The development and evaluation of the SemanticGeoBrowser

- A Semantic Web application for browsing
the spatial dimension of the Web of Data

Master thesis

By: Lars Berg Hustveit
Supervisor: Csaba Veres

Department of Information Science and Media Studies
University of Bergen

June 1, 2013

Abstract

The Semantic Web is woven together into a Web of data by statements expressed through the Resource Description Framework (RDF) syntax. This syntax only accepts sentences that are shaped by a subject, predicate and object, which is described as a RDF triple. The purpose of creating a RDF triple is to describe a relation between two resources, the subject and object, through the use of a predicate. The syntax enables computers to effectively process RDF data. Plain RDF triples is however not easily read and understood by humans. A common way for humans to browse the Web of data is nevertheless the general web browser, originally designed for browsing the Web of interlinked hypertext documents, created for human consumption. As semantic technologies are being put into practice and the Web of data is growing, the issue of how to browse the Semantic Web has raised on the agenda of the Semantic Web community. The SemanticGeoBrowser is an effort to contribute to the spatial dimension of the Semantic Web. The focus of this design-science research study has been to identify and develop a user-friendly design for browsing geospatial things described in the Web of data. An iterative search and development process has resulted in a proof of concept artifact. This prototype demonstrates a possible solution on how a Semantic Web browser can work. The design artifact was evaluated through a descriptive evaluation method, selected from the design-science knowledge base.

Acknowledgements

The greatest thanks goes to my supervisor Csaba Veres from University of Bergen and colleague Terje Aaberge from Western Norway Research Institute (WNRI). Csaba introduced me to the topic of the Semantic Web with enthusiasm, which lead me to wanting learning more. This master thesis would not have been completed without his many feedbacks. Terje gave me unique insight on the topic through the view of logic and through the Semantic Web community by inviting me to his ISO 15926 and Semantic Technology Conferences. I have learned a lot from both Csaba and Terje, and the many discussions I have had with them have been enlightening and inspiring, so thank you!

I would also like to thank my family and friends for being awesome, inspiring, generous and supportive!

At last, my thanks go to all the other friendly people that have shared their knowledge, experiences and wisdom with me!

Table of Contents

List of figures

Listings

Chapter 1

1 Introduction

Background

Ever since the official proposal for the World Wide Web was introduced in 1990 (Berners-Lee & Cailliau 1990) people have perceived the web browser as the main tool for viewing information resources through the Internet. The web browser was ultimately designed to display information from interlinked hypertext documents, also known as web pages, which constitute the "Web of documents". The information in these documents is generally annotated with HyperText Markup Language (HTML), a markup language that leaves the publisher in full control over how the information is presented to the end-user.

As a result of HTML annotation, the World Wide Web (also referred to as "the Web") developed most rapidly into a medium of documents designed for human consumption (Berners-Lee et al. 2001). While a HTML presentation can enforce the understanding of data amongst humans, this method alone has little impact when it comes to make machines understand the same data. Since machines are not able to understand *real* meaning in human text on its own (Aaberge 2011), machines must instead rely on humans to add additional machine-readable data about the data content, also called metadata. This type of data may possess a *formal* meaning in which machines are able to understand (Aaberge 2011).

The use of machine-readable data, combined with explicit semantics, has over the years extended the World Wide Web with a "Web of data", also referred to as the "Semantic Web" (Berners-Lee et al. 2001), the "Web of machine-readable data" or the "Web of data about things" (Heath 2008). Like the Web of documents, the Web of data is constructed with documents on the Web. Unlike the Web of documents, where links are used to connect hypertext documents into a single global information space, the Web of data uses links to connect any kind of object or concept into a single global data space (Berners-Lee 2006; Heath & Bizer 2011, chap.2.1).

This global data space is based on “Linked Data”, which is the basic idea of applying the general architecture of the Web to the task of sharing structured data on a global scale (Heath & Bizer 2011, chap.2). In order to guide people towards nesting a Web of linked data, a set of best practices for publishing and interlinking structured data on the Web have been published (Berners-Lee 2006; Heath & Bizer 2011, chap.2). The set of best practices, which has become known as the “Linked Data principles” (Heath & Bizer 2011, chap.2), consists of four rules that is to be viewed as expectations of behavior (Berners-Lee 2006).

The first Linked Data principle advocates the use of Uniform Resource Identifiers (URIs), a globally unique identification mechanism, when naming things. An URI comes in form of a compact string of characters and can be used to identify anything from Web documents to real world objects or abstract concepts. Examples of real world objects can be things like people, places or cars, whereas abstract concepts can for example be used to refer to a color, a set of colors or a type of relationship between something (Berners-Lee et al. 1998; Heath & Bizer 2011, chap.2.1; Berners-Lee 2006).

Even though URIs are widely used as identifiers, the Semantic Web community have lately started to replace it with Internationalized Resource Identifiers (IRIs). This change is however minor and is happening because IRIs is a generalization of URIs that allows all characters beyond the US-ASCII charset[1]. Every absolute URI is an IRI. Nevertheless, since the use of IRIs has merely started to be applied by the community, and since most of the literature used in this project refer to the term URI, I will continue to use the term URI throughout this project (Cyganiak 2011; Cyganiak et al. 2012).

The second Linked Data principle advocates the use of HTTP URIs so that people and machines can look up things by their name. Combining globally unique identification with the Hypertext Transfer Protocol (HTTP), which is the Web’s

[1] http://en.wikipedia.org/wiki/ASCII

universal access mechanism, enables identified objects or concepts to be looked up for related data retrieval (Berners-Lee 2006; Heath & Bizer 2011, chap.2.1).

The third Linked Data principle advocates the use of a single data model when publishing structured data on the Web. Publishing data in a standardized content format will make it consumable by a wide range of different applications. While the Web of documents is shaped through the dominant use of HTML, the Web of data is shaped through another standardized format, named the Resource Description Framework (RDF). The RDF data model is a World Wide Web Consortium (W3C) specification for making statements about things in machine-readable form (Berners-Lee 2006; Heath & Bizer 2011, chap.2.1; Heath 2008).

Each statement consists of a *subject*, *predicate* and *object*, and is referred to as a RDF triple. A triple represents the structure of a simple sentence, for example:

"Tim Berners-Lee is creator of WorldWideWeb"

The subject, which is the first part in a triple, is usually the name of a described resource. This name comes in form of an URI and will uniquely identify the resource (as described in the first Linked Data principle) and refer to another RDF dataset with statements that might be useful (like described in the second Linked Data principle). Constructing an RDF triple of the sentence above should therefore contain the person's public URI as a subject, which is:

http://www.w3.org/People/Berners-Lee/card#i

The object, which is the third part in a triple, is often a literal value, like a string, number or date; or the URI of another resource that is somehow related to the subject (Heath & Bizer 2011, chap.2.4.1). Continuing the translation of the example sentence, an URI should also be used as object in order to identify the meaning of the resource. The first web browser ever created was in fact named "WorldWideWeb", typed with no spaces. This name is easily confused with the abstract information space which spelled "World Wide Web" with spaces

(Berners-Lee n.d.). Even though the individual has participated in the creation of both concepts (Berners-Lee & Cailliau 1990), using an URI will make the meaning of the triple's object clear. The following URI, which was already describing the first web browser created, is suitable for reuse:

http://dbpedia.org/resource/WorldWideWeb

The predicate, which is the second part in a triple, indicates what type of relationship exists between the subject and object. A relationship is expressed through the use of an URI that comes from a vocabulary. Vocabularies in Linked Data context are collections of URIs that can be used to represent information about a certain domain (Heath & Bizer 2011, chap.2.4.1). In order to make RDF statements recognizable by a wide range of different applications, reuse of suitable terms from well-known vocabularies are advised (Heath & Bizer 2011, chap.4.4.4). In order to complete the translation of the example sentence, the RDF triple should contain a predicate that equals the meaning of the concept "is creator of". The following URI would be suitable for reuse:

http://purl.org/dc/elements/1.1/creator

This predicate comes from the well-known Dublin Core Metadata Initiative (DCMI) Metadata Terms vocabulary and defines general metadata attributes such as *title* and *date* (Heath & Bizer 2011, chap.4.4.4).

The fourth Linked Data principle advocates the practice of nesting a Web of linked data by including URIs to other resources. This will allow explorers of Linked Data to discover relevant resources when looking up HTTP URIs and prevent published data from becoming hidden data islands, isolated from the rest of the Web (Heath & Bizer 2011, chap.2.1; Heath 2008).

While breaking the rules presented by the Linked Data principles does not destroy anything, ignoring them misses an opportunity to make data interconnected (Berners-Lee 2006). The individuals that do follow them are the

ones nesting a Web of machine-readable data with open standards. By doing so, they are participating in an Open Data Movement that are making it possible for others to re-use structured data in unexpected ways (Berners-Lee 2006).

Even though a lot of RDF triples are published through publically available data-stores, the Web of machine-readable data is not by any means separated from the established Web of hypertext documents. The extension of the Web is rather described as "*another layer of cloth interwoven with the Web as we know it*" (Berners-Lee et al. 2001; Heath 2008). Through the use of RDFa (which stands for Resource Description Framework – in – attributes), RDF triples can be integrated into any hypertext document, making structured data understandable to both human and machine (Heath 2008).

The Semantic Web has however created significant challenges and opportunities for human-computer interaction (Heath 2008; Berners-Lee 2006). Where the traditional web browser has proven to be an excellent tool for presenting HTML content when interacting with the Web of documents, its general design does not appear to be ideal when it comes to browsing the Web of data. Open linked data has moved the Web into a seismic shift where data can be seen in new ways that the original creators might not have anticipated in advance (Heath 2008).

