UNIVERSITY OF GOTHENBURG

Linked Data

A study of how to extract data into a machine readable format by using semantic web technologies

MARTIN AGFJORD

Bachelor Thesis in Applied Information Technology

Report No. 2011:004
ISSN: 1651-4769

University of Gothenburg
Department of Applied Information Technology
Gothenburg, Sweden, March 2011

Abstract

The effort to transform and extend data is a growing business in many industries. Proprietary data formats and inconsistent data structures create complexity for machines to understand these formats, and each new dataset needs human attention in order for it to work in a system.

This study investigates how data can be transformed into a machine understandable format, and make it possible to link and access objects on the web by giving them unique references. Semantic web technologies and linked data have been adopted to investigate this procedure.

The investigation was done by means of the research method of laboratory experiments. A real world example was created from example data provided by AstraZeneca R&D and the organization CDISC. Tests were executed against this example environment to examine the theories behind the semantic web and linked data.

The study shows that data can be parsed into a structured, machine readable, graph data structure with RDF and OWL. The structure can easily be extended. The converted objects can in this new data format be linked to from other repositories of data. Intelligent queries can also be executed against the new data with SPARQL.

The report is written in english.

Keywords: Linked Data, Clinical Data, Semantic Web, AstraZeneca, RDF, OWL, SPARQL, Jena

Acknowledgements

I met many helpful people during my stay at AstraZeneca R&D. I would like to express my special gratitude to researcher Kerstin Forsberg for giving me the opportunity to do this study, her never ending support, her ideas and interest for my work. I would also like to thank researcher Chimezie Ogbuji for his expert advices on semantic web languages.

Finally, I would like to thank my academic supervisor Faramarz Agahi for his exellent guidance, constructive inputs and advices.

March 19, 2011

Martin Agfjord

Contents

1 Introduction

Many organizations are creating alot of data, and the data is often stored in a specific system. The organizations can store the data in any way they want. When two organizations wants to exchange information, the data has to be converted and maybe extended for it to work in a new system. With so many different systems and formats the effort to transform and extend data is a growing business, and it creates complexity when it comes to the distribution of the data.

Problem Motivation

As of today, there is generally no standard of how data should be stored. Even if two organizations stores the exact same information, the actual data can be represented in many different ways. Every industry contains so many different organizations and companies, it is practically impossible to create a standard structure of data.

Organizations can distribute their data on the web today with powerful web friendly data formats such as XML and JSON. A human being with knowledge of XML has no problem understanding the meaning of this kind of data, providing it has the knowledge of the used terms. With this knowledge, a human being can easily use the data in applications. However, computers or machines has no idea of how to handle XML-data without help from a human being. Consider the following two XML examples:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<laboratory-test>
        <type>HE1204</type>
        <name>Hemoglobin</name>
        <value>16</value>
</laboratory-test>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<laboratory-test type="HE1204" name="Hemoglobin" value="16" />
```

The two examples represents the exact same information, but they are using different approaches. The first expresses the data through XML-tags, the second through XML-attributes. Even though computers could know the exact syntax of every programming language in the world, they will not know the structure or the data types of these documents.

There is no standard regarding how to construct XML-documents. Computers can never understand or automatically know how programmers think when they design data sets, because they simply match words. They has to be taught what purpose the XML-tags and attributes has to be able to make intelligent decisions. The same goes with every document on the web, computers has to be teached what the meaning of those documents are.

It is not an option to strictly standardize how data sets should be designed, because it would lead to stagnation, bureaucratic problems or other solutions to bypass the problem, which would lead to them not following the standard anyway.

Problem summary

> Computers can never understand foreign data automatically, they are just matching words. They need a human being to examine the data for them.
>
> Datasets from organizations will never look exactly the same. They have to be loosly coupled.

Therefore, to integrate a new data set into an application, the programmer needs to customize the application for it to understand the new data set[1]. Applications built with this technology will never scale with global sources.

With this technology, data in the world can never be interconnected and work together to combine the huge amount of knowledge that is available.

1.1 Semantic Web as Alternative

The creators of the World Wide Web were early aware of the problems with textbased word matching on the web. In 2001, Tim Berners-Lee published a paper[5] where he proposed that a majority of the information on the web at that time was created solely for people, not machines. Documents on the web needed to be in a format which would allow machines to understand the information, not only display it. The solution has nothing to do with advanced artificial intelligence programming, but with the extra effort to mark each data with extra metadata to define it for computers. With well-defined data, machines can perform sophisticated queries and attain more precise results, instead of attaining irrelevant results from unstructured data. This successor of the web is called The Semantic Web, and contains many tools available for structuring and defining data. The semantic web is a "huge engineeing solution" to the problem of unstructured data.[7]

The semantic web adopts a framework called Resource Description Framework (RDF) and is a common acronym, because it is one of the essential tools in producing structured data. Data created with RDF is made to work in a specific kind of database, called a graph database. The graph database differs from traditional storage of data, e.g.

relational or tree-based databases (i.e. XML). Graph databases store relationships between objects. Relationships are not connected through primary keys as in relational databases. Moreover, there is no intrinsic importance, graph data is simply resources connected with other resources (objects).

Graph databases are therefore schema-less[21], which makes them highly suitable for environments with fuzzy and changing requirements.

A piece of RDF-data is called a triplet. A triplet is a statement of three items, two objects and a relationship (predicate) that connects them[1]. Example of a triplet is:

```
<LabTest> <hasTestPerson> <John> .
```

When describing a triplet, usually the terms subject predicate and object is used. In this case LabTest is the subject, hasTestPerson the predicate and John the object.

Linked Resources

RDF adopts URI's (Uniform Resource Identifier) as identifiers for objects, this makes it possible to create unique objects, point to other objects and be sure that a computer will not mistake the object for another object[1]. The above statement could look like this with URI's:

