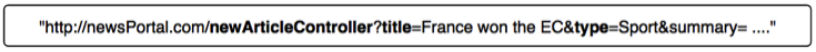


Figure 2.16: Request with parameters

By the data entry creation the task of the controller is to get the values from the request an instantiate the class representing the data to be created. Then initialized class instance is passed to the database where the entered data will be persistently stored.

### 2.3.3 Applications with RDF Data

This section aims to give an insight to web application that are based on RDF data. It will be covered what kind of requirements do the software have on the server side to create RDF data, and what is the difference between the

```
String title = request.getParameter("title");
....
News news = new News(title, type, summary, text);
DatabaseConnector.insert(news);
```

Figure 2.17: Example Java routine for data storage

RDF model based applications and the relational ones. In the last section by the input input processes, a class (namely the people) could have been, which were instantiated. This means that in relational databases the each class of scheme is represented by a class of the programming language. But as we have seen in section 2.2 the vocabularies can contain thousands of class, and thus it is not an option the translate them into the class of server application program. Therefore the semantic web based applications have a little bit different structure. First of all the most important is to see the main elements of an RDF dataset that can appear on the definition in the classes.

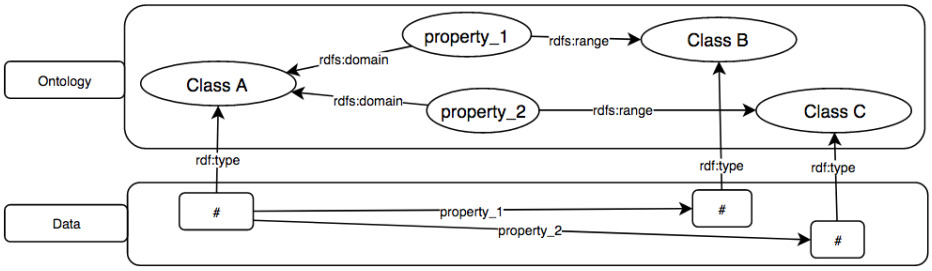


Figure 2.18: Ontology and data

```
SELECT ?property ?relatedInstance
WHERE {
  ?instance     rdf:type      :ClassA .
  ?property     rdfs:domain   :ClassA .
  ?instance     ?property     ?relatedInstance .
}
```

Listing 2.9: Dynamic SPARQL query

```
<#list properties as property>
  <#list property.dataSet as instance>
  <#list>
</#list>
```

Listing 2.10: Ontology adaptive template file

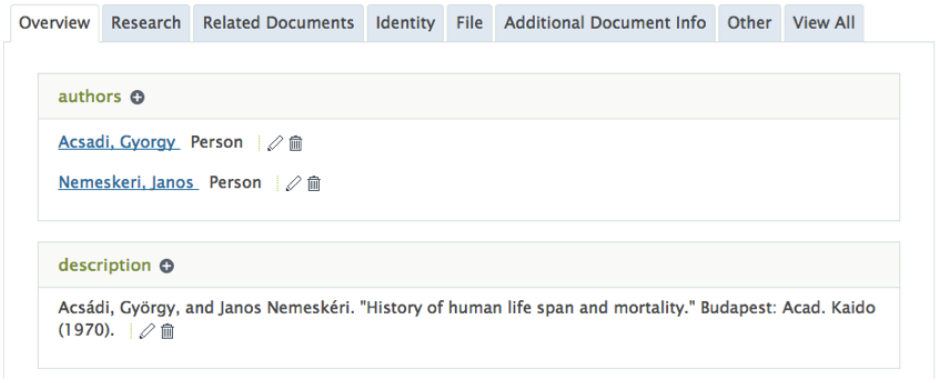


Figure 2.19: VIVO Profile page

Above the display of the existing data, RDF based applications are different as well in the data input mechanism. First of all, due to the fact that ontologies can contain thousand of class it is not an option to represent them all as classes of the server application language as well. It is time consuming and the system would loose its flexibility in the cases when new ontology subsets are supposed to be loaded. Therefore there are neither for each type of the database an entry form with a unique controller servlet, but more generic approaches are used. To define what dataset has to be created, semantic web based applications simply define them as a set of triples, with variables like in SPARQL, just the data flows the other way around.