Since the Web of data is based on standardized web architecture and on a single data model, it has become possible to implement generic applications that operate over the complete data space (Heath & Bizer 2011, chap.2.1). This has for example leaded to the development of Linked Data browsers and -search engines. Linked Data browsers are designed for enabling the user to view data from one data source and then follow RDF links within the data to other data sources. The purpose of Linked Data search engines is to crawl the Web of data and index it in order to provide sophisticated query capabilities on top of the complete data space (Heath & Bizer 2011, chap.2.1).

While it seems the development of Linked Data search engines like Sindice.com have been moving on the right path from the start, Heath (2008) points out that

the earliest Semantic Web browsers rather misses the point. The one-page-at-a-time style of browsing, which is well known from the Web of documents, does not take advantage of the potential that lies within integrated views of data assembled from numerous locations (Heath 2008). Karger & Schraefel (2006) argue that simply echoing graphs containing RDF triples have limited value as they are hard for humans to read and does not necessarily solve any of an user's tasks.

The question "*How will we interact with the Web of data?*" (Heath 2008) has been buzzing within the Semantic Web community ever since it's beginning. It first started out as future predictions and visions, but has gradually climbed on the researchers' agenda as semantic web technologies are being put into practice. While the community seems to agree on the Semantic Web browser as a concept, the challenge has rather been to come up with good answers to questions like:

"*What should a Semantic Web browser look like?*" (Heath 2008)

"*How do we elegantly support the range of possible interactions both in pre-defined Semantic Web applications and in dynamic explorations of Semantic Web resources?*" (Karger & Schraefel 2006)

Heath (2008) predicts a shift in the Web's user interaction paradigm where browsers of the Web of data operate on the level of "things", rather than the level of documents. This is because each thing described in a document is of far greater relevance than the documents and the lines of RDF triples themselves. Heath (2008) further suggest this type of applications are named "thing browsers" where things like people, places and other concepts are treated as first-class citizens of the interface. It would be the machines' job to assemble this data into a coherent view (a view that includes all the data the user expects it to) that is ready for human consumption (Heath 2008). Heath (2008) thinks the use of look-up services such as Sindice is a success factor in the development of Semantic Web browsers. This is because semantic web indexes are able to provide quick and advanced query capabilities. A single query could result in

different RDF documents from several data sources mentioning a particular URI of a thing (Heath 2008).

Web of data interaction is a general problem where much more innovative work is possible and needs to be done (Heath 2008; Karger & Schraefel 2006). In this project, the effort is focused on the construction of a proof of concept artifact that aims to demonstrate possible solutions on how a Semantic Web browser can be used as a tool for interacting with the Web of data. The construct will pursue the concept of a "thing browser", which will place the things in the center of the user interface, rather than raw RDF triples from documents. The thing-oriented artifact will be combined with a semantic web index, which will provide more advanced query capabilities than single data sources can provide. In order to limit the scope, the artifact will have a user-targeted interface that are focusing on geospatial things and designed for users within a selected domain.

Motivation

My motivation for choosing to conduct research within the area of the Semantic Web was firstly based on my own enthusiasm for the Web. Through a master course at the University of Bergen, INFO310 - Advanced Topics in Information Systems, I noticed an opportunistic enthusiasm from people within the Semantic Web community. The introduction to this opportunistic vibe made me curious to continue exploring this area further. At the same time, I was eager to develop my skills in the art of web programming.

Research question

The following research question is the focus of this study:

How can we build a user-friendly Semantic Web browser that enables its users to discover and explore geospatial things described in the Web of data?

In order to answer this research question, a demonstrator will be constructed as a proof of concept artifact. The development process will be conducted through an iterative process. The progress and level of success will be measured

throughout the project by conducting an evaluation at the end of each iteration. These evaluations will include a measurement against the artifact's system requirements.

Hypothesis

As pointed out by the Semantic Web community, there is a need for Semantic Web browsers that will make it easier for humans to interact with the things described in the Web of data. In order to support the research question, I would like to propose the following hypothesis:

A thing browser, like the SemanticGeoBrowser, will make it easier for humans to discover and explore geospatial things described in the Web of data.

System requirements

In the planning of the SemanticGeoBrowser, a set of system requirements was formulated. These requirements represent my opinion on what is to be expected of the proof of concept artifact. My points of view have been formed through the reading of literature from the Semantic Web community, my many discussions with supervisor Csaba Veres, and colleague Terje Aaberge. These are my proposals for the system requirements:

The SemanticGeoBrowser should

1. operate at the level of "things" (instead of at the level of documents) and treat them as first-class citizens in an user-friendly interface. *This requirement is based on Heath (2008).*
2. contain an interactive map of the planet Earth, which enables the user to
    a. explore the Web of data by selecting an area of interest.
    b. interact with the "things" discovered on the Web of data.
3. be knowledge-based in order to
    a. help the user search for relevant things.
    b. help the user recognize things that are relevant to the domain of operation.
    c. present facts about relevant things in a user-friendly way.

4. make use of a semantic web index look-up service that provide
    a. access to a large amount of RDF datasets from the Web of data.
    b. advanced geospatial query capabilities to be made within a selected area of interest.

    This requirement is based on Heath (2008).
5. avoid solutions that would trigger the web browser to reload a lot.
6. be able to assemble and handle RDF data seamlessly behind the scenes.

    This requirement is based on Heath (2008).
7. be able to draw conclusions from facts described in the properties of things.

    This requirement is based on Heath (2008).
8. support different data sources and apply knowledge from an external ontology.
9. help the user to discover patterns shaped by the coordinates of geospatial things.
10. allow users to conduct text searches when available thing characterizations aren't enough.

These system requirements will be addressed in the construction phase of the artifact, which starts in the third iteration and described in chapter 4.

Chapter 2

2 Literature

This chapter will present technologies and data sources that have been used in this project. The literature used in this study is listed in the reference list.

Technology

This sub-chapter will present technologies and data sources that have been used in this project.

Programming Languages, APIs and Frameworks

The proof of concept artifact has been constructed through the use of the following technologies: Dojo Toolkit 1.5 and 1.6, Google Maps JavaScript API v3, EyeServer, Sindice APIs, HTML, CSS, JavaScript, jQuery, PHP, SPARQL, Lucene Query Syntax, RDF, and RDFa.

Sindice – The Semantic Web Index

The Sindice platform, available at Sindice.com, present itself as "The Semantic Web Index" and is a lookup service over resources crawled on the Semantic Web (Tummarello et al. 2007). While a lot of the semantic data is collected from web documents, their crawlers also support SPARQL endpoints. Their crawlers support formats like RDF, RDFa, Microformats and Microdata, and it is possible to add data to their index by notifying the service where to find new data to crawl. By offering advanced search and querying services, through their web pages and specialized APIs, they are encouraging software developers to build applications on top of their collected data (Anon 2013a). Sindice offer by this a counterbalance to the decentered publication model of the Semantic Web and make it possible for developers to build rich Semantic Web applications with little effort (Tummarello et al. 2007; Hausenblas 2009).

The infrastructure of Sindice is based on Lucene[2], a free and open source information retrieval software library. Lucene is however not built to handle

[2] http://lucene.apache.org/

large semi-structured document collections. Sindice have therefore built SIREn[3] (Semantic Information Retrieval Engine), a Lucene plugin developed to efficiently index and query RDF. SIREn is released under the GNU Affero General Public License, version 3 open source license and encourage by this people to implement their solution when approaching the Web of Data (Anon 2013f).

OpenStreetMap and LinkedGeoData

OpenStreetMap presents itself as "*an effort to add a spatial dimension to the Web of Data / Semantic Web. LinkedGeoData uses the information collected by the OpenStreetMap project and makes it available as an RDF knowledge base according to the Linked Data principles*" (Stadler 2012).

The Norwegian Mapping Authority

The Norwegian Mapping Authority (NMA), in Norwegian also known as "Statens Kartverk" or "Kartverket", is a public agency under the Ministry of the Environment and describes themselves as "*the national provider and administrator of geodesy, geographical and cadastre information covering Norwegian land, coastal and territorial waters*" (Andersen 2009). The public agency was founded in 1773 and have since then been working on the many tasks of building and maintaining the Norwegian Spatial Data Infrastructure. This makes Statens Kartverk the most important data source when it comes to geographical information about Norway.

Even though the Norwegian government is financing a large amount of the public agency´s yearly budget, Statens Kartverk has a long tradition of keeping their information silos closed to Norwegian taxpayers and other businesses. As the government does not cover all the expenses, the agency argues they have to cover their expenses by other means. Statens Kartverk is therefore practicing the selling of geospatial data through map products and other services (Engeland 2012). The income generated by this practice was in 2011 on 138 million Norwegian kroner and is covering approximately 14 percentage of the public agency's budget (Brombach 2012).

[3] http://siren.sindice.com/

As of 2009, Statens Kartverk started to offer public access to their map data, free of charge. The new service was made available through their own API, allowing web applications to communicate with their servers. With this, Statens Kartverk states that developers should come up with creative solutions on how to use their map data. However, the data is still being kept on a short leach as their user agreement restricts the usage to individual people, associations, applications that are not generating any form of income, and the number of daily requests is heavily limited (Amundsen 2009b).

Since 2009, Statens Kartverk has continued to release map data, free of charge (Engeland 2012). Critics have argued that their service usage policy and API limitations are restricting innovation. Statens Kartverk is also criticized for giving microscopic releases of open data compared to the large amount of raw data the public agency are sitting on (Amundsen 2009a; Brombach 2012).

Chapter 3

3 Method

This chapter will introduce the literature of the research method that has been used to execute this project. It will also describe how the methods presented in the literature have been used to conduct the research.