```
<http://meds.fake/LabTest> <#hasTestPerson> <http://john.fake/John> .
```

The URI's are actually not URL's, but unique objects which a computer will not mistake for other objects representing other Labtests and persons named LabTest or John. This is where the semantic web becomes interesting. Since the objects are identified through URI's it would be convenient if the URI's also would lead to more information (triplets) about the object if one typed that URI into a browser. If it were true, the machine could on its own explore new sources by following the identifier. The technology is called Linked Data and is one of the reasons why the semantic web has become increasingly popular lately. Linked data opens up for a whole new way of sharing data. Instead of linking to the full document of data, one can simply link to the specific object and integrate it directly into the application. Data on the internet is evolving into a "web of data". Resources link to different resources, and those resources link to more data.

The figure below visualizes how some of the semantic data is cross linked over the web. The circles are databases or organizations with data and the arrows shows how individual objects are linked with resources not part of their own domain. The most linked database in this figure is the circle named DBpedia. DBpedia is an initiative to extract data from Wikipedia into semantic data.

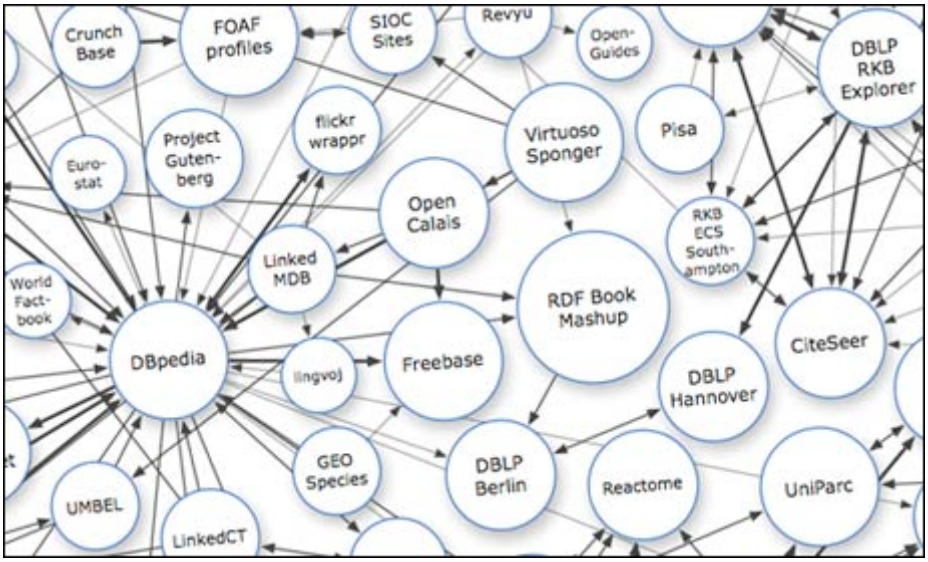

Figure 1: Linked datasets create a web of data.

By extracting data into RDF, the extracted data be cross linked from other domains and link to other resources.

1.2 Producing Semantic Web Data

Li Ding *et. al.* published the paper *Data-Gov Wiki: Towards Linking Government Data*[13] where they described how they could preprocess and enhance the data from the US Government portal site data.gov[1]. The study has been based on how Li Ding *et. al.* exported government data into RDF-documents. Some of the complications they discovered is presented below.

Datasets can be in many different formats, and many of them are often in a proprietary format. These datasets can be faulty and contain blank nodes. Datasets of this kind need to be extracted and preprocessed by an RDF-engine in order to be useful to semantic web applications. In the preprocessing phase, the data gets connected with shared vocabularies and can thereby be inter-connected with other objects[13].

[1] `http://data.gov/`

One of the most powerful qualities of RDF-data is the possibility of linking to other resources. Objects can refer to external objects on other domains, to provide related information[13]. This can for example be used to link laboratory tests to related information about the test, e.g. how the test was taken.

In general, developers have limited knowledge about the semantic web. To satisfy them and make RDF-data available to more people, there is need for a more user friendly format. Web application programmers are used to XML and JSON, the software stack of the semantic web causes them not to be able to take part of data from RDF-documents[16].

Research Question

The above complexity discovered by Li Ding et. al.[13] is something that will be investigated in this thesis. Linked data has to be created from initial data. Their encountered problems has shaped the research question of this study.

How can data be extracted into linked data by using semantic web technologies?

1.3 Delimitation

I have delimited the initial data to clinical data produced by AstraZeneca R&D. The term clinical data can be used to describe many different types of data. This study has been focused on data which represents laboratory tests. The result will represent one solution that potentially will be able to work with many different types of data, not only the laboratory test data.

The implementation part of the study is delimited to the Java programming language, the semantic web Java library Jena. Many examples on the web are programmed in Java, this has been one important motivation why I have chosen this programming language. The Jena framework is developed by Hewlett-Packard Development Company, LP and is used by many developers world wide, it is also programmed in Java. OWL is a W3C recommendation and is used widely by semantic web applications and has many advantages over other languages.

1.4 Disposition

The disposition of the thesis is arranged in the following way.

Chapter 2, Method: This chapter clarifies how the research of this thesis was made. What kind of method I used to find the results and how I got the in data. The chapter also reviews the empirical setting at AstraZeneca R&D and the empirical process.

Chapter 3, Common Semantic Web Tools: This chapters reviews some of the common languages and tools available to build semantic web applications.

Chapter 4, Theory: The research was built upon existing theories from previous work in the field. This chapter explains those theories.

Chapter 5, Empirical Work: This chapter describes how I applied the theory from the previous chapter into a functional software application. The chapter also explaines how the tests that will produce the results was designed.

Chapter 6, Results of Experiments: The results of the thesis is showed and explained in this chapter.

Chapter 7, Discussion: The results are being discussed in this chapter, how they are related to the theory of the thesis. My own thoughts and reflections are included in the discussion. Suggestions for future work are also presented.

Chapter 8, Conclution: This chapter summarizes the discussions of the thesis and concludes the findings in general terms.

2 Research Method

The following chapter will describe the chosen research method for the study and how I collected the in data. The chapter will also define the empirical setting at AstraZeneca R&D and the empirical process.

The main goal of the study is to find out how to extract data into structured linked data, and a efficient way to accomplish this task is to simulate a real world example. This is something that the research method laboratory experiments allows.

2.1 Laboratory Experiments Method

Experiments are something that human beings are doing all the time, but not always are aware of. Every time someone is doing a test which gives a result, it is a experiment. The scientific research method is when the researcher is trying to learn as much as possible from the results. By doing experiments in a supervised environment, a researcher can control the experiments and change conditions and thereby manipulate the results. Cornford & Smithson describe the method as an arrangement of tests with variables that the experimenter is in control of[20]. By changing the input variables, the output result data can change.

To create as accurate results as possible, it is important for a researcher to be aware of every element of an experiment. Laboratory experiments generally test hypotheses to investigate if the results either support or disapprove the hypothesis. The method can also be used to investigate a question or past results. Experiments cannot confirm hypotheses, they can only support them. They can however prove the hypothesis to be false, by doing a counter example if the example is repeatable.

What motivates this research method is primarily the control of variables that are predicted to affect the system and the capabilities to repeat and reiterate the experiments are high. The downside is the complexity and time consuming process of creating an emulation of a real world scenario.

I have chosen to investigate the research question of the thesis by executing designed laboratory experiments on a semantic web environment.

2.2 Empirical Setting - AstraZeneca R&D

AstraZeneca is a pharmaceutical company. They do clinical tests to determine the effects of drugs. The documentation of these tests is stored in SAS-datasets (Statistical Analysis System). These datasets are very similar to Microsoft Excel-documents. They have columns with names and rows with data. The column names are standardized codes from the Study Data Tabulation Model (SDTM) from the organization CDISC (Clinical Data Interchange Standards Consortium). CDISC is a global non-profit organization which develops standards to facilitate the exchange of clinical data. It is convenient to use the same codes as everyone else in the industry, for example to convince authorities that the drugs do not have any side effects (for instance, US authorities do extensive investigations on drugs before they allow them to be sold in the country). When authorities inspect the datasets, they do not have to render the codes because they are familiar with them already.

However, this solution has some drawbacks. The same SAS-dataset is used for various tests, and some of the columns are not used in all tests. If a test does not use every column, the unused columns will contain blank spaces. Another problem is the lack of possibility of extendeding SAS-datasets. At this time, the dataset is not enough for some kinds of tests, therefore they adopt reference-datasets to extend these tests with extra data. This causes the system to be more complex and harder to understand, especially for a third-party investigator.

Because AstraZeneca adopt proprietary systems, it is not possible to automatically understand their datasets within another system, even though they use the same vocabulary for their data. The datasets have to be examined by a human being before they can be understood by a different system.

2.3 In Data Collection

The initial data for the study was collected from CDISC, the organization has made SAS-datasets with artificial data available for the public. The data is created to be simulated and show what data that is used to record laboratory tests looks like. This data can be used to elaborate with. The SAS-format is a proprietary format, and I didn't have any program to read the data. A software developer at AstraZeneca R&D converted the data into Microsoft Excel format. The conversion process was an easy task, the developer simply opened the file in the application SAS Viewer, marked all rows and copied them into a Excel document.

The header of the Excel-file contained all the CDISC-codes. These codes was rendered with a PDF-documentation also available from CDISC, named Study Data Implementation

Guide: Human Clinical Trials. The documentation can be found at CDISC's website[1] and requires a free registration.

With the help from a researcher at AstraZeneca R&D, I removed a few columns in the file to simplify the process to extract the in data. We choosed to focus the in data to tests related to hemoglobin lab tests. Consequently, every column not related to those tests was scraped from the spread sheet. The result was 11 columns and 8 rows not counting the header. The in data was in other words decreased into 8 rows, who all were describing information about hemoglobin lab tests. The columns are explained more in detail in chapter 5.

The Excel-document was later saved as a CSV-file, this can be done in Excel. The CSV-file was used as in data for the application developed in the study to transform the data into RDF.

2.4 Development of a Test Environment - Empirical Process

The empirical process is the development of the semantic web applicaton. The application can read the in data and tranform it into RDF. The end result was a environment that can be used to execute tests on RDF-data.

I've chosen to work with OWL (Web Ontology Language) to represent knowledge in a semantic structure. OWL is explained more in detail in chapter 3. An OWL ontology was developed for the study. The ontology includes classes for Hemoglobin lab tests and persons.

The semantic web application was developed in Java EE with help from the semantic web library Jena. The application contains two main parts.

The first part is the one that reads a CSV-file and converts each row into RDF-resources and triples, the application serializes them into a OWL-document. The second part reads the OWL-file and the OWL-ontology and makes the resources and triples available on the web with help from SPARQL and servlets. The empirical process is explained more in detail in chapter 5.

Four experiments was designed to execute tests on the new data. The tests used different methods to access the data from the application. The results of the tests was analyzed by refeering how they was related to the theory of the thesis.

[1] `http://www.cdisc.org/sdtm`

2.5 Validity

The research background and the theoretical framework of the thesis was found by searching the web for research papers and analyzing them for the previous work in the field. The first weeks of the study was dedicated to attaining a good foundation of the semantic web. The fact that the semantic web framework contains a rather big software stack was discovered. To be able to understand the research problems of this thesis, a solid understanding of the common terms used while developing semantic web applications is needed. Consequently, the next chapter of this thesis is dedicated to describing these terms.

As a result of the study of previous work in the field, chapter 4 describes a few directives that previous researchers have observed when developing a semantic web application.

The involved research problems of this thesis were chosen by looking at articles that had made successful extractments of data in other fields, and adopting them into the field of clinical data. Articles were found by using the search engine Google Scholar[2]. Articles were chosen by [i] looking at the number of times the article had been cited and [ii] the year of publication.

[2] http://scholar.google.com/

3 Common Semantic Web Tools

Semantic web applications were first introduced through the Resource Description Framework (RDF), but the creators quickly became of that the framework was not sufficient to successfully fulfil the vision of the semantic web. The tools available today to create semantic web applications are a software stack that builds upon RDF. This chapter will explain some of the common tools available.

3.1 RDF

One of the problems with the web today, as described in the introduction chapter, is that the same information can be expressed in many different ways (see the XML-example in the introduction), and machines cannot understand the information, because they are not familiar with the data structure. Actually, this is also the case with RDF. RDF can be expressed with different syntaxes, but in the end, it is the underlying triplets which define the data, not the RDF-syntax[19]. The following sections will describe three different syntaxes of RDF.

RDF/XML

One of the most frequently used syntaxes is RDF/XML. It adopts, like its name says, XML-syntax. To create an example, let us define a describing document about the laboratory test showed earlier.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:cdisc="http://reference.cdisc.org/LabTests#">
  <rdf:Description rdf:about="http://data.astrazeneca.org/LabTest01">
    <cdisc:hasTestPerson rdf:resource="http://john.fake/John" />
  </rdf:Description>
</rdf:RDF>
```

This is how the triplet can be defined with RDF/XML, with some improvements. The first tag defines a RDF-namespace, so that computers will understand that the document is not an ordinary XML-document. Resources are defined with the Description tag in the RDF-namespace, and the identifier is created with the about attribute in the same

namespace. The LabTest01 object is connected with John with the tag hasTestPerson from the namespace cdisc. The namespaces will be explaned in further detail later.