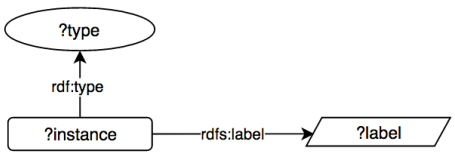


Figure 2.20: Triples representing a new instance

Figure 2.20 simple set of triples that have to be created by new data entry generation. The value of the variable textit?instance will be IRI that have not been used. This is an essential part of every triple store implementation that they provide unused uris for the server application for the new set of

triples. The values of the *type* and label, are coming from the input form. Here the

```
SELECT ?class ?label
WHERE {
  ?class       rdf:subClassOf*    obo:OBI_0000339 .
  ?class       rdf:label      ?label .
}
```

Listing 2.11: SPARQL query for the input form

will be a new unused IRI of the triple store, while the ?type variable come from the client. So let assume an entry form which displays the

# Chapter 3

# Problem Statement

## 3.1 Challanges of the RDFBones project

### 3.1.1 Human skeleton

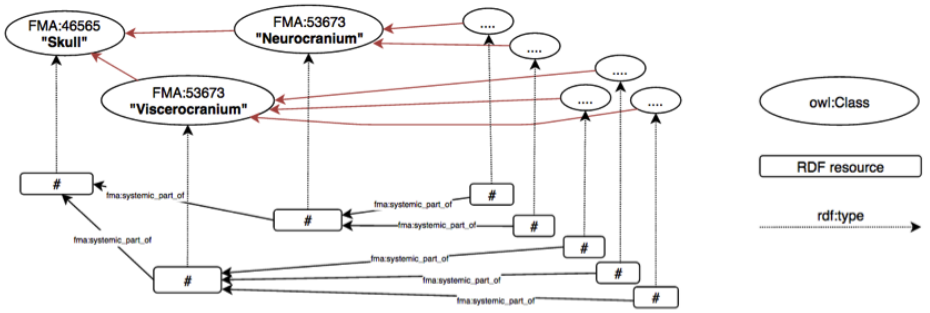


Figure 3.1: RDF Triple representation of a skull

To implement an entry form that allows the user to create such triple set takes considerably more effort then the cases explained in the previous sections, because not only key value pairs has to be sent from the client to the server, but a multi dimensional dataset. Like this:

```
{
  skeletalSubdivisionUri : "FMA:46565",
  sytemic_parts : [{
    uri : "FMA:53672",
    systemic_parts : [{
      uri : "FMA:52788",  //Right parietal bone
```

```
    } , { ... }]
  } , { ... }
  ]
}
```

Listing 3.1: Form data representing skull

Here the really important point the not only values are added like the simple form of the previous examples, but some elements can be added or just or just leaved. This leads to more dynamic interfaces.

### 3.1.2 Study Design Execution

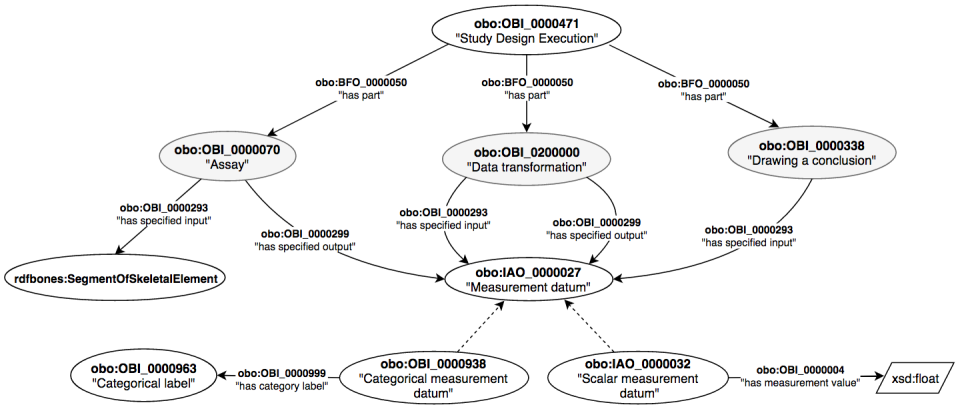


Figure 3.2: RDF Triple representation of a skull

In most investigations the researcher take a set of bones belong to one individual and examine different tokens. Tokens refer to specific features of parts or regions of bones. These token have particular expressions.