Design-science research

This project has been executed as a "design-science research". In order to conduct a successful design-science research, this project has been using elements from the framework and following the guidelines proposed in the research essay "Design Science in Information Systems Research" by Hevner et al. (2004). Because of the authors primary goal to "*inform the community of IS researchers and practitioners of how to conduct, evaluate, and present design-science research*", and how they do this by "*describing the boundaries of design science within the IS discipline via a conceptual framework for understanding information systems research and by developing a set of guidelines for conducting and evaluating good design-science research*", the research essay has proven to be a good guide to understand the process of the selected research method. Figure 1 shows how Hevner et al. (2004, p.80) illustrates the conceptual framework of design-science.

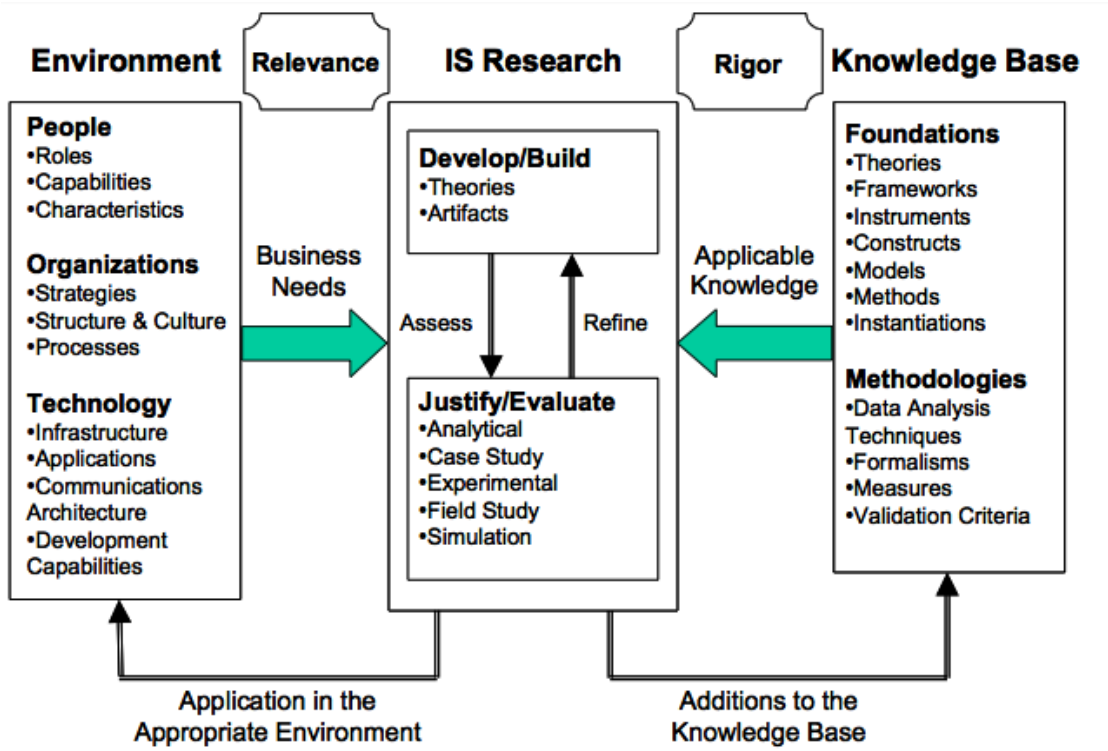


Figure 1: Information Systems Research Framework

Research guidelines

This section will introduce the seven guidelines that (Hevner et al. 2004) has established to assist researchers and others to "*understand the requirements for effective design-science research*".

Guideline 1: Design as an Artifact

"*Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.*"

In the first guideline Hevner et al. (2004, p.82) points out that the process of design-science research must result in a purposeful IT artifact within an appropriate domain. Instantiations, constructs, models and methods, can all be defined as IT artifacts, and their capabilities are all equally crucial in the development and use of information systems. Hevner et al. (2004, p.83) also points out that "*artifacts constructed in design-science research are rarely full-grown information systems that are used in practice*". Instead, artifacts should be

innovative by defining new ideas, practices, or technical capabilities (Denning 1997; Tsichritzis & Metcalfe 1998) cited by Hevner et al. (2004, p.83).

Guideline 2: Problem Relevance

"*The objective of design-science research is to develop technology-based solutions to important and relevant business problems.*"

In the second guideline, Hevner et al. (2004, p.85) explains that efforts to solve problems in design-science research should be done with respect to a constituent community. The problem should therefore be *real* and relevant to the community. A good indication on this is when people within the community have addressed the problem.

Guideline 3: Design Evaluation

"*The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.*"

In the third guideline, Hevner et al. (2004, p.85) emphasize the importance of evaluation as a crucial component of the research process. "*Because design is inherently an iterative and incremental activity, the evaluation phase provides essential feedback to the construction phase as to the quality of the design process and the design product under development. A design artifact is complete and effective when it satisfies the requirements and constraints of the problem it was meant to solve*".

Guideline 4: Research Contributions

"*Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.*"

In the fourth guideline, Hevner et al. (2004, p.87) introduce three types of research contributions and explains that any design-research project must contain one or more of these contributions. The first type of contribution, "The

Design Artifact", is the artifact itself and "*must enable the solution of heretofore unsolved problems*" or "*apply existing knowledge in new and innovative ways*". The artifact may also "*extend the knowledge base*" in the conceptual framework of design-science. This knowledge base is illustrated in Figure 1. The second type of contribution, "Foundations", is the "*the creative development of novel, appropriately evaluated constructs, models, methods, or instantiations that extend and improve the existing foundations in the design-science knowledge base*". The third type of contribution, "Methodologies", is any creative development and/or use of evaluation methods that can be applied by others in design-science research.

Guideline 5: Research Rigor

"*Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.*"

In the fifth guideline, Hevner et al. (2004, p.87) argue that methods used in design-science research must be both rigorous and relevant. Researchers should therefore be extremely thorough by using the theoretical foundations and research methodologies that are found in the knowledge base of design-science. "*Success is predicated on the researcher's skilled selection of appropriate techniques to develop or construct a theory or artifact and the selection of appropriate means to justify the theory or evaluate the artifact*".

Guideline 6: Design as a Search Process

"*The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.*"

In the sixth guideline, Hevner et al. (2004, p.88) argue that it is often hard to find the best, or optimal, design for realistic information systems problems. Because creation of design essentially is "*a search process to discover an effective solution to a problem*", the design process should be iterative. The iterations can be conducted by repeating the process presented in the "Generate/Test Cycle",

which is illustrated in Figure 2 by Simon (1996) and cited by Hevner et al. (2004, p.88).

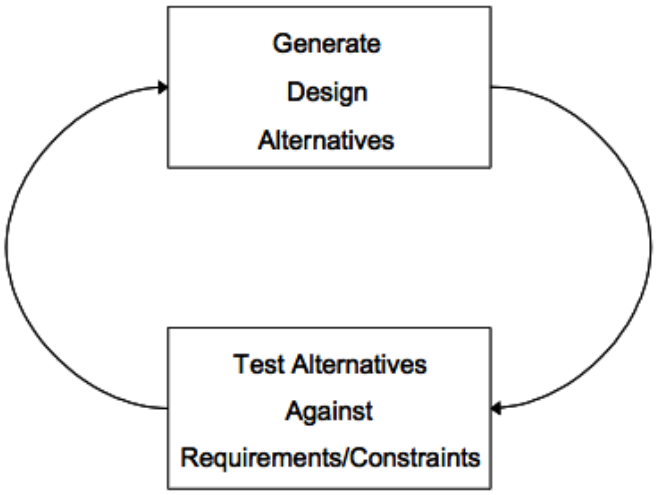


Figure 2: Generate/Test Cycle

In order to find an effective solution to a problem, Hevner et al. (2004, p.88) introduces three factors of problem solving by Simon (1996). The factors should be repeated in the Generate/Test Cycle and are cited and explained by Hevner et al. (2004, p.88) like this: "*Means are the set of actions and resources available to construct a solution. Ends represent goals and constraints on the solution. Laws are uncontrollable forces in the environment.*" By repeating relevant means, ends and laws iteratively, progress will be made as the scope of the design problem is expanding. The factors will be refined for each repetition in the process, while the design artifact itself will become more relevant and valuable.

Guideline 7: Communication of Research

"*Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.*"

In the seventh guideline, Hevner et al. (2004, p.90) suggest that technology-oriented audiences are provided with "*sufficient detail to enable the described artifact to be constructed (implemented) and used within an appropriate organizational context*". This should enable "*practitioners to take advantage of the benefits offered by the artifact*" and allow "*researchers to build a cumulative*

knowledge base for further extension and evaluation". Management-oriented audiences should also be provided with sufficient details to understand the problem and the benefits acquired by constructing or using the artifact within an organizational context.

Limitations

In order to limit the scope, the concept artifact will be designed for browsing geospatial things described in RDF data.

Chapter 4

4 Development

This chapter will describe the process of developing the SemanticGeoBrowser, by presenting the work done in each of the project's six iterations. Each iteration is represented by a subchapter and is evaluated in the end.

First Iteration

The first iteration consisted of the following tasks that would get the research project started:

1. Find datasets containing data about geospatial things, preferably data in the form of RDF and in the local area of Hordaland, the county of University of Bergen.

2. Identify and get familiar with technologies that would be good choices for the development of the proof of concept artifact.

Finding geospatial data sources

Task one: Find datasets containing data about geospatial things, preferably data in the form of RDF and in the local area of Hordaland, the county of University of Bergen.

The purpose of this task is to find and explore RDF data that can be used as data source for the Semantic Web Browser.

The search was conducted in the Web of documents, using one of the many search engines available. A lot of different web pages were found which provided RDF datasets by linking to data files for download, but also by providing access to data stores through SPARQL endpoints, which gives people and machines querying capabilities.

At first the plan was to add interesting findings to a data store and make the RDF datasets accessible to the proof of concept application through an SPARQL endpoint. A lot of time in the beginning of this iteration was therefore used on downloading RDF datasets. However, this process was stopped when it came to my attention what opportunities the Sindice platform was providing.