Notation3

Another syntax is called Notation3 (N3), and can in many cases be more straightforward and compact than RDF/XML. N3 is designed to be human readable and is being developed by the Semantic Web Interest Group at World Wide Web Consortium (W3C)[6].

```
@prefix cdisc: <http://reference.cdisc.org/LabTests#> .
@prefix az: <http://data.astrazeneca.org/> .
az:LabTest01 cdisc:hasTestPerson <http://john.fake/John> .
```

N3 offers programmable relationships in RDF.

```
@prefix cdisc: <http://reference.cdisc.org/LabTests/> .
@prefix az: <http://data.astrazeneca.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <http://www.example.org/> .
<http://john.fake/John> :age "65" .
az:LabTest01 cdisc:hasTestPerson <http://john.fake/John> .
{?b :age ?c . ?c :moreThan "55" } => {?a :isA cdisc:SeniorTest} .
```

The above example shows first that John has the age 65 and states that John is part of the az:LabTest01. The code then checks if the LabTest-person has an age that is more than 55, if its true, another triple is generated automatically that states that the test is a cdisc:SeniorTest.

N-Triple

N-Triple is a simplified and read only version of Notation3[11]. It adopts full identification to every resource and relationship.

```
<http://data.astrazeneca.org/LabTest01>
<http://reference.cdisc.org/LabTests/hasTestPerson>
<http://john.fake/John>.
```

The code represents one statement. The statements are separated with dots, do not mind the line breaks. The layout of this document was simply not big enough.

3.2 Namespaces

The above examples have shown how to produce triplets for defining well-structured data. Objects has been defined with unique references with URI's. Also, predicates have been using URI's. This how RDF allows to avoid custom programming when developing applications. If the computer which is reading the data is not familiar with the predicates, it would only understand the structure, not the meaning of the data[1]. In other words, it would not understand what kind of relationship the objects have. Therefore, it is important to try to use as common namespaces as possible when choosing predicates.

The examples in the above sections have used the cdisc namespace, unfortunately that namespace is made up. Computers will not know how to understand the data, but for explaining the use of namespaces it was necessary to create relevant clinical examples of RDF.

For other kind of data, there are several namespaces available. The Friend of a Friend (FOAF) namespace is often used to describe people. The Dublin Core namespace can be used to describe documents. MusicBrainz for music artists.

3.3 Vocabularies and Languages

The previously introduced methods for handling structured data have provided the possibility of creating a relationship between objects. The relationships can be distributed and used in applications which can use the information, because the machines understand the meaning of the data. However, there is still no real semantics behind the data. Computers using information about Labtest01 would have to "guess" or simply try every predicate there is, to find any useful information. There is no data model behind it which tells the computer that the Labtest01 object has some base values, such as a lab value, a test person and a date.

The Web Ontology Language (OWL) is one solution to this problem. OWL is a language used to create an ontology in the semantic web. The primary reason to create an ontology is to represent information about objects and how they are related. OWL defines these semantics as classes, sub classes and properties.[8]

"Ontologies are formalized vocabularies of terms, often covering a specific domain and shared by a community of users."[18]

- W3C

Class-instances in OWL is however not the same thing as an instance of a class in object oriented programming. One difference is that the triples which represents an instance

of a class is not bound to that OWL-class by any means. They can however together acquire the status of a class-instance, if they fulfil the requirements from the class. In programming environments, an OWL-reasoner is used to make this possible.

Classes

Classes are used to encapsulate values in a specific domain. A class can have a subclass, which will have inherit all the properties from its parent class.

Example of an OWL Ontology defining a class and a sub class (in N3-syntax):

```
1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3 @prefix owl: <http://www.w3.org/2002/07/owl#> .
4 @prefix cdisc: <http://reference.cdisc.org/def/owl#> .
5
6 <http://reference.cdisc.org/def/owl#> rdf:type owl:Ontology ;
7 rdfs:comment "An example clinical ontology
8         to show the basics about OWL"^^rdf:PlainLiteral ;
9
10 cdisc:LabTest a owl:Class .
11
12 cdisc:HemoglobinLabTest a owl:Class;
13         rdfs:subClassOf cdisc:LabTest .
```

There are a few new things introduced in the code. Line 1, 2 and 3 devine namespaces used by OWL and line 10 defines the namespace used to refer to this ontology, named cdisc. At line 6 the object referring to the ontology is set as an ontology, and a comment is associated with the object.

At line 10 a class is defined, for now the class does not contain any information, other than that its name is LabTest and it belongs to the cdisc namespace. Line 12 defines a sub class of the class called HemoglobinLabTest.

Properties

Properties are used to give the classes relationships to other objects or definitive values. Properties refeering to other objects are called ObjectProperty, properties referring to values are called DataTypeProperty[12]. Properties combine two objects/datatype's with a predicate. ObjectProperty defined:

```
1 cdisc:hasCategory
2       a       owl:ObjectProperty .
```

Properties can be restricted to certain values or objects.

```
cdisc:hasCategory
  a  owl:ObjectProperty ;
  rdfs:domain cdisc:LabTest ;
  rdfs:range cdisc:Category .
```

rdfs:domain restricts the subject of a triple, rdfs:range restricts the object of a triple. In the above example it means that the instantiated object must be of the class, or sub class of cdisc:LabTest and the property has to be an instance of the class cdisc:Category.

To demonstrate the previous explanation about how OWL can be used to define classes that always have certain properties, an example is needed. Let us create a class that assumes the following:

Everything that is a laboratory test and has the category "hemotology" is a hemoglobin test.

```
cdisc:HemoglobinLabTest
  a owl:Class ;
  owl:equivalentClass
  [ a owl:Class ;
    owl:intersectionOf
      ( cdisc:LaborationTest
        [ a owl:Restriction ;
          owl:onProperty cdisc:hasCategory ;
          owl:hasValue cdisc:Hematology
        ])
  ] .
```

This code makes it possible for a computer program to know that a resource and its set of triples together are the same as an instance of the class HemoglobinLabTest, even if they are not actually are an instance of the class. This is achieved with owl:equivalentClass and owl:intersectionOf. intersectionOf takes a collection of classes as argument. It assumes that the resource fulfil the requirements of every class in the collection. In this case it means; 1. a resource must be an instance of cdisc:LaborationTest (or a subclass). 2. it must have the property cdisc:hasCategory, and that property must point to the object cdisc:Hematology.

To accomplish the second restriction an anonymous class has been created, which makes the restriction. In other words, to be able to call a resource a cdisc:HemoglobinLabTest, it must be possible for it to be an instance of cdisc:LabTest and the defined anonymous class.

3.4 Sophisticated Queries

RDF-documents can quickly become large. Graph-databases often contain millions of triples. Like most database systems, also RDF has a query language to get relevant information instead of searching through a whole database[23]. SPARQL is an recursive acronym and stands for SPARQL Protocol and RDF Query Language. SPARQL asks queries against the graph-database, and since there is no schema, or many different tables with primary keys, it is much easier to combine questions compared to traditional systems. Answers which would have been hard to attain with relational queries are easy to get with SPARQL. SPARQL is known to be faster than relational databases too.

To put this in context, a simple SPARQL-query to select every resource that is an instance of the class HemoglobinLabTest has been defined below.

```
1 PREFIX cdisc: <http://reference.cdisc.org/def/owl#>
2 SELECT ?resource WHERE {
3   ?resource a cdisc:HemoglobinLabTest .
4 }
```

3.5 Linked Data Principles

As mentioned earlier, linked data is about making object URI's dereferenceable. This does not however mean that every object has to be in a separated RDF-document. Instead, the URI's should lead to a website which performs a SPARQL query to the graph-database, and retrieves the information about the object[19].

Many websites provide both a browser interface and the RDF-data in its raw form, however the URI to the object can lead to both these sites. The webserver simply detects if the user agent is coming from a web-browser or an RDF-client.[13] One way to accomplish this is to give the HTTP header attribute "Accept: " the value "application/rdf+xml" to retrieve RDF/XML syntax.

4 Theory

This chapter introduces three directives to follow when extracting existing data and developing semantic web applications. The advice is more or less based on the article *Data-Gov Wiki: Towards Linking Government Data*[13]. The article describes how researchers extracted data from the site Data.gov[1]. The article only describes brief information about what kind of tools they used, not any detailed tutorials. As a result, a deeper study about the terms used in the article was necessary, and the results of some those terms were previously presented in chapter 3.

The semantic web and its tools are growing and the technology to develop applications is being outdated quickly, the motivation to choose and base the previous work solely on one article was motivated by that it was published in 2010. Nonetheless, some of the recommendations in the article were contradicted by other researchers results.

4.1 Producing Linked Data

The data used in this study has been restricted to the CSV-format. CSV files contain a header-row with column names and afterwards the data, separeted by the comma symbol.

With CSV-files, the conversion to RDF is very straightforward. Li Ding *et. al.*[13] propose that data transformation should be "minimal and extensible". The reason is to keep the original data structure and creating as simple triples as possible. Simple triples make the RDF-data more understandable, which can facilitate for developers to use and extend the data.

The transformation procedure includes creation of URI's for each row of data, not counting the header. The values in the row are mapped into the corresponding RDF-property. The properties are created from the header-row. According to Li Ding *et. al.*[13], RDF-properties must be linked to the dataset it comes from, i.e. the properties must only be used in RDF-documents generated from the corresponding CSV-file. Properties can mean different things in different situations. This however contradicts the whole idea of the semantic web, that the documents should be machine readable. Documents with unique properties will only be readable to customized applications. According to

[1] `http://data.gov/`

Doan *et. al.*[14] data should be mapped to properties with ontologies. It is an significant feature that is needed for documents to be machine readable.

To make the RDF-documents available and usable on the web, the format RDF/XML has been chosen by Li Ding *et. al.*[13]. RDF/XML has more readability than other RDF-formats. Both RDF and XML-readers can understand it, XQuery and SPARQL can query data from the format.