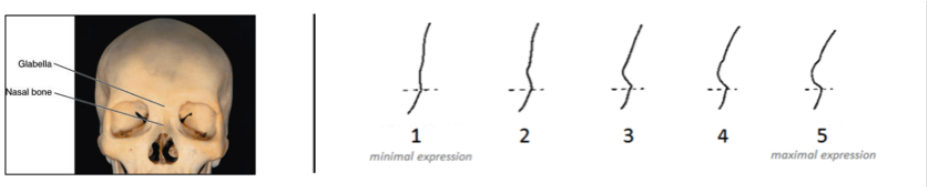


Figure 3.3: Glabella and its expressions

The previous images show the token called glabella, and its expressions. The task of the web application is let the researcher select one of the already

added Nasal Bones (because on that bone is the glabella token), and set the expression of it. The following data structure models the process.

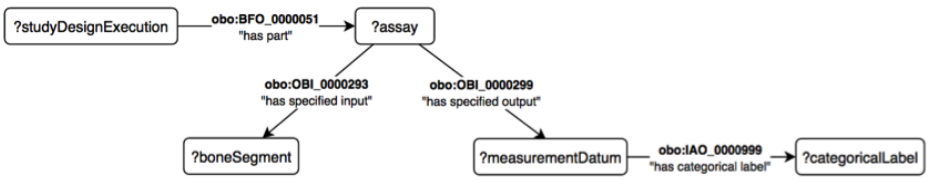


Figure 3.4: Study Design Execution Data Structure

Where the variable ?categoricalLabel represents the expression of the token. The values this variable can take are defined in the ontology extension. The variable ?boneSegment is the bone on which the glabella can be found. This instance won't be as well newly created, but an already added bone has to be selected on the interface. The ?assay and the ?meausurementDatum variables are new instances. To be able to generate an entry form for the problem, the following ontology extension has to be defined.

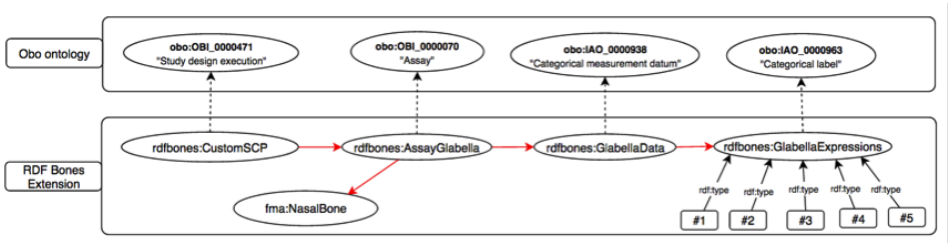


Figure 3.5: Ontology extension for Glabella

Mention that the task of the user is to add some elements if they exists or not.

### 3.1.3 Ontology Extensions

It is often the case that in an investigation not only the bone itself, but also particular segments has to be addressed. However the bone segments of the bones are not standard, and they can differ according to researcher or research project. Therefore FMA do not contain any bone segment of the bone organs, and consequently we have to define it on our own. Important that the skeletal subdivision instances do not appear on the dataset on their

own, but they are connected to Skeletal Inventories. Skeletal inventories are used to gather information about particular skeletal remains. The following image show the triple structure of skeletal inventories.


Figure 3.6: Skeletal Inventory Data Structure

The variable ?skeletalInventory is the instance of the class rdfbones:SkeletalInventory, while the ?boneSegment is from the class rdfbones:SegmentOfSkeletalElement.
The core ontology of the project contains a subclass of the rdfbones:SkeletalInventory, the rdf:PrimarySkeletalInventory. This skeletal inventory type is for skeletal remain collections where only the whole bone organs have to be addressed. The way to define custom bone segments is always through a custom skeletal inventories, which contains restrictions on the property obo:isAbout and on the class of custom bone segments. Of course the custom bone segments has to be assigned to the bone organ class they belong to, via restrictions on property obo:systemic_part_of. The following image illustrates the extension definition.