The Sindice APIs

The Sindice platform was selected as the first data source for the SemanticGeoBrowser. One of the reasons is because of their enormous collection of geospatial data, accessible through one platform, free of charge. By continuously indexing this growing data collection, the Sindice platform provides an overview representing the Semantic Web. It is this overview that opens up the possibility for the SemanticGeoBrowser to query the entire Web of data.

Even though a large amount of the distributed and machine-readable data on the Web of data, are linked together, searching the Semantic Web without the support of an index platform, like Sindice, would not be feasible. The platform also provides access to information islands; resources that are not linked together with other discovered datasets. Without the support of a search index the SemanticGeoBrowser would only be able to view selected information resources.

For an application to conduct a search in the semantic web index of Sindice, a search query, containing a query object, is sent through their "Search API Version 3" as a HTTP request. The Search API has a wide aspect of supported parameters that can be used to construct the query object. An overview over these parameters is listed in their Search API documentation[4].

The simplest form of search query can be made using the **q** parameter. Queries containing the **q** parameter are called a "keyword query". According to the Sindice documentation this parameter allows the user to find "*all the relevant*

[4] http://sindice.com/developers/searchapiv3

documents that contain either a keyword or a URI using full-text search syntax". Here is an example of a search query using the **q** parameter:

http://api.sindice.com/v3/search?q=hotel

This search query asks for all documents containing the word "hotel" in the semantic web index of Sindice.

The Search API supports the result formats JSON, RDF/XML and ATOM. While the search query in the previous example would return the result in ATOM, including a preferred format in the **format** parameter will override this default setting, like this:

http://api.sindice.com/v3/search?q=hotel&format=json

In the time of writing, searching the web page version of Sindice for all documents containing the word "hotel" gave a result of 7,865,288 documents. However, querying the Sindice Search API for the same word will not return the same amount since the result is limited into 100 result pages. Each result page will contain up to ten documents. Which result page returned is controlled by the **page** parameter. The proof of concept artifact will therefore have to send up to 100 HTTP requests in order to fetch as many items as possible. The next example shows how one of the many search queries will look like when an application fetches items from a large search result:

http://api.sindice.com/v3/search?q=hotel&page=38&format=json

Querying for patterns in RDF triples is done by using the **nq** parameter. The Search API documentation (Anon 2013e) explains that any query containing the **nq** parameter is called an "Ntriple query", and are used to "*produce precise search results using simple, but powerful triple patterns to represent partial or complete triples*". A triple pattern is a complete or partial representation of a triple, which consists of a subject, predicate and object. In order to create a

partial representation of a triple, the wildcard symbol ***** is included to substitute any part of the triple. The **nq** parameter will allow the SemanticGeoBrowser to search for things described with specific properties.

An Ntriple query, requesting things, described as a type of hotel, using the URI http://schema.org/Hotel, could be constructed like this:

1. http://api.sindice.com/v3/search?nq=
2. * *(Subject)*
3. *(White space)*
4. http://www.w3.org/1999/02/22-rdf-syntax-ns#type *(Predicate)*
5. *(White space)*
6. http://schema.org/Hotel *(Object)*
7. &format=json

Several triple patterns can be included in one Ntriple query by combining the patterns with the boolean operators **AND**, **OR** and **NOT**.

Another reason for selecting Sindice as a data source is because of the Search API´s support for limiting a search by the use of geographic coordinates. This makes it possible to generate queries that will only look for "things" within a selected area of interest. In order to generate such a query, two geographical coordinates, each described with latitude and longitude, are needed as input. By requesting the *south west* and *north east* coordinates from a map feature, an area of interest could be defined to be within the rectangle view of a map. Figure 3 illustrates how an area of interest can be selected in a map feature through the use of Google Maps JavaScript API v3.

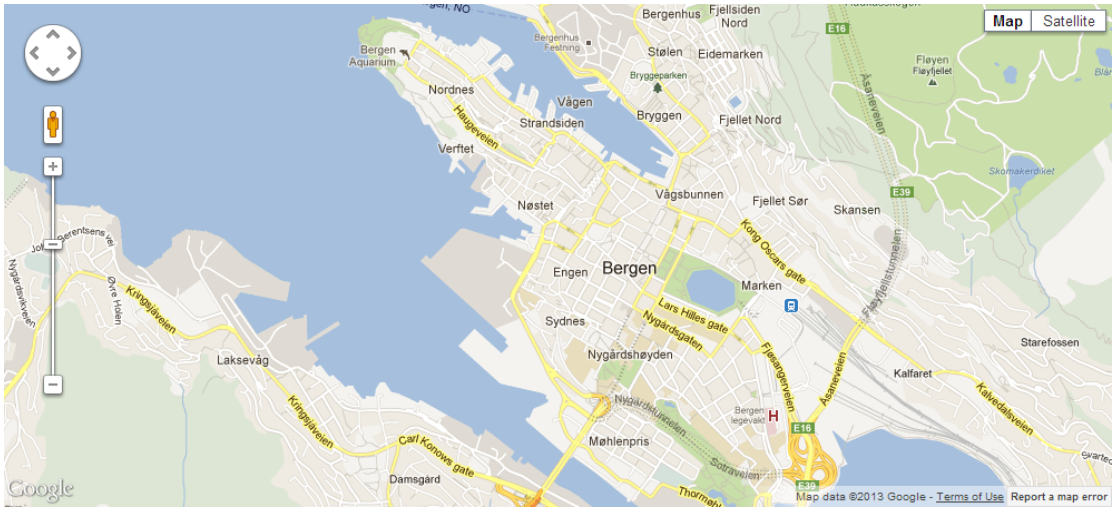


Figure 3: A map feature showing the city of Bergen

As an example, the geographical coordinates from Figure 3 is as follows:

- SOUTH WEST (60.38216815444581, 5.2740525357666)
- NORTH EAST (60.4013357170463, 5.35945407080078)

These coordinates is used in the next example, which is a query asking for all documents containing geospatial data within a square border defined by the two geo locations:

1. http://api.sindice.com/v3/search?q=
2. (geo:lat [60.38216815444581 TO 60.4013357170463])
3. *(White space)* AND *(White space)*
4. (geo:long [5.2740525357666 TO 5.35945407080078])
5. &format=json
6. &page=1

Line two first requires the latitude value from the south west corner of the map, and then the same from the north east corner. Line four requires the same, but using the longitude values.

Since the infrastructure of Sindice is based on Lucene, the queries used in the Sindice APIs can be considered as Lucene queries.

Finding appropriate technology

Task two: Identify and get familiar with technologies that would be good choices for the development of the proof of concept artifact.

The SemanticGeoBrowser has been planned as a web application from the early stages of this research project. It was therefore already decided that the front-end part of the demonstrator should be developed in HTML 5 and CSS, in combination with map features from Google Maps JavaScript API v3[5]. While these front-end technologies were easy to choose because of my experiences from other projects, it was in the start not so obvious to me what back-end technologies that were the best choose for requesting and handling data from third party services. In order to identify what back-end technologies to use, three technologies were considered.

The first technology considered was to write most of the code in PHP 5, a server-side scripting language that is common to use when developing dynamic Web pages. Even though I have much experience with this language and have earlier used it in scripts that request and handle data from SPARQL endpoints, PHP was not considered as the best choice for this project. The conclusion was made on the fact that PHP is a server-side language, and I assumed this would trigger the web browser to reload the web application a lot. Avoiding the one-page-at-a-time style of browsing, triggered by reloading the web browser a lot, is one of the requirements created for the demonstrator.

Even though the next technology considered is running on the server-side as well, the programming language Java was also considered because of its ability to run on different platforms without having to recompile the source code. The code produced could in this way have been reused in an Android application on a later date. The server-side framework, Play Framework 1.2, was also considered because of its attempt on making Java web application development easier. This solution was considered and tested for a week, but was for similar reasons as PHP not selected as a solution. Using complex server-side solutions for a proof of

[5] https://developers.google.com/maps/documentation/javascript/

concept application with mostly client-side tasks was at this stage considered as unnecessary time consuming.

After reviewing possible solutions in PHP and Java, in combination with the map solution from Google Maps JavaScript API v3, I learned that most of the functionality could be done on the client-side, using the scripting language JavaScript. Since JavaScript runs on the client-side it will allow tasks to be done without triggering the browser to reload. JavaScript is also the perfect match for integrating map functionality into the web application because Google Maps JavaScript API v3 is based on the language.

Dojo Toolkit 1.5 was selected as the main framework to support the development in JavaScript. There were several reasons that this framework was selected. Firstly, the framework has features for sending and handling HTTP requests cross-domain. Communicating with servers that comes from other domains than the original host in JavaScript presents a high security risk. This is because JavaScript execute code on the client-side, leaving the client vulnerable. Web browsers have therefor implemented different security measures to secure the use of JavaScript. Because the demonstrator need to communicate with the API's of Sindice, supporting functionality for cross-domain communication is therefore necessary. Secondly, the framework has its own data store. Storing and retrieving data fast on the client-side will be necessary when handling results from the Sindice APIs. Thirdly, the framework has features for creating a user interface. Since the demonstrator will need a user-friendly user interface, features that could improve the user experience are considered as useful. Fourthly, a web application provided by the consulting firm Computas demonstrated some techniques on how to use the framework to send and handle request from a SPARQL endpoint. The ability to study their source code gave me a good idea on how the SemanticGeoBrowser could be constructed.