The unique URI for each row shall be dereferenceable by using the HTTP-protocol[2]. This can be done by for example having a servlet listening on a base URL i.e.

`http://example.org/resource/`. Everything that comes after that URL can be queried by the servlet. A SPARQL-DESCRIBE query retrieves all triplets related to the subject. For example, the URL:

```
1 http://example.org/resource/Hemoglobin
```

will retrieve all triplets related to the object that has the same URI.

For a developer, it is not hard to understand that this kind of procedure needs some kind of programming environment to work in an application. In the book *A Developer's Guide to the Semantic Web*[19] the semantic web framework Jena is used for this cause. Jena is a set of Java-libraries that can parse RDF documents into Java objects and vice versa. It also has tools to execute SPARQL-queries on RDF-documents. Finally, it can handle OWL-ontologies. These tools make it a very suitable framework for a Java EE environment to create a semantic web application.

4.2 Linking Data

Li Ding *et. al.*[13] converted many datasets with government data, and identified data which was very similar or identical to other datasets. They created a namespace to use with datasets that treated the same type of information. It is also possible to use the same class if using an ontology[1].

Ontologies can provide programmers with opportunity to link data in many ways. In particular, they are very good at linking to related data. Semantic web tools available can link describing texts to URI's, it can therefore be easier to explore new linked data.

When dealing with resources that have an object in another dataset describing the same object, the two resources can be linked in a statement with owl:sameAs. The object can thereby be identified as a common object, even if it has a unique URI.

4.3 Increasing the Usefulness of Linked Data

The growing software stack of the semantic web is making it harder and harder for a developer to understand and consume RDF. The different RDF-syntaxes require special libraries for programming languages to be parsed. John Sherida and Jeni Tennison[16] describe that highly experienced programmers failed to understand a SPARQL resource at data.gov.uk[2].

Most developers are however known to understand RESTful web services using API's with popular return syntaxes. Data.gov.uk developed a layer on top of the SPARQL service that transformed the RDF into JSON or XML[16]. With this layer, the provider of the data can store it as RDF, link to other RDF-objects and perform intelligent SPARQL queries and still return developer friendly end results.

[2] `http://data.gov.uk`

5 Empirical Work

This chapter explains how the real world example was created from the in-data provided by AstraZeneca R&D.

A summary of the conversion process is presented below:

- Copying SAS-dataset into Microsoft Excel.
- Saving the Excel-document as a CSV file.
- Reading the CSV-file in Java with help from OstermillerUtils.
- Creating hash maps from the headers and the values.
- Creating resources and triples from values in the hash maps.

The end results from this conversion are merely triples containing the data in RDF-format. To be able to create queries to retrieve information about specific laboratory tests, an ontology is needed. The ontology will be used to define what kind of values a laboratory test will have to contain to be a valid instance of a class in the ontology. The application will read the ontology and define which triples that together represent instances of classes.

Resources will get unique URI's by the conversion process and become dereferenceable via a SPARQL end point.

5.1 Initial Datasets

Before defining an ontology, the initial dataset is studied. Data received from a hemoglobin test was converted from SAS-documents into Microsoft Excel through copy and paste of the columns and rows. The first two rows are represented in OpenOffice Calc below.

The bold text is the header, containing CDISC-codes. The header names will be explained below.

- STUDYID. Study id which the test is part of, for example a study to test a certain product for side effects.
- USUBJID. User Subject Identification. It represents the person who was involved in the test, the test person.

A	B	C	D	E
STUDYID	**USUBJID**	**LBSEQ**	**LBTESTCD**	**LBTEST**
CDISC01	CDISC01.100008	11	HGB	Hemoglobin
CDISC01	CDISC01.100008	12	HGB	Hemoglobin

F	G	H	I	J	K	L
LBCAT	**LBORRES**	**LBORRESU**	**LBSPEC**	**LBFAST**	**LBDTC**	
Hematology	12.0	g/dL	BLOOD	Y	2003-04-15T	11:20
Hematology	11.3	g/dL	BLOOD	N	2003-10-13T	11:55

Figure 2: Hemoglobin laboratory test data

- LBSEQ. A unique number to ensure the uniqueness of the lab test within the dataset.
- LBTESTCD. Short name of the measured item in the test.
- LBTEST. The measured item in the test.
- LBCAT. The category of the test.
- LBORRES. The measured value of the test.
- LBORRESU. The unit of the measured value.
- LBSPEC. The fluid of how the test was taken.
- LBFAST. Describes if the person was fasting or not while participating in the test.
- LBDTC. The date of the test.

5.2 Ontology Development

Five classes has been developed from the dataset. The main class to represent a hemoglobin lab test was taken from the example in chapter 3. It was supplemented by three additional restrictions. A resource needs the following relations to be equal to a hemoglobin test.

- Be an instance of the class LabTest
- Measures Hemoglobin.
- Have a fasting value
- Have a date for the test
- Have a clinical value

The class in N3 syntax:

```
cdisc:HemoglobinLabTest
  a owl:Class ;
  owl:equivalentClass
  [ a owl:Class ;
    owl:intersectionOf
      ( cdisc:LabTest
        [ a owl:Restriction ;
          owl:onProperty cdisc:isMeasuring ;
          owl:hasValue cdisc:Hemoglobin
        ]
        [ a owl:Restriction ;
          owl:onProperty cdisc:hasClinicalValue ;
          owl:someValuesFrom cdisc:ClinicalValue
        ]
        [ a owl:Restriction ;
          owl:onProperty cdisc:hasDate ;
          owl:someValuesFrom xsd:dateTime
        ]
        [ a owl:Restriction ;
          owl:onProperty cdisc:isFasting ;
          owl:someValuesFrom xsd:boolean
        ])
  ] .
```

The class LabTest has no restrictions.

```
cdisc:LabTest
      a       owl:Class .
```

To encapsulate values related to the actual test value, the class ClinicalValue has been created. An instance of the class needs to fulfil the following requirements:

- Be a part of a LabTest
- Have a value
- Have a unit
- Have a value which explains the fluid of the measured test.

The class in N3 syntax:

```
cdisc:ClinicalValue
  a owl:Class ;
  owl:equivalentClass
  [ a owl:Class ;
    owl:intersectionOf
```

```
6       (
7         [ a owl:Restriction ;
8           owl:onProperty cdisc:isPartOf ;
9           owl:someValuesFrom cdisc:LabTest
10        ]
11        [ a owl:Restriction ;
12          owl:onProperty cdisc:hasValue ;
13          owl:someValuesFrom xsd:float
14        ]
15        [ a owl:Restriction ;
16          owl:onProperty cdisc:isMeasuredIn ;
17          owl:someValuesFrom xsd:string
18        ]
19        [ a owl:Restriction ;
20          owl:onProperty cdisc:isMeasuredFrom ;
21          owl:someValuesFrom xsd:string
22        ])
23  ] .
```

The class Person is just a reference class, and does not have any values yet.

```
1 cdisc:Person
2       a       owl:Class .
```

The final class is named Hemoglobin. It is connected to a lab test with the predicate cdisc:isMeasuring. The class hemoglobin is connected with another ontology through the `owl:sameAs` predicate.

```
1 cdisc:Hemoglobin
2       a       owl:Class ;
3       owl:sameAs ncicb:Hemoglobin .
```

NCICB is an acronym for National Cancer Institute Center for Bioinformatics and provides an OWL-ontology. This code states that the resource is the same as hemoglobin provided by NCICB.

5.3 Conversion of CSV-files

The following section will explain how the first entry in the CSV-file was parsed into a RDF-resource.

```
STUDYID,USUBJID,LBSEQ,LBTESTCD,LBTEST,LBCAT,LBORRES,LBORRESU,LBSPEC,LBFAST,LBDTC
CDISC01,CDISC01.100008,11,HGB,Hemoglobin,Hematology,12.0,g/dL,BLOOD,Y,2003-04-15T11:20
CDISC01,CDISC01.100008,12,HGB,Hemoglobin,Hematology,11.3,g/dL,BLOOD,N,2003-10-13T11:55
```

Figure 3: Hemoglobin laboratory test data in CSV-format

Parsing CSV-files into Jena-objects

The Java library OstermillerUtils[1] has been used to read and parse CSV-files. The first two values are values concerning what study and which person took the test. They are being used as identifiers for the resource. The third value is a customized solution, it is a number that represents that this is a lab test. Lab tests have number 01 and clinical values have the number 02. The fourth value is the sequence number to ensure uniqueness within a study.

The first resource created has the study id `CDISC01`, user id `100008` and unique number `11`. The URI is showed below.

```
1 http://data.astrazeneca.com/resource/CDISC01/100008/01/11/
```

The second resource will represent a clinical value. It attains almost the same URI as the LabTest, with one small difference, the identifier has changed from `100008/01/11` to `10000/02/11`.

```
1 <http://data.astrazeneca.com/resource/CDISC01/100008/02/11>
```

The last resource created is the one representing a person.

```
1 <http://data.astrazeneca.com/resource/CDISC01/100008>
```

Predicates (properties) assign values to the resources.

5.4 Making Resources Dereferenceable

To be able to link to objects created from CSV-files, all resources has got an unique URI from the extractment process. Each URI created has the same base URI:

```
1 http://data.astrazeneca.com/resource/
```

After that part, the resources URI's gets shaped by the specific values. The reason for this base URI is that the Java servlet listens to the `/resource/` directory. Every URL with `/resource/` as base URL will be processed through the SPARQL-servlet. A sidenote here, on the test computer, the hosts-file is redirecting all local traffic which tries to access `data.astrazeneca.com` to the local machine. The subdomain

[1] `http://ostermiller.org/utils/`

`data.astrazeneca.com` does not exist of the performance of this study. It is a fictional example.