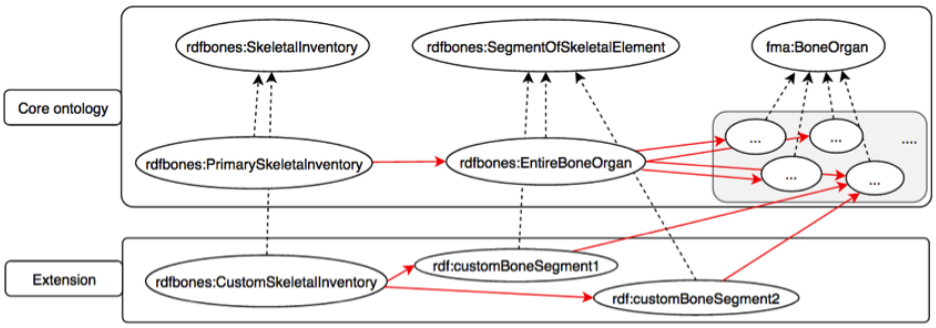


Figure 3.7: Ontology extension scheme

As these extensions are expressed by OWL restriction the application can query the definitions. Consequently if the custom entry form is called from the profile of a skeletal inventory instance, then the entry form processor routine can ask, what bone segment are defined to the type of the subject variable coming as input, and can offer them on the interface.

## 3.2 RDF Data input

### 3.2.1 Multi dimensional form

As it was addressed in the previous section each data input process of the application can be modeled by means of a tree style data structure. This means in terms of the data of the form, that just single key-value pairs like by the static HTML form is not sufficient for the problem. Therefore the task is provide such an interface that allows the user to add dynamically subforms, whose data object will be stored in arrays. Figure 3.9 illustrates the idea of the structure.

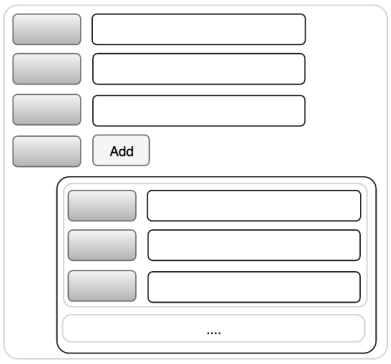


Figure 3.8: Multi dimensional form layout

So the forms consist of the selectors, and literalfield explained in the Chapter 1., but with an additional element that add further subforms. To achieve this JavaScript routine is required that adds the elements automatically and fills the form object with the data. The produced data of the form is looks as follows.

```
{
  key1 : "value1",
  key2 : "value2",
  ...
  subFormKey1 : [
  {
    key1_1 : "value1_1",
    ...
  }, { ... }]
}
```

Listing 3.2: JSON object of the form

### 3.2.2 Form dependencies

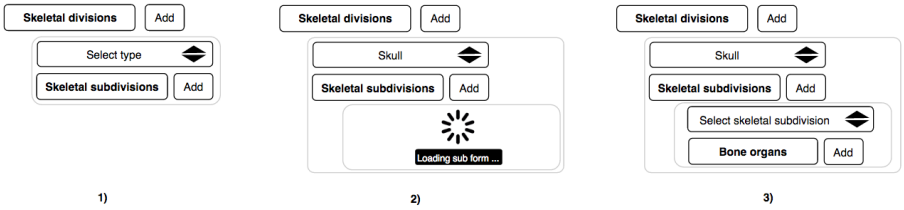


Figure 3.9: Subform dependencies

These dependencies can occur not only between subforms but as well by in form dependencies where the selector elements can change based on selections.

### 3.2.3 Instance browsing

- Figure 3.10 shows a further option for instance selection.

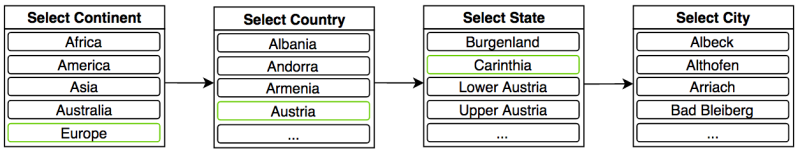


Figure 3.10: Navigator example

- The implementation requires on the server side query and the grouping of the result
- Client side - programming the navigator window

### 3.2.4 Validation

As by each form there is required field - required - restrictions The client has to get the information, about restrictions, and so

### 3.2.5 Editing form data

As it was already addressed in section **??**, the dataset created by the forms have to be edited as well. By editing, the HTTP request calling the entry form contains an additional field, namely the *objectUri*. Based on the data model of the form, the server has to prepare the dataset, in our case a JSON object. The challenge of the server implementation is that in such multi dimensional dataset, it is not sufficient to perform only one query for the whole form data.