Evaluation

The first iteration was a long and educational process for me. Even though I was confident about the research question and its relevancy to the Semantic Web community, I had some concerns when it came to approaching and solving the problem. Since I did not have a clear idea about what RDF datasets I was going to base the proof of concept artifact upon, nor what technology I was going to use to handle it with behind the scenes, I started out by researching these aspects. At the same time I was also reading scientific literature on the topic. Using the idea of the SemanticGeoBrowser as a vision, different pieces of relevant information gradually were discovered and became apparent.

While the first task started out by gathering relevant RDF datasets that would be interesting to browse in a Semantic Web browser, this approach suddenly became irrelevant upon the discovery of the Sindice platform. When I saw what kind of features and number of gathered RDF triples the lookup service could provide, it became clear that the SemanticGeoBrowser should be based on this platform. After reviewing geospatial things in the semantic web index I was however disappointed over the lack of additional properties that would characterize the individuals with facts. This lead to the decision of adding richer data to the semantic web index in the next iteration. Even though the downloaded RDF datasets became irrelevant to this project, the process of finding them lead me to discover a suitable data source for supplementing with the Sindice data.

The second task of finding suitable technology to include in the SemanticGeoBrowser also started a bit of course by looking into Java technology. But after reviewing a demonstrator constructed by Computas, it became apparent that JavaScript technology was the best and fastest choice. Since I did not have much experience with this scripting language, learning JavaScript by studying their source code was extremely helpful. It also gave me the

opportunity to discover and learn techniques provided by the Dojo Toolkit framework[6].

In summary, this iteration got off to a bumpy start, but turned out to be a successful one. Building more knowledge and discovering suitable technologies were necessary for the project to move forward. Even though I am disappointed over the data quality of the scraped data in Sindice, it does not matter as suitable data can be added to the semantic web index later. The important thing is that it seems that the technology discovered is significant to construct the proof of concept artifact envisioned.

Second Iteration

In the second iteration, the focus is on finding richer data about geospatial things that could be supplemented to the data source of Sindice. After reviewing Sindice data in the first iteration, it became clear that just a few triples in each RDF dataset were property facts about things. The rest were mostly data about other data, for example metadata about the web page where the RDF data were fetched from. Because there is only so much that can be done with a geospatial thing without having interesting property facts, it became clear that the planned web application would need more interesting data to work with. I therefor started looking for data about things with more property facts.

Lifting legacy data from BT.no Sprek

In the search for geospatial data that would be interesting to browse in the SemanticGeoBrowser, the web service BT.no Sprek[7], was discovered. The service enables their users to share information about foot hikes in the local county of Hordaland, Norway. Data about foot hikes would be interesting to browse because it would contain a lot of different property facts. Foot hikes are also popular within the tourist domain and has the potential of providing some good user scenarios with examples of how a semantic geo browser could be used. The organization behind Sprek was therefore contacted and they agreed to provide hike data for this project.

[6] http://dojotoolkit.org/
[7] http://tur.bt.no/

The data from Sprek were provided in form of an Extensible Markup Language (XML)[8] file, dumped from their MySQL database. XML is a markup language that makes it easy to share structured data between information systems over the Internet (Anon 2013g). It is a good format to receive legacy data in because it can easily be converted with Extensible Stylesheet Language Transformations (XSLT)[9]. XSLT is a language for transforming XML documents into any other type of documents (Anon 2013h).

The legacy data from Sprek was lifted in a two-step process. The reason for this was my participation in another research project, Semantic Sognefjord[10], led by Western Norway Research Institute (WNRI). The aim of the Semantic Sognefjord project was to explore what benefits the local tourism industry could gain by combining semantic- and other open technologies (Aaberge 2012b, p.3). Since both projects were in need of the same type of data, it was decided that the lifting process could benefit both projects. The Semantic Sognefjord project was however experimenting with a new modeling methodology to structure things in RDF with. The first step therefore resulted in an alternative data structure that was more complex than needed for this master thesis project. While the first step captured all the relevant data needed from the XML source, the second step restructured Sprek data from the RDF triples produced in the first step. The lifting process, in both steps, is described below.

The first step in the lifting process consisted of lifting XML data, by writing XSLT code. This code constitutes a XSLT style sheet and describes a set of template rules on how the XML data is going to be used to construct a result document. In order to generate several output documents, it was necessary to write the code using XSLT 2.0. The selection of a XSLT processor fell on Saxon[11] (Home Edition) Version 9.3 because of its support of XSLT 2.0. The outcome documents generated were in form of RDF/TURTLE. The RDF triples described in these

[8] http://www.w3.org/TR/REC-xml/
[9] http://www.w3.org/standards/xml/transformation
[10] http://www.vestforsk.no/rapport/semantisk-sognefjord.no
[11] http://saxon.sourceforge.net/

documents were added to a data store with a SPARQL endpoint. This enables anyone to access the newly lifted data through the use of SPARQL queries.

Lifting legacy data into RDF statements requires the use of vocabulary terms. This is decided by the RDF syntax. The syntax decides what are to be accepted as well formed sentences. In RDF, the syntax only accept sentences in which are shaped by a subject, predicate and object. The purpose of this is to ensure that RDF triples can carry meaning. The semantics is a theory on how meaning of words is tied to external objects and activities. In order to get a formal meaning into every RDF triple, vocabulary terms is used as predicate to describe the relationship between the subject and object. Vocabularies are collections of terms, identified by HTTP URIs, which can be used to represent information about a certain domain (Heath & Bizer 2011, chap.2.4.1). Since the meaning of a sentence is determined by the meaning of the words composing it, it is important to be thoughtful in the process of select terms in the construction of new RDF triples. Sentences that are not well formed are meaningless.

A vocabulary term is however not meaningful in itself. The formal meaning of vocabulary terms is defined by ontologies. An ontology is an explicit specification of a conceptualization (Gruber 1993). A conceptualization is an abstract, simplified view of the world (Gruber 1993). Its purpose is to represent objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold them (Genesereth & Nilsson 1987) cited by (Gruber 1993). Every knowledge base is committed to some conceptualization, explicitly or implicitly (Gruber 1993).

In the process of creating RDF triples, reuse of suitable terms from well-known vocabularies are advised. In this way, existing terms do not have to be reinvented and it rises the probability that data can be consumed by applications that may be tuned to well-known vocabularies, without requiring further pre-processing of the data or modification of the application (Heath & Bizer 2011, chap.4.4.4). Similar terms from different vocabularies may however have

different meanings. It is therefore important to select terms intended for the domain of operation.

If there is no suitable term to use within the domain of operation, one must create it in a new ontology. Since I could not find any ontologies created within the domain of hiking, I decided to create one. An ontology for an object language is in addition of a non-logical vocabulary supplemented by a set of *extensional and intensional definitions*, and *axioms* (Aaberge 2011).

An extensional definition of a predicate is essentially *a list of the names* (or pairs of names) of the individuals that constitute its extension. When the names are denoting identifiable individuals of the domain, the extension of the predicate representing its meaning is given (Aaberge 2012a). All predicates thus possess extensional definitions.

An intensional definition states *the properties an individual must possess* for the predicate to apply (Aaberge 2012a). While it is possible to describe the properties of for example a hotel, it is not possible to describe what a color is or ten kilos through intensional definitions.

An axiom is an implicit definition that relates the primary terms of the vocabulary (Aaberge 2011). When axioms are defined through logical statements they are assumed to be *true*. The truth presented can thereby be used as a fact to support other (theory and domain dependent) truths. This makes axioms the foundational ingredient for reasoning to take place (Anon 2013b). A common example on axioms is to describe family relations. For example a father's brother is an uncle.

In the process of creating a new ontology, I decided to accomplish the following tasks, which are described as an ideal method for ontology construction by Aaberge (2011):

1. delimit the domain of discourse

2. identify a primary vocabulary
3. establish the axioms
4. introduce secondary terms by intensional definitions
5. introduce further secondary terms by extensional definitions

The second step in the lifting process consisted of lifting data for this project. In order to change the structure of the lifted hike data from step one and at the same time supplement it with geospatial data from the Norwegian Mapping Authority, I decided to run the lifting process through a series of five scripts based on the scripting languages PHP and JavaScript. This process is described in the next subchapter.

Supplementing geospatial data from the Norwegian Mapping Authority

Even though the Sprek data source provided hike paths and geo locations, which can easily be illustrated on a map, this type of property facts contains a greater potential when it comes to finding more characterizations of foot hikes. Since the hike paths and geo locations are located within the borders of Norway, the existing hike data can be supplemented with data from Statens Kartverk. This will extend the amount of property facts about each foot hike, which will result in more detailed RDF data to use in the SemanticGeoBrowser.

The Norwegian Mapping Authority provides four types of services through their Web Processing Service (WPS). These are named “elevation”, “elevation Chart”, “elevation JSON” and “elevation XML”. The services are based on open standards supported by the international voluntary consensus standards organization Open Geospatial Consortium (OGC) (Hirsch 2011).

The way in which these services work is by sending a HTTP request to the Mapping Authority’s WPS server. The HTTP request must contain the path to the WPS server, the selected service, and required parameters.

The “elevation” service requires a single geographical coordinate as input and returns XML data about the height, terrain information and place name for the geo point (Hirsch 2011).

The “elevation Chart” service requires a URL to the path of a GPX file as input. GPX stands for “GPS eXchange Format” and must contain the geographical path of a single hike. A successful request to this service will result in a link to a generated PNG picture. The picture should contain the terrain profile of the hike as a chart (Hirsch 2011).

The “elevation JSON” and “elevation XML” services requires the same input as the “elevation Chart” service. The difference between these three services is the output. Whereas the “elevation Chart” service illustrate the data as a profile in a picture, the “elevation JSON” and “elevation XML” services returns the same data as text, formatted in JSON and XML (Hirsch 2011).

After exploring the possibilities of the Norwegian Mapping Authority’s WPS services, these two tasks were planned:

- Task one: Generate extra property facts about the hikes.
- Task two: Generate a visual profile about each hike.