To retrieve triples about a object, e.g.

```
1 http://data.astrazeneca.com/resource/28/28464
```

The SPARQL-servet creates a DESCRIBE query:

```
1 DESCRIBE <http://data.astrazeneca.com/resource/28/28464>
```

The query will retrieve a RDF-document containing every triplet in the graph associated with the object.

Since RDF is a language with many syntaxes, the results can be in various formats. If the client is a web browser, the results will be a HTML-page which will display the triplets in a user-friendly way. This HTML-presentation is actually merely N-Triples in a fancy way.

If the client however comes from a semantic web-client, an RDF/XML-document will be retrieved. The SPARQL-servlet looks at the HTTP header to find out what kind of document the client accepts. For example, semantic web browsers can use the following code in the header:

```
1 Accept: application/rdf+xml
```

Web browsers can however also retrieve the triples in other formats. The use of parameters is adopted to accomplish this:

```
1 http://data.astrazeneca.com/resource/28/28464?format=RDF/XML
```

The above URL retrieves a RDF-document in the RDF/XML-format. Other formats available are N-TRIPLE, TURTLE, N3 and JSON. The three first are provided by Jena. Jena does not provide any JSON-format for DESCRIBE-queries. To produce JSON-format, Talis[2] has been used. Talis is an open source library which works with Jena.

5.5 Design of Tests on New Data

Sophisticated queries - search for specific items

This test's purpose is to examine the capability of retrieving specific items, with certain conditions. The test uses the SPARQL-engine in a web browser. The following queries are going to be executed against the SPARQL-engine:

[2] `https://github.com/talis/rdf-json-writer`

```
PREFIX cdisc: <http://reference.cdisc.org/ontology#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
     SELECT ?resource ?realValue ?date WHERE {
          ?resource a cdisc:HemoglobinLabTest.
          ?resource cdisc:hasDate ?date.
          ?resource cdisc:hasClinicalValue ?value.
          ?value cdisc:hasValue ?realValue.
}
```

The query asks the graph for a resource that fulfil the requirements of the class HemoglobinLabTest. It specifies variables in the query as the values and references to objects that the resource has.

The next query to be executed is displayed below.

```
PREFIX cdisc: <http://reference.cdisc.org/ontology#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
     SELECT ?resource ?date ?realValue WHERE {
          ?resource a cdisc:HemoglobinLabTest.
          ?resource cdisc:hasDate ?date.
          ?resource cdisc:hasClinicalValue ?value.
          ?value cdisc:hasValue ?realValue.
          FILTER(?realValue >= "14"^^xsd:float)
          FILTER(?date < "2004-01-28T00:00:00Z"^^xsd:dateTime)
}
```

This query asks the graph for the same resources as the previous query, in addition, it also filters the results to only hemoglobin values higher or equal to 14 and lab tests before the date 2004-01-28.

The results of both tests should give a list with [i] URI's to the objects representing the resources, [ii] the date and [iii] the value of the test.

Access linked data through linked data principle

The previous test is supposed to retrieve a list of URI's which representing resources. This test's purpose is to try out if a client can follow the URI's that the previous test provided and retrieve more information about the object. The URI of a resource will be put into a web browser (thereby through the HTTP-protocol).

Access data through terms from unknown and foreign vocabularies

The purpose of this test is to try to search for anything that is related to Hemoglobin with terms not defined in the original ontology. The test contains three SPARQL-queries. The first two tests have no idea of any vocabulary which can be used. It searches with and without wildcards (the asterisk, '*'). The third query uses an ontology provided by NCICB.

Query 1:

```
1      SELECT ?resource ?predicate WHERE {
2          ?resource ?predicate Hemoglobin .
3 }
```

Query 2:

```
1      SELECT ?resource WHERE {
2          ?resource ?predicate *:Hemoglobin* .
3 }
```

Query 3:

```
1 PREFIX ncicb: <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#>
2      SELECT ?resource WHERE {
3          ?resource ?predicate ncicb:Hemoglobin .
4 }
```

Manipulate the HTTP header to retrieve other syntaxes

This test is going to examine if it is possible to fetch various different syntaxes by manipulating the HTTP header attribute: `Accept`. The accept value will be tested with `application/rdf+n3`, `application/rdf+xml` and `application/rdf+json`. To manipulate the header, the command line application cURL is used.

6 Results of Experiments

This chapter presents the findings of the experiments. The tests were executed against the semantic web application developed in the work of this thesis.

6.1 Test 1 - Sophisticated queries: Search for specific items

The test executed two queries against the SPARQL end point. The adress of the SPARQL end point is `http://data.astrazeneca.com/`. The end point contains a form with a textarea where a SPARQL-query is supposed to be written. The server responded with a table with three columns for each query. The first column contains URI's to resources that fulfil the requirements of a HemoglobinLabTest. The URI's are hyperlinked to their URI. The second column, named realValue, contains numbers of the datatype float. The third column contains dates of the datatype dateTime. The datatypes are defined by W3C's XMLSchema.

The first table below.

resource	realValue	date
<http://data.astrazeneca.com/resource/CDISC01/100014/01/11>	"13.0"^^<http://www.w3.org/2001/XMLSchema#float>	"2004-03-30T12:50:00Z"
<http://data.astrazeneca.com/resource/CDISC01/200001/01/12>	"13.5"^^<http://www.w3.org/2001/XMLSchema#float>	"2004-02-02T07:58:00Z"
<http://data.astrazeneca.com/resource/CDISC01/200002/01/11>	"14.2"^^<http://www.w3.org/2001/XMLSchema#float>	"2003-09-19T12:45:00Z"
<http://data.astrazeneca.com/resource/CDISC01/100014/01/10>	"13.7"^^<http://www.w3.org/2001/XMLSchema#float>	"2003-10-06T09:30:00Z"
<http://data.astrazeneca.com/resource/CDISC01/100008/01/11>	"12.0"^^<http://www.w3.org/2001/XMLSchema#float>	"2003-04-15T09:20:00Z"
<http://data.astrazeneca.com/resource/CDISC01/200001/01/11>	"14.0"^^<http://www.w3.org/2001/XMLSchema#float>	"2003-09-09T09:02:00Z"
<http://data.astrazeneca.com/resource/CDISC01/200002/01/12>	"13.4"^^<http://www.w3.org/2001/XMLSchema#float>	"2004-03-29T07:40:00Z"
<http://data.astrazeneca.com/resource/CDISC01/100008/01/12>	"11.3"^^<http://www.w3.org/2001/XMLSchema#float>	"2003-10-13T09:55:00Z"

Figure 4: Query 1 - Ask for every instance of the class HemoglobinLabTest.

The result gives 8 rows. The second table below.

resource	realValue	date
<http://data.astrazeneca.com/resource/CDISC01/200002/01/11>	"14.2"^^<http://www.w3.org/2001/XMLSchema#float>	"2003-09-19T12:45:00Z"
<http://data.astrazeneca.com/resource/CDISC01/200001/01/11>	"14.0"^^<http://www.w3.org/2001/XMLSchema#float>	"2003-09-09T09:02:00Z"

Figure 5: Query 2 - Ask for every instance of the class HemoglobinLabTest and delimit them to lab values higher or equal to 14 and tests taken before the date 2004-01-28.

The result gives 2 rows.

6.2 Test 2 - Access data through linked data principle

This test takes the first URI received from the second query in the previous test, *http://data.astrazeneca.com/resource/CDISC01/200002/01/11*, and executes it in Chromium[1] web browser. When an URI is inputed and executed, the SPARQL servlet is triggered and creates a DESCRIBE query. The query retrieves every triplet related to the specified URI. The triples are presented as a HTML table. The table contains two columns. Each row are together with the input URI is a triplet. The subject of each triple is the URI. The predicate of each triple can found in the first column, and the object in the second.

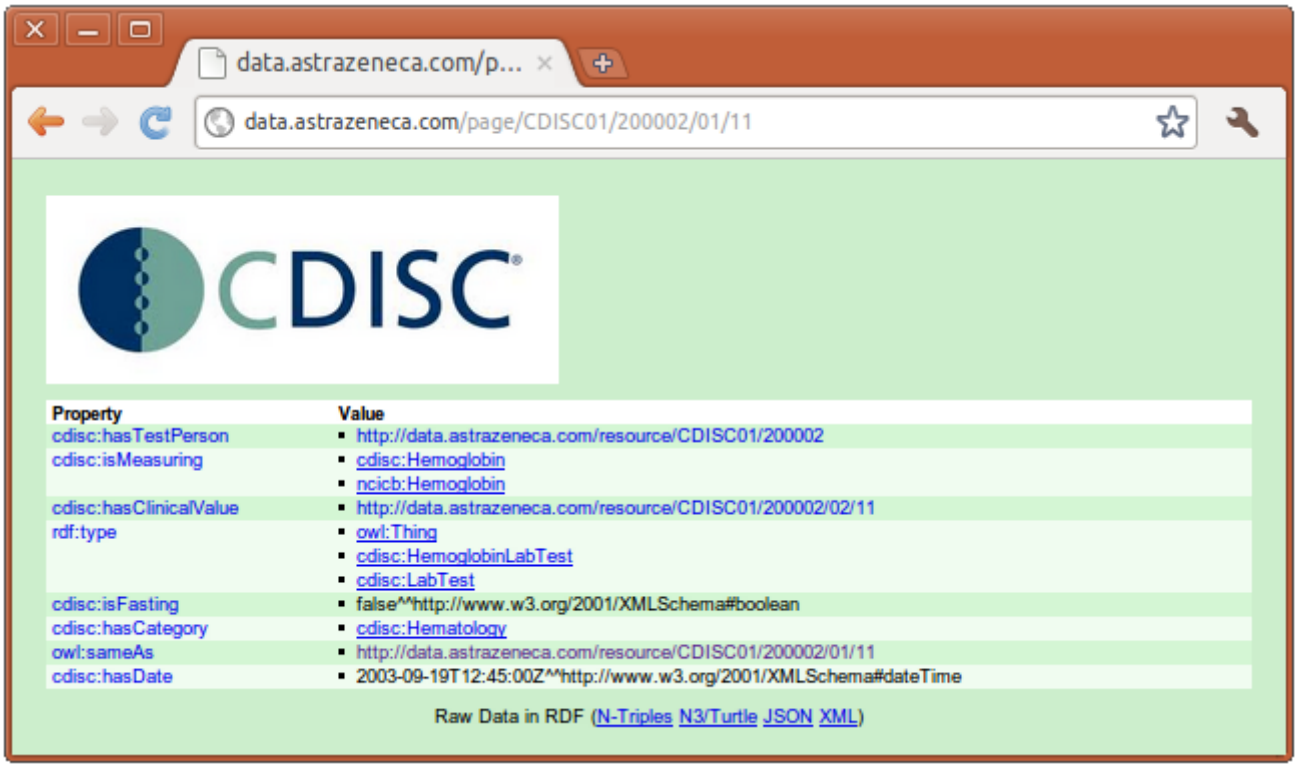

Figure 6: Web page 1.