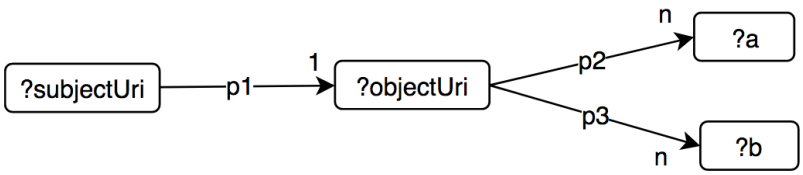


Figure 3.11: Example data model

```
SELECT ?a ?b
WHERE {
  ?objectUri    p1    ?a .
  ?objectUri    p2    ?b .
  FILTER ( ?objectUri = <inputParameter>)
}
```

Listing 3.3: SPARQL query for the form data

The example data model from Figure 3.11 helps to understand the problem in more detail. If the SPARQL query on Listing 3.3 for variables ?a and ?b with incoming *objectUri* value were executed, then result table of the query is inconvenient to process. For example if there are two instances for both ?a and ?b present in the dataset, then the result table contains *2 x 2 = 4* elements (Table 3.1).

Therefore the data object of the form has to be retrieved gradually, by dividing the data model graph by the predicates, whose cardinality is larger than one.

The next step after that the server has prepared the multi dimensional JSON object for the client, is to restore the state of the form, in which it was submitted by the user. This requires firstly the filling of the fields with

| ?a | ?b |
|---|---|
| a1 | b1 |
| a1 | b2 |
| a2 | b1 |
| a2 | b2 |

Table 3.1: SPARQL Result

the existing values, and adding the sub forms based on the arrays. Secondly the options of the selectors must be loaded, so that they conform to the dependencies explained in section 3.2.2.

Finally if a value of selector or literal field changes, or new sub forms has to be added or removed, the entry form data should not be completely sent again to the server, but only the data fields that are concerned by the modification. Thus it does not require a complete page reload, and these operation can be performed through AJAX calls. To achieve this the client has to be prepared to be able to send data modification requests to the server on change event of any form element or sub form.

### 3.2.6 Saving data

Appendix A

# Glossary

Just comment `\input{AppendixA-Glossary.tex}` in Masterthesis.tex if you don't need it!

## Symbols

| | |
|---|---|
| $ | US. dollars. |

## A

| | |
|---|---|
| A | Meaning of A. |

## B

## C

## D

## E

## F

## G

## H

## I

## J

## M

## N

# P

# Q

# R

# S

# T

# U

# V

# W

# X

Appendix B

# Appendix

## B.1 Something you need in the appendix

Just comment `\input{AppendixB.tex}` in Masterthesis.tex if you don't need it!

## Erklaerung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

| | |
|---|---|
| Ort, Datum | Unterschrift |

# Bibliography

[1] OWL 2 web ontology language document overview. W3C recommendation, W3C, October 2009. http://www.w3.org/TR/2009/REC-owl2-overview-20091027/.

[2] Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. On the semantics of sparql. In *Semantic Web Information Management*, pages 281–307. 2009.

[3] Anita Bandrowski, Ryan Brinkman, Mathias Brochhausen, Matthew H. Brush, Bill Bug, Marcus C. Chibucos, Kevin Clancy, Mélanie Courtot, Dirk Derom, Michel Dumontier, Liju Fan, Jennifer Fostel, Gilberto Fragoso, Frank Gibson, Alejandra Gonzalez-Beltran, Melissa A. Haendel, Yongqun He, Mervi Heiskanen, Tina Hernandez-Boussard, Mark Jensen, Yu Lin, Allyson L. Lister, Phillip Lord, James Malone, Elisabetta Manduchi, Monnie McGee, Norman Morrison, James A. Overton, Helen Parkinson, Bjoern Peters, Philippe Rocca-Serra, Alan Ruttenberg, Susanna-Assunta Sansone, Richard H. Scheuermann, Daniel Schober, Barry Smith, Larisa N. Soldatova, Christian J. Stoeckert, Jr., Chris F. Taylor, Carlo Torniai, Jessica A. Turner, Randi Vita, Patricia L. Whetzel, and Jie Zheng. The ontology for biomedical investigations. *PLOS ONE*, 11(4):1–19, 04 2016.

[4] Cornelius Rosse and José L.V. Mejino Jr. A reference ontology for biomedical informatics: the foundational model of anatomy. *Journal of Biomedical Informatics*, 36(6):478 – 500, 2003. Unified Medical Language System.