In order to conduct these two tasks, a series of scripts were made to request and fetch data, thereafter processing it into usable RDF triples. The process is explained below.

The outcome of task one should be to have more property facts about the existing hikes then we got from tur.bt.no. These facts should say something more about a path than the current length and approximately duration property does. Here are some questions that will provide informative property facts if they are based on data from the Mapping Authority:

- What is the hike’s lowest and highest elevation point above sea level?

- What is the difference between the hike's lowest and highest elevation point?
- From the start to the end of a hike, how many meters of the path is uphill, and how many are downhill?

The scripts constructed for task one were written with these questions in mind. Here is a presentation of the five scripts that were made to solve both tasks:

Script 1 (Written in JavaScript and PHP)
The purpose of the first script written is to download data about all the geo points in a hike path. These data is going to be used in the script 2, which is going to find the answers to the questions that are raised in task one.

The reason for not requesting data directly from the external server in script 2 is because the WPS services limit the number of requests accepted by each Internet Protocol address (IP address) in a time period. This made it difficult to use the WPS services directly from the artifact since the number of requests is likely to extend the limit.

Here is a short presentation over what happens when script 1 is executed. Script 1 starts by gathering a list over all hike paths in the SPARQL endpoint. This is done by querying the SPARQL endpoint containing the hike data from step one in the lifting process.

Next, script 1 decodes the encoded hike paths. The hike paths were originally encoded with the "Encoded Polyline Algorithm Format" in the tur.bt.no dataset. The format is convenient to use because it encodes a list of geo points into a single string, which is easy to handle and decode again. The encoding scheme is also a part of the Google Maps API (Anon 2012a), which makes it the obvious choice when displaying paths on Google Maps. Based on this, the format was therefore kept in the lifting process of the hike data. The WPS services of the Norwegian Mapping Authority do however not support this encoding scheme.

Their WPS "elevation" service can only accept one single geo point in a request, so the hike paths must be decoded and feed to the service point by point.

The next task is to fetch and save data about each geo point from the NMA's "elevation" service. Since this script has to be executed several times, this process starts by checking if the current geo point in the list is downloaded before. If it is not downloaded, a HTTP request with the geo point's latitude and longitude will be generated and sent to the "elevation" service.

If the reply from the service is successful, the XML data will be saved to the hard drive of the local server in which the request was sent from. Because JavaScript does not have the ability to save files from where it is running, saving the data is done through PHP. This can be done because script 1 is constructed to execute in a local server environment. A work around solution was implemented to let the JavaScript code save XML files using a proxy server solution constructed in PHP.

After having executed script 1 until all the geospatial data about each point in the hike paths were downloaded from the NMA server, the output folder contained XML data about 3468 geo points.

Script 2

The second script is going to use the XML data downloaded in the first script to find the answers to the questions that are raised in task one.

Script 2 starts in the same way as script 1. It fetches data about all hikes from the data store. This is done in JavaScript and SPARQL. Script 2 then starts the process of generating new RDF triples about each hike. This is done in PHP. The process starts by decoding the selected hike path. A ported version of the "Encoded Polyline Algorithm Format" decoder from JavaScript to PHP was used (Chng 2008). If XML data about all the geo points in a hike path is found in the output folder from script 1, script 2 will have all the required data to generate the extra property facts.

When all the new property facts are generated for each hike, the RDF triples are created in the RDF/TURTLE format and stored locally in an output folder. Listing 1 is an example of an output .ttl file.

```
@prefix sf_ont:       <http://data.sognefjord.vestforsk.no/resource/ontology#> .
@prefix geo:          <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix ucum:         <http://purl.oclc.org/NET/muo/ucum/> .
@prefix owl:          <http://www.w3.org/2002/07/owl#> .
@prefix owl-time:     <http://www.w3.org/2001/sw/BestPractices/OEP/Time-Ontology#> .
@prefix foaf:         <http://xmlns.com/foaf/0.1/> .

sf_ont:hike104
       a                                          owl:Individual ;
       a                                          geo:SpatialThing ;

       owl-time:duration [
              owl-time:minute                     """480""" ;
       ] ;

       sf_ont:Length [
              sf_ont:Kilometer                    """33.4""" ;
       ] ;

       sf_ont:minimumElevation [
              ucum:meter                          """6.69576822445""" ;
       ] ;

       sf_ont:maximumElevation [
              ucum:meter                          """605.0""" ;
       ] ;

       sf_ont:differenceInElevation [
              ucum:meter                          """598.30423177555""" ;
       ] ;

       sf_ont:heightIncrease [
              ucum:meter                          """2629.548002548""" ;
       ] ;

       sf_ont:heightDecrease [
              ucum:meter                          """2627.2345240311""" ;
       ] ;

       sf_ont:StartOf [
              geo:lat                             """60.39233""" ;
              geo:long                            """5.25482""" ;
              geo:altitude                        """26.8681751755""" ;
       ] ;
```

```
        sf_ont:EndOf [
                geo:lat                                 """60.39201""" ;
                geo:long                                """5.3307""" ;
                geo:altitude                            """29.1816536925""" ;
        ] ;

        sf_ont:Path [
                sf_ont:GoogleEncodedPath                """…""" ;
        ] ;

        sf_ont:Profile
                <http://sognefjord.vestforsk.no/resource/route-graph/hike104.png> ;

        foaf:isPrimaryTopicOf
                <http://sognefjord.vestforsk.no/page/hike/104> ;

        owl:sameAs
                <http://tur.bt.no/tur/104> .
```

Listing 1: Property facts that are generated for hike

After the RDF/TURTLE files are generated, they are uploaded to the SPARQL endpoint so they are available for querying.

Script 3

The purpose of the third script is to prepare the download of terrain profiles about every hike path. A terrain profile is a picture that illustrates the hike path in a chart, which provides a visual overview over the hike's variation of elevation, terrain and place names. Figure 4 is an example of a terrain profile that have been generated by the "elevation Chart" service and fetched by the fourth script. The example can be found on the web page about this hike: http://sognefjord.vestforsk.no/page/hike/104

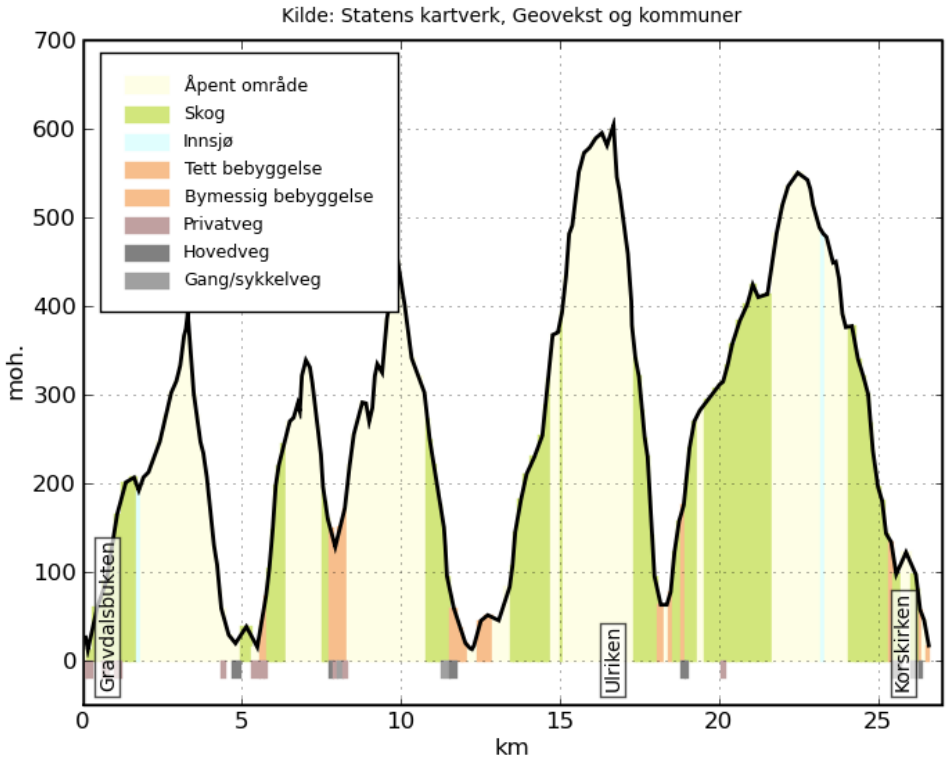


Figure 4: A generated terrain profile of hike

In order to fetch the terrain profiles, the third script will have to generate a GPX file of each hike path. This is because the process that calculates charts in the "elevation Chart" service requires the URL of a GPX file as an input parameter (Hirsch 2011). Each GPX file will contain all the geo points of a selected hike path. They will be stored in a local output folder during creation and thereafter uploaded manually to a server where they will be accessible through URLs. The process in the third script therefore consists of requesting all the hike paths from a SPARQL endpoint, decode the paths into a list of geo points and create a GPX file for each of them.

Script 4

The purpose of the fourth script is to download a terrain profile about every hike path, using the GPX files generated in the third script. The terrain profiles will firstly be downloaded to a local output folder and thereafter uploaded manually to a server where they will be accessible through URLs.

The reason for not designing the artifact to fetch terrain profiles directly from the "elevation Chart" service, are because of the limitations, speed and capacity factors of NMA's WPS-services. Interacting with the "elevation Chart" service in the fourth script revealed issues with all these factors. The service uses more than 30 seconds to calculate a single chart, which is a long time for a user to wait while interacting with a web application. The "elevation Chart" service also had a 30 second limitation on how long a calculating process could last before canceling the request. As a result of this, every request sent to the service would be canceled after 30 seconds. The solution was to contact the NMA which was kind enough to raise the timeout limitation. Script 4 managed to download 105 terrain profiles generated by the "elevation Chart" service.