Web page 1 - Site given when inputing the resource URI of an object. The web page shows what triples the resource has. ObjectProperties are hyperlinked to the object URI.

[1] http://www.chromium.org/

To test if it is possible to link to other objects within a resource, the next part of the test is to simply click on one of the URI's, the one the predicate cdisc:hasClinicalValue points to, *http://data.astrazeneca.com/resource/CDISC01/200002/02/11*.

The result can be seen in the figure below.

Figure 7: Web page 2.

Web page 2 - Site given when clicking the link of the object to the property cdisc:hasClinicalValue of the previous resource.

The second web page was an internal site within the application. It was linked from

http://data.astrazeneca.com/resource/CDISC01/200002/01/11. The next part of the test start at the first web page. The predicate cdisc:isMeasuring points to an object named ncicb:Hemoglobin. The URI of the object is

http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#Hemoglobin.

The third part of the test follows that URI and tries to reach the object.

Web page 3 - The URI to ncicb:Hemoglobin did not give any web site. Instead it returned an OWL-file, Thesaurus.owl, a file with the size 207 MB. It took 13 minutes to download it. The file contains the object that cdisc:isMeasuring pointed to, but it also contains several of other objects.

6.3 Test 3 - Access data through terms from unknown and foreign vocabularies

This test used the same SPARQL endpoint as the first test. The first two queries of the test did not give any results, they could not be executed by the SPARQL-engine. The third query gave a table with two columns. The query asked to get every resource and predicate that was part of a triplet where

`<http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#Hemoglobin>` was included. Each row of the returned table is the subject and predicate that is connected with

`<http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#Hemoglobin>`. The first column is the subject and the second is the predicate.

resource	predicate
<http://reference.cdisc.org/ontology#Hemoglobin>	<http://www.w3.org/2002/07/owl#equivalentClass>
<http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#Hemoglobin>	<http://www.w3.org/2002/07/owl#equivalentClass>
<http://reference.cdisc.org/ontology#Hemoglobin>	<http://www.w3.org/2000/01/rdf-schema#subClassOf>
<http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#Hemoglobin>	<http://www.w3.org/2002/07/owl#sameAs>
<http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#Hemoglobin>	<http://www.w3.org/2000/01/rdf-schema#subClassOf>
<http://reference.cdisc.org/ontology#Hemoglobin>	<http://www.w3.org/2002/07/owl#sameAs>
<http://data.astrazeneca.com/resource/CDISC01/200002/01/12>	<http://reference.cdisc.org/ontology#isMeasuring>
<http://data.astrazeneca.com/resource/CDISC01/200001/01/11>	<http://reference.cdisc.org/ontology#isMeasuring>
<http://data.astrazeneca.com/resource/CDISC01/100014/01/11>	<http://reference.cdisc.org/ontology#isMeasuring>
<http://data.astrazeneca.com/resource/CDISC01/200001/01/12>	<http://reference.cdisc.org/ontology#isMeasuring>
<http://data.astrazeneca.com/resource/CDISC01/100014/01/10>	<http://reference.cdisc.org/ontology#isMeasuring>
<http://data.astrazeneca.com/resource/CDISC01/200002/01/11>	<http://reference.cdisc.org/ontology#isMeasuring>
<http://data.astrazeneca.com/resource/CDISC01/100008/01/11>	<http://reference.cdisc.org/ontology#isMeasuring>
<http://data.astrazeneca.com/resource/CDISC01/100008/01/12>	<http://reference.cdisc.org/ontology#isMeasuring>

Figure 8: Query 3 - Ask the server for every resource and predicate that has a relationship with a specified object.

The query returns 14 rows with two columns.

6.4 Test 4 - Manipulate the HTTP header to retrieve other syntaxes

This test uses the command line application cURL[2] to send HTTP-requests to the semantic web application with a manipulated header to retrieve different RDF-syntaxes. Three commands are going to be executed, but they will all access the same URL. The URL is an URI to a RDF resource, *http://data.astrazeneca.com/resource/CDISC01/200002/01/11*.

[2] `http://curl.haxx.se/`

The URL will trigger the SPARQL servlet and create a DESCRIBE query (as mentioned earlier). Each command will first be written, and after the command a screen shot of the result is shown (from the application gnome-terminal).

```
curl -L -H "Accept: application/rdf+n3"
        http://data.astrazeneca.com/resource/CDISC01/200002/01/11
```

```
eidel@eidel-ubuntu:~$ curl -L -H "Accept: application/rdf+n3" http://data.astrazeneca.com/resource/CDISC01/200002/01/11
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix cdisc:   <http://reference.cdisc.org/ontology#> .
@prefix ncicb:   <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#> .

<http://data.astrazeneca.com/resource/CDISC01/200002/01/11>
      a       owl:Thing , cdisc:HemoglobinLabTest , cdisc:LabTest ;
      cdisc:hasCategory cdisc:Hematology ;
      cdisc:hasClinicalValue
              <http://data.astrazeneca.com/resource/CDISC01/200002/02/11> ;
      cdisc:hasDate "2003-09-19T12:45:00Z"^^xsd:dateTime ;
      cdisc:hasTestPerson <http://data.astrazeneca.com/resource/CDISC01/200002> ;
      cdisc:isFasting "false"^^xsd:boolean ;
      cdisc:isMeasuring ncicb:Hemoglobin , cdisc:Hemoglobin ;
      =       <http://data.astrazeneca.com/resource/CDISC01/200002/01/11> .
eidel@eidel-ubuntu:~$
```

Figure 9: Retrieves code with cURL.

The command returns code in Notation3 syntax. The next command is:

```
curl -L -H "Accept: application/rdf+xml"
        http://data.astrazeneca.com/resource/CDISC01/200002/01/11
```

```
eidel@eidel-ubuntu:~$ curl -L -H "Accept: application/rdf+xml" http://data.astrazeneca.com/resource/CDISC01/200002/01/11
<?xml version="1.0" encoding="utf-8" ?><rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ncicb="http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:cdisc="http://reference.cdisc.org/ontology#" >
  <rdf:Description rdf:about="http://data.astrazeneca.com/resource/CDISC01/200002/01/11">
    <cdisc:hasTestPerson rdf:resource="http://data.astrazeneca.com/resource/CDISC01/200002"/>
    <cdisc:hasDate rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2003-09-19T12:45:00Z</cdisc:hasDate>
    <owl:sameAs rdf:resource="http://data.astrazeneca.com/resource/CDISC01/200002/01/11"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <cdisc:hasCategory rdf:resource="http://reference.cdisc.org/ontology#Hematology"/>
    <rdf:type rdf:resource="http://reference.cdisc.org/ontology#HemoglobinLabTest"/>
    <rdf:type rdf:resource="http://reference.cdisc.org/ontology#LabTest"/>
    <cdisc:isMeasuring rdf:resource="http://reference.cdisc.org/ontology#Hemoglobin"/>
    <cdisc:isFasting rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">false</cdisc:isFasting>
    <cdisc:hasClinicalValue rdf:resource="http://data.astrazeneca.com/resource/CDISC01/200002/02/11"/>
    <cdisc:isMeasuring rdf:resource="http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#Hemoglobin"/>
  </rdf:Description>
</rdf:RDF>
eidel@eidel-ubuntu:~$
```

Figure 10: Retrieves code with cURL.

The command returns code in RDF/XML syntax.