Script 5

The purpose of the fifth script written is to generate the RDF triples that will link each terrain profile to the correct URI of a hike. The RDF triples will firstly be stored locally in a RDF/TURTLE file and thereafter manually loaded into the data store.

In order to generate the RDF triples, the script goes through the folder with the terrain profiles and reads the name of each picture file. The script thereby has all the names of hikes with available terrain profile and uses this to create the correct URI of each hike. The RDF triples in which the hike data are modeled require the script to get hold of a blank node from the data store. This blank node is going to be used as a subject in the RDF triple. The script therefore uses the hike URIs to query the SPARQL endpoint for the correct blank node of each hike. When the script has all the blank nodes it generates the new RDF triples. These RDF triples were uploaded manually through the data store administration panel.

As a result of the lifting process, we now have newly lifted RDF data that were uploaded to a data store and made accessible through an SPARQL endpoint. The

data store in use is the Virtuoso Open-Source Edition[12] by OpenLink Software. The SPARQL endpoint is accessible here:

- http://sognefjord.vestforsk.no:8890/sparql

Add RDF data to Sindice

In order to add data to the semantic web index of Sindice, the data in question must be detected by the crawlers of the web service. Unless the crawlers have detected the RDF triples of interest by themselves, the service will have to be notified on where to find it. Sindice offers a Ping Submission API[13] for this purpose, which can be used to automate the submission process of RDF data by a system. They also offers a submit form[14] on their web site where datasets can be added manually. RDF data is not submitted to the web service directly, but accepts URLs to semantically enabled pages containing triple statements in form of RDF, RDFa or Microformats.

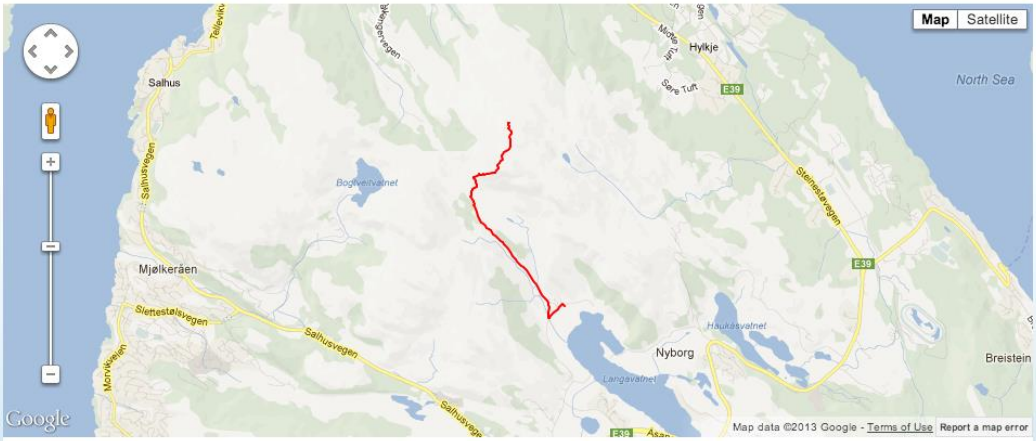
Since Sindice is designed to fetch data from web pages with semantic content, a web document layout was constructed to present the lifted hike facts from the SPARQL endpoint. The layout was constructed through the use of HTML, CSS, JavaScript, PHP, Dojo Toolkit, SPARQL and Google Maps JavaScript API v3. The web page layout was then marked up with RDFa. The hikes were in this way made accessible through both the Web of documents and the Web of data. Figure 5 illustrates the web page of a selected hike. Here are the URLs to a few example of the result:

- http://sognefjord.vestforsk.no/page/hike/101
- http://sognefjord.vestforsk.no/page/hike/102
- http://sognefjord.vestforsk.no/page/hike/104
- http://sognefjord.vestforsk.no/page/hike/105
- http://sognefjord.vestforsk.no/page/hike/109

[12] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/
[13] http://sindice.com/developers/pingApi
[14] http://sindice.com/main/submit

In order to submit the RDFa data to the semantic web index, the URLs of the example pages were added through Sindice's submit form. Sindice then crawled and analyzed the submitted web pages for RDF data before indexing the discovered statements. The result of this submission is illustrated in Figure 6 and can be found here:

- http://sindice.com/search/page?url=http%3A%2F%2Fsognefjord.vestforsk.no%2Fpage%2Fhike%2F101
- http://sindice.com/search/page?url=http%3A%2F%2Fsognefjord.vestforsk.no%2Fpage%2Fhike%2F102
- http://sindice.com/search/page?url=http%3A%2F%2Fsognefjord.vestforsk.no%2Fpage%2Fhike%2F104
- http://sindice.com/search/page?url=http%3A%2F%2Fsognefjord.vestforsk.no%2Fpage%2Fhike%2F105
- http://sindice.com/search/page?url=http%3A%2F%2Fsognefjord.vestforsk.no%2Fpage%2Fhike%2F109



Mellingen-Rimmaskaret-Veten

Veten på 486 moh er det høyeste fjellet på den nordlige delen av Bergenshalvøya. På toppen er det en enestående utsikt særlig mot Nordhordland.

Rutebeskrivelse:
Postvegen stiger bratt oppover mot Rimmaskaret. Langs vegen er det flere gamle restaurerte steinløer. På toppen av stigningen er det en alternativ sti til Veten via Håkeberget. Men vi fortsetter videre på Postvegen og kommer snart til en steinløe som ligger på venstre side. Her er det benker og bord med gode muligheter til rasting både ute og inne. Like etter løen tar stien mot Veten av til høyre. Den fortsetter bratt opp mot toppen. Stort sett er det fast og fint å gå, men noen korte partier med myr.

Tilkomst:
Til Mellingen kommer en enklest med privatbil. På motorvegen gjennom Ãâ €¦sane tar vegen til Mellingen og Espelid av på Nyborg ved Gullgruven. Bak det nye bygget til IKEA åpner det seg et gammelt kulturlandskap. Første del av denne turen følger den gamle postvegen som starter rett før gårdene på Mellingen. Det er parkeringsplass litt lenger borte langs bilvegen.

Tidsbruk:
Turen tar tilsammen omtrent to timer.

Passer for:
Alle som liker å gå tur i variert terreng.

Utstyrsbehov:

På tørre dager kan du godt gå med joggesko. Pass på at det kan blåse på toppen og kle deg deretter.

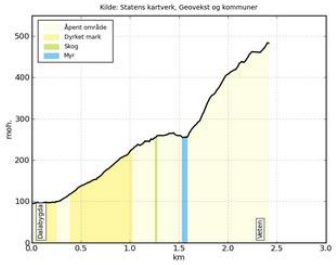
Fakta om turen:
Navnet Veten finner man igjen på mange fjell. Det har bakgrunn i et eldgammelt varslingssystem, på vetene varslet man om fiender ved å fyre opp bål.

Facts

Duration
60 minutes

Length
2.5 kilometer

Profile



Minimum elevation
94.7445042805 meter

Maximum elevation
484.149341852 meter

Difference in elevation
389.4048375715 meter

Height increase
414.2001581891 meter

Height decrease
25.7197524276 meter

Same as
http://tur.bt.no/tur/101

sognefjord.vestforsk.no

Figure 5: A web page presenting information about a hike

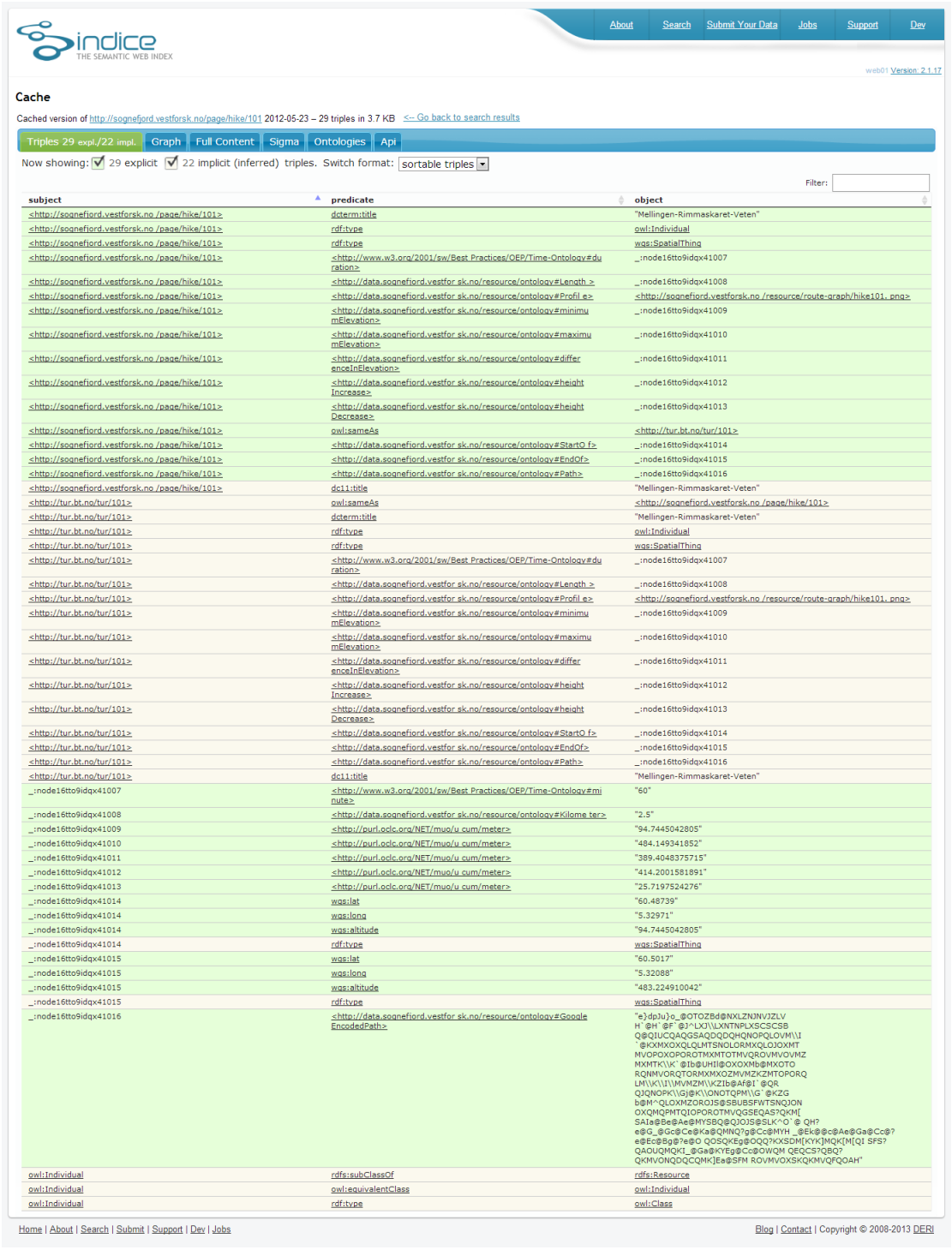


Figure 6: How RDF triples from a web page looks like after being submitted to the Sindice platform

Evaluation

The second iteration, was like the first iteration, also a long and educational process.