The next command is:

```
curl -L -H "Accept: application/rdf+json"
        http://data.astrazeneca.com/resource/CDISC01/200002/01/11
```

```
eidel@eidel-ubuntu:~$ curl -L -H "Accept: application/rdf+json" http://data.astrazeneca.com/resource/CDISC01/200002/01/11
{
  "http:\/\/data.astrazeneca.com\/resource\/CDISC01\/200002\/01\/11" : {
    "http:\/\/reference.cdisc.org\/ontology#hasTestPerson" : [ { "value" : "http:\/\/data.astrazeneca.com\/resource\/CDIS
C01\/200002", "type" : "uri" } ],
    "http:\/\/reference.cdisc.org\/ontology#isMeasuring" : [
      { "value" : "http:\/\/reference.cdisc.org\/ontology#Hemoglobin", "type" : "uri" },
      { "value" : "http:\/\/ncicb.nci.nih.gov\/xml\/owl\/EVS\/Thesaurus.owl#Hemoglobin", "type" : "uri" }
    ],
    "http:\/\/reference.cdisc.org\/ontology#hasClinicalValue" : [ { "value" : "http:\/\/data.astrazeneca.com\/resource\/C
DISC01\/200002\/02\/11", "type" : "uri" } ],
    "http:\/\/www.w3.org\/1999\/02\/22-rdf-syntax-ns#type" : [
      { "value" : "http:\/\/www.w3.org\/2002\/07\/owl#Thing", "type" : "uri" },
      { "value" : "http:\/\/reference.cdisc.org\/ontology#HemoglobinLabTest", "type" : "uri" },
      { "value" : "http:\/\/reference.cdisc.org\/ontology#LabTest", "type" : "uri" }
    ],
    "http:\/\/reference.cdisc.org\/ontology#isFasting" : [ { "value" : "false", "type" : "literal", "datatype" : "http:\/
\/www.w3.org\/2001\/XMLSchema#boolean" } ],
    "http:\/\/reference.cdisc.org\/ontology#hasCategory" : [ { "value" : "http:\/\/reference.cdisc.org\/ontology#Hematolo
gy", "type" : "uri" } ],
    "http:\/\/www.w3.org\/2002\/07\/owl#sameAs" : [ { "value" : "http:\/\/data.astrazeneca.com\/resource\/CDISC01\/200002
\/01\/11", "type" : "uri" } ],
    "http:\/\/reference.cdisc.org\/ontology#hasDate" : [ { "value" : "2003-09-19T12:45:00Z", "type" : "literal", "datatyp
e" : "http:\/\/www.w3.org\/2001\/XMLSchema#dateTime" } ]
  }
}
eidel@eidel-ubuntu:~$
```

Figure 11: Retrieves code with cURL.

The command returns code in JSON syntax.

7 Discussion

7.1 Analysis of the Results

The findings of the research in the thesis can be hard to understand without any deeper knowledge about the field. The purpose of this section is to analyze how the result are related to the theoretical framework of the thesis. My own reflections will also be a part of this section.

Producing Linked Clinical Data

The extraction of the SAS-dataset produced resources and RDF-triples. The resources got URI's as identifiers based on the values in the SAS-dataset. The first test executed two SPARQL-queries against the new data. The results was two tables with data. The first data contained resources equivalent to the requirements of a HemoglobinLabTest-class. The second query had the same requirements as the first query, but it also added some filters to delimit the results. The results went from eight rows into two with the filters. The filters shows how SPARQL can be used to get specific information from RDF-datasets. However, query languages are nothing new in the field of computer science. The interesting point about the queries is how easy it can be to include data from more than one resource. For instance, both queries started from the HemoglobinLabTest-class and followed their reference to an instance of the class ClinicalValue, to get the data type value. With for example SQL, a solution would involve primary and foreign keys to make this possible. While this example would be rather easy to implement SQL, it still shows how more easily it can be done with semantic web applications.

It also gives an understanding of how the data structure has been changed from the plain container standard with the SAS-datasets into a graph structure. The resources are connected to other objects with help from predicates. And those objects are connected with other objects. Resources can easily connected with new objects by creating new triplets. There is no limitation from the data structure, the resources can be extended and changed in any way. The only limitation would be from an ontology, if the ontology has certain demands to fulfil any requirements for specific classes.

Below this paragraph is a visualization of a Person-resource connected to two HemoglobinLabTests. The circles with text whom starts with `http://` are resources created from the CSV-file used in this study. The figure shows the relationships of each object. The Linked Data Cloud is an illustration of how every node in this graph structure can be interconnected to other objects by following references. In the figure, the Linked Data Cloud is connected to `ncicb:Hemoglobin`, and I chose the node because it is part of a well established ontology. But every object in this graph can be connected to other databases by adopting linked data. Either by creating new triples which links to terms outside the ontology, or by others who links to this graph.

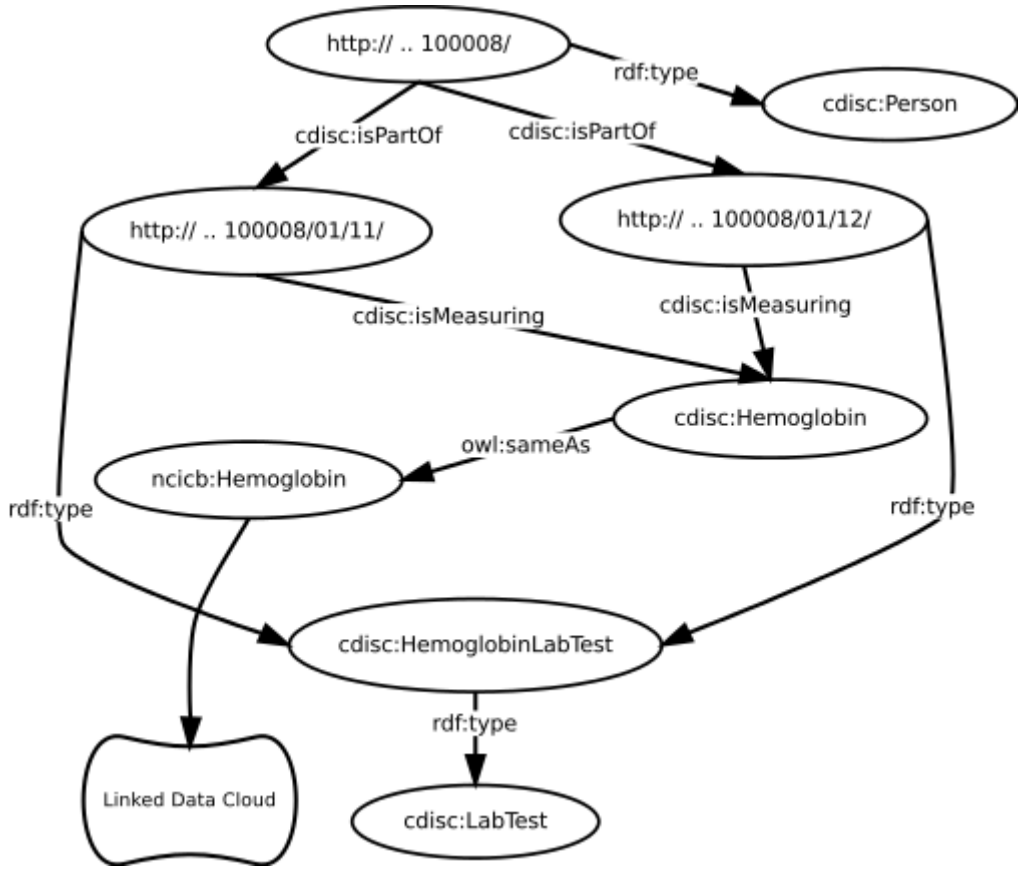

Figure 12: The graph structure of a Person object.

The objects are no longer bound to their dataset.

Linking Data

While the extracting and conversion process of initial data can be seen as an important part of the development of a semantic web system, the linking process is also an essential part of the process. One of the tests was named *Access data through linked data principle*, but what might not be realized is that every test was related to linked data. At some point, in every test, there was a URI to a resource which was expected to be dereferenceable. This shows what an important role linked data has for semantic web applications.

The real question about the results related to linked data is: Was is possible to directly link to resources? Resources were made dereferenceable by the use of the HTTP-protocol. This was tested directly in two tests. The web browser accepted only HTML results and the command line application cURL manipulated the header to force it to accept other syntaxes than HTML.

The results were web pages and RDF-code containing the information related to the resource. The web page also hyperlinked the ObjectProperties, i.e. the properties to other resources. The experiment tested to follow the reference of one resource, to get more information about that object. As the results show, it followed another resource (the ClinicalValue of the test) that was created from the exractment process. However, when the application tried to follow a resource outside the frames of the application, the result give the full document of the OWL-file Thesaurus.owl. The providers of the owl file do not use a SPARQL-DESCRIBE query to give only information of the related object. therefore the whole document is given.

Resources within the application built for this project can be linked to by any semantic web application, but if they are trying to follow more links from our resources, they might not get any results. The Thesaurus.owl file took 13 minutes to download, it is unlikely to think that systems are willing to wait that long to get information.

Semantic web applications needs to improve their systems. The owners of the file Thesaurus.owl did expose the file online, so people can link to it, but they missed one essential part of linked data, the capability to link to individual objects instead of a whole document.

Using Unknown and Foreign Terms

The results from the third test *Access data through terms from unknown and foreign vocabularies* executed queries with foreign terms. The first two queries did not give any results, they were probably invalid. The third asked the server for every object and predicate with a relation to the object ncicb:Hemoglobin. The query returns a table with 14 rows. I find it unlikely that anyone could use this information. To make use of ncicb:Hemoglobin, I suggest that the user uses it with a specified predicate. In this case, the predicate cdisc:isMeasuring is highly related to ncicb:Hemoglobin, since

cdisc:Hemoglobin is set to be equivalent with owl:sameAs. Although, this suggestion assumes that the user has some knowledge of the ontology.

My understandings is that the connection between different, unknown, ontologies is the most complex thing to achieve with semantic web applications. The application created in the study of this thesis is linked into another ontology, but how the linked data works practically is not known.

Increasing the Usefulness of Linked Data

One of the tests manipulated the HTTP header to change the return syntax of the RDF-document. The results showed that clients can choose which syntax they would like the server to return by changing the `Accept` attribute of the HTTP header.

One of the syntaxes retrieved was the JSON. Many web developers are familiar with JSON and the language is fairly easy to understand. However, the return syntax of JSON from the application in this study did return RDF as JSON. While the syntax would most probably be much more understandable than RDF/XML, Notation3 or N-Triples for a web developer, it is still RDF and developers have to learn RDF before they can use the syntax.

The below code shows a draft from the results, it is clearly RDF-based JSON.

```
{
 "http:\/\/data.astrazeneca.com\/resource\/CDISC01\/200002\/01\/11" : {
  "http:\/\/reference.cdisc.org\/ontology#hasTestPerson" : [
  {
   "value" : "http:\/\/data.astrazeneca.com\/resource\/CDISC01\/200002",
   "type" : "uri"
  }
   ],
   "http:\/\/reference.cdisc.org\/ontology#isFasting" : [
   {
    "value" : "false",
    "type" : "literal",
    "datatype" : "http:\/\/www.w3.org\/2001\/XMLSchema#boolean"
   }
   ]
 }
}
```

Personally, I think that this is a poor solution, I'd rather use Notation3 than JSON, I find it more human readable. The code in N3-syntax:

```
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix cdisc:   <http://reference.cdisc.org/ontology#> .

<http://data.astrazeneca.com/resource/CDISC01/200002/01/11>
    cdisc:hasTestPerson
       <http://data.astrazeneca.com/resource/CDISC01/200002> ;
    cdisc:isFasting "false"^^xsd:boolean .
```

The problem with web developer friendly results is that, seemingly they cannot return RDF. To make this possible, a new data structure has to be generated from RDF. A structure without any URI's and no linked data. It really does not sound like a solution that would last long, I mean, is that not exacly why RDF was invented? At this point in time , RDF is not very known by developers, but in 5 years, I believe the data-format will be widely spread among programmers world-wide. Instead of being conservative, it is better to learn the new language.

7.2 Suggestions for Future Work

The analysis of the results observed that the findings supported the theory of the thesis in some ways. However, a few things could have been done better. This section will give suggestions of related work that could not be included in this thesis, mostly due to the timeframe of the thesis.

Using recognized ontologies

Applications familiar with the developed ontology will be able to make use of the application. Nonetheless, in chapter 3, it was proposed that semantic web applications should use as common predicates as possible. Also, common classes can be sub-classed to be customized for the application and still be recognized by applications. The developed ontology for this study uses almost exclusively predicates and classes that were created for this ontology. The classes and predicates are not understandable to a machine with no knowledge of the ontology.

The following paragraph will introduce a few ontologies that can be used when dealing with clinical data in semantic web applications.

CPR (Computer-Based Patient Record)[1]. This is an ontology that can be used to create electronic patient records.

[1] http://code.google.com/p/cpr-ontology/

MUO (Measurement Units Ontology)[2]. This ontology is used to measure values. For example, a measurement of hemoglobin could be an instance of a class in this ontology, with information about the value, which unit, and how it was measured.

OBO Relation ontology[3]. The two previous ontologies mainly provide classes to encapsulate values. This ontology's purpose is to connect objects with predicates. Examples of predicates includes: `has_participant`, `is_a`, `part_of` and `instance_of`. The working field of the ontology is biology.

These ontologies increase the chances that computers will understand the clinical data sets.

XHTML-pages with RDF

The application developed in this study could represent triplets in HTML if the client was a web browser, and if the client was an RDF-application, the server would return RDF/XML (or some other RDF-syntax). There is however another, maybe better solution. It is possible to build XHTML-pages with RDF-syntax by adopting RDFa[22]. One document can therefore be read by both a web browser and RDF-clients.

RDFa is an acronym for RDF - in - attributes, and the name speaks for itself, the data is expressed in attributes in XHTML. I think that RDFa is an important part of the vision of creating a semantic web. It is not likely that every web page owner will set up a full RDF-environment like I have done in the work of this study, but I do think that they are willing to make the little effort to mark up the XHTML-pages they are creating. If they do, their data can be understood by machines just like any RDF.

An interesting fact is that the content management system Drupal has many features available to facilitate for developers to use RDFa.

7.3 Reflections on the Study

I knew that I wanted to make a study about the semantic web when I first began to think about my upcoming bachelor thesis. I had read a lot of the semantic web, and frankly, I found it very confusing to read about the theories behind it. Therefore to learn as much as possible I wanted to set up a working environment. I found the laboratory experiments method to match my needs very well. It allowed me to read a lot about the semantic web software stack and simultaneously elaborate with the programming environments.

[2] `http://idi.fundacionctic.org/muo/muo-vocab.html`
[3] `http://www.obofoundry.org/ro/`

I do not think that more theoretical methods can compete with laboratory experiments when it comes to the implementation of a practical system. Overall, I am satisfied with the results of the study. However, some of the results are not as good as others. Specifically the results from the third test, *Access data through terms from unknown and foreign vocabularies*. Those results could probably been better if I designed the test in another way. I found the other tests to work very well to either support or disapprove the theories.

8 Conclution

The purpose of the study was to answer the research question *How can data be extracted into linked data by using semantic web technologies?*.

I have used the research method of laboratory experiments and created a real world example, based on example data from the organization CDISC provided by AstraZeneca R&D and concluded that data can be converted into a machine readable format by using semantic web technologies (RDF, OWL and SPARQL) and the Java library Jena. However, the data has to use a known vocabulery to be machine readable by a computer.

The study showed how the converted data can easily be interconnected with other RDF-resources, by using triples in a graph structure. This makes the RDF-data schemaless and extendable. The graph structure allows organizations to choose their own data structure, and simultaneously making it readable by another system with a different data structure. SPARQL can be used to perform intelligent queries on a graph database and easily get information of related objects from a given match.

The study also showed how objects in a graph database can be available on the web by using URI's as references and making the URI's dereferenceable. This allows semantic web applications to link directly to an individual object, instead of linking to the full database or document.

Clients whom are requesting objects can change the RDF-syntax by modifying the `Accept` attribute of the HTTP header. In addition, the clients can also choose to get a HTML visualization of an object by only accepting HTML syntax.

The study converted RDF code to JSON, which is a data format which is largely recognized by web developers. The purpose was to facilitate for web developers to use RDF. The JSON format of the study was in fact RDF in JSON-syntax. The syntax is not more developer friendly than any other RDF-syntax. A recommendation to web developers was given, to learn RDF instead of being conservative.

Finally, the study showed the complexity to integrate alien objects to an ontology. The study tried to connect laboratory tests with a foreign ontology to make it readable by more machines. It was pointed out that it was difficult to interpret the results, and they lead to confusion to the user.

Bibliography

[1] L. Yu, *Introduction to the Semantic Web and Semantic Web Services*. Chapman & Hall/CRC.

[2] T. Berners-Lee, "Linked data - design issues." http://www.w3.org/DesignIssues/LinkedData.html, June 2009.

[3] A. Culley, "Technical basis for the semantic web." http://instructionaldesign.com.au/Academic/TechnicalTheme1.htm, June 2006.

[4] W3C, "Rdf - semantic web standards." http://www.w3.org/TR/rdf-concepts/, February 2004.

[5] T. Berners-Lee and J. Hendler, "Scientific publishing on the 'semantic web." http://www.nature.com/nature/debates/e-access/Articles/bernerslee.htm, 2001.

[6] T. Berners-Lee, "Notation3 (n3) a readable rdf syntax." http://www.w3.org/DesignIssues/Notation3.html, 2006.

[7] S. B. Palmer, "The semantic web: An introduction." http://infomesh.net/2001/swintro/, September 2001.

[8] M. K. Smith, C. Welty, and D. L. McGuinness, "Owl web ontology language guide." http://www.w3.org/TR/owl-guide/, 2004.

[9] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data." http://www.springerlink.com/content/rm32474088w54378/, 2007.

[10] DBpedia, "wiki.dbpedia.org : About." http://dbpedia.org/About, January 2011.

[11] J. Grant and D. Beckett, "Rdf test cases." http://www.w3.org/TR/rdf-testcases/#ntriples, February 2004.

[12] M. K. Smith, C. Welt, and D. L. McGuinness, "Owl web ontology language guide." http://www.w3.org/TR/2004/REC-owl-guide-20040210/#SimpleProperties, February 2004.

[13] L. Ding, D. DiFranzo, A. Graves, J. R. Michaelis, X. Li, D. L. McGuinness, and J. Hendler.

[14] A. Doan, J. Madhavan, P. Domingos, and A. Halevy, "Learning to map between ontologies on the semantic web." http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.7937&rep=rep1&type=pdf, 2002.

[15] D. L. McGuinness and P. P. da Silva, "Explaining answers from the semantic web: the inference web approach." http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.93.9550&rep=rep1&type=pdf, 2004.

[16] J. Sherida and J. Tennison, "Linking uk government data." http://events.linkeddata.org/ldow2010/papers/ldow2010_paper14.pdf, 2010.

[17] R. Ejvegård, *Vetenskaplig Metod*, vol. 4:2. Studentlitteratur AB.

[18] W. O. W. Group, "Owl 2 web ontology language document overview." http://www.w3.org/TR/owl2-overview/, 2009.

[19] L. Yu, *A Developer's Guide to the Semantic Web*. Springer-Verlag Berlin Heidelberg 2011.

[20] T. Cornford and S. Smithson, *Project Research in Information Systems: A Student's Guide*, vol. 2. Palgrave Macmillan.

[21] S. Elbassuoni, M. Ramanath, R. Schenkel, and G. Weikum, "Searching rdf graphs with sparql and keywords." ftp://ftp.research.microsoft.com/pub/debull/a10mar/weikum-paper.pdf, 2010.

[22] B. Adida, M. Birbeck, S. McCarron, and S. Pemberton, "Rdfa in xhtml: Syntax and processing." http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014/, 2008.

[23] E. Prud'hommeaux and A. Seaborne, "Sparql query language for rdf." http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/, 2008.