The first task of finding a geospatial dataset containing relevant property facts became a much more time consuming process than expected. The Sprek data source did not contain all the property facts itself, but had potential of becoming the dataset that I was looking for. The process of lifting the legacy data for two different projects and use it to pull more data out from other data sources was an interesting experience that demonstrates much of the potential that lies within a Web of data. Combining data from different data sources can become more useful and valuable.

Regarding the lifting process itself, lifting the XML file from Sprek, using XSLT, was conducted as expected. Requesting data from the Norwegian Mapping Authority were more challenging as of the service limitations.

In summary, the second iteration was time consuming and required a lot of research in theory and practice. The outcome of the process resulted in a great learning experience and a new data source with geospatial things, rich on property facts, which could be indexed by Sindice. The second iteration therefor provided what is needed to move forward with this project.

Third Iteration

After two long iterations with preparations, it was finally time to focus on constructing the SemanticGeoBrowser. Starting on the construction phase, the tasks set for this iteration is based on the system requirements, which is a result of the previous iterations.

Designing an user interface that operates on the level of things

System requirement #1: The SemanticGeoBrowser should operate at the level of "things" (instead of at the level of documents) and treat them as first-class citizens in a user-friendly interface.

The first draft constructed of the user interface is based on the web page layout used in the Semantic Sognefjord project, but also on the web page layout that was used to add hikes to the Sindice platform. This layout type was designed to make the user focus on the things presented on the page. The decision to implement this layout was firstly made because it was a quick solution to employ, but I also wanted to see if the layout could work when operating on the level of things. I was also eager to start developing on the Sindice functionality so I decided to first test out this quick layout solution and rather change it in a later iteration if needed. An example of the first layout used is illustrated in Figure 7.

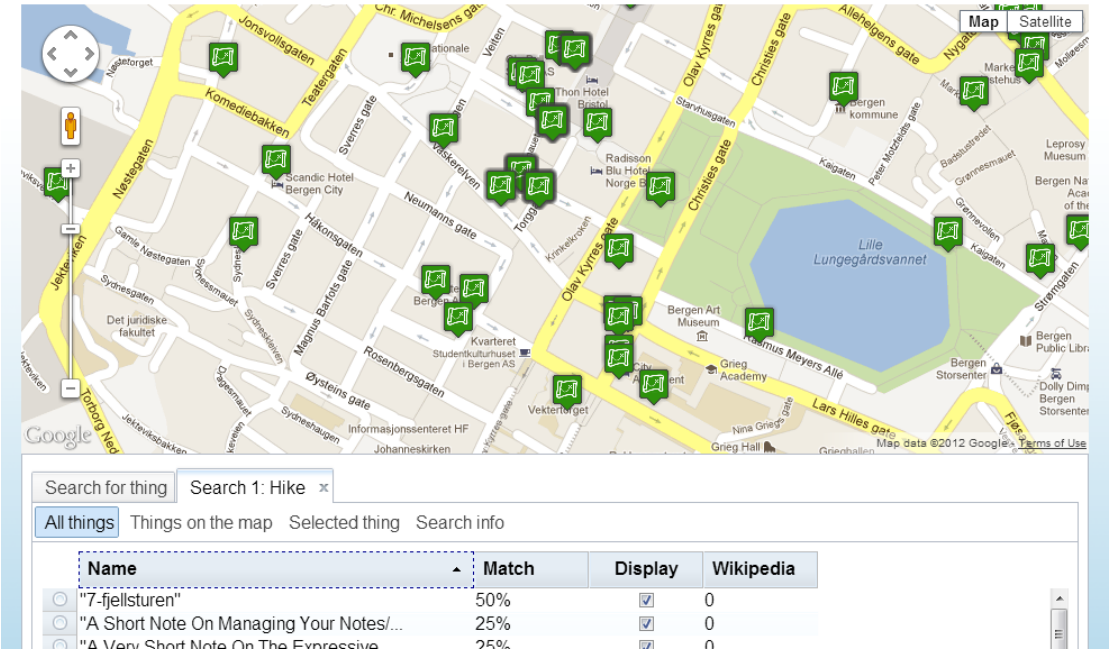


Figure 7: The first layout of the SemanticGeoBrowser

The layout has a common web page setup and is divided into two sections. The first section displays a map pane. The second section is divided in tabs. The first tab, named “Search for thing”, displays the control pane. This is where the search criteria are set before conducting a search. Other tabs display search results. A new tab pops up when a new search is executed and lists the things discovered for user review.

Selecting an area of interest

System requirement #2: The SemanticGeoBrowser should contain an interactive map of the planet Earth, which enables the user to

a. **explore the Web of data by selecting an area of interest.**
b. **interact with the "things" discovered on the Web of data.**

The map feature is essential to the SemanticGeoBrowser. In order to make the SemanticGeoBrowser operate on the level of things, it is important to place the map pane in the center of the layout. This is because most of the user's focus and interaction is expected to be directed towards the map and the things illustrated on it. The intention is also that this will lead the artifact to become more user-friendly. The map pane is illustrated in Figure 7. The map enables the artifact to work as a lens over the semantic web by magnifying geospatial things that is described in a selected area. Although the map feature is great for visualizing geospatial things, it is also useful in the process of conducting a search. The idea is to implement the map usage in both the process of conducting the search and the process of exploring the search result.

Knowledge about thing characterizations

System requirement #3: The SemanticGeoBrowser should be knowledge-based in order to

a. **help the user search for relevant things.**
b. **help the user recognize things that are relevant to the domain of operation.**
c. **present facts about relevant things in a user-friendly way.**

The SemanticGeoBrowser should contain knowledge about different types of things within the domain it is operating. The reason for this is to help the user in the process of searching and exploring relevant things described in RDF data.

Searching things in a central area can be like looking for a needle in a haystack. Especially if the user is uncertain on what characterizes the things of interest. If the user conducts an open search for all geospatial things in an area of interest

that happens to be a popular area in the world, the result can be hundreds or thousands of different things. Figure 8 illustrates how a search in Sindice, for all geospatial things, will look like if it is not limited when exploring the area of Bergen.

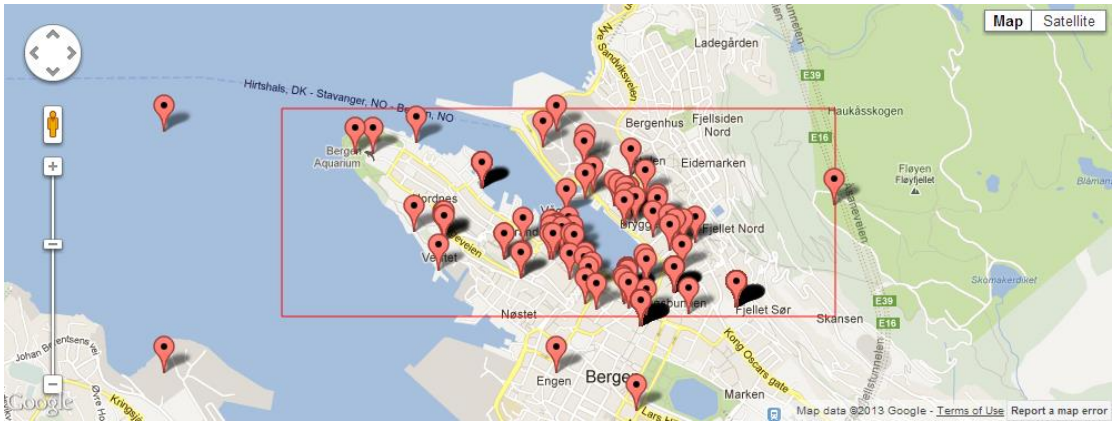


Figure 8: A search in Sindice for all geospatial things in a popular area in the city of Bergen

The task of examining just a few geospatial things for relevant facts in RDF data would be time-consuming and tiresome for a human user. Since machine artifacts can do an excellent job of reading RDF data, they have the potential of becoming important tools for RDF interpretation. In order for an artifact to become useful in this, it will have to know what to do with the RDF data. The SemanticGeoBrowser will therefore have to contain knowledge about what vocabulary terms is used in RDF to describe relevant things. The artifact's knowledge can then be applied to look for, recognize and present relevant things to the user in different ways.

In order to meet the system requirements described in this section, the artifact must gain the knowledge from someone. System requirement #3 (a) and (b) can in my view be solved in two different ways. The knowledge can either be added by the user, which requires the user to have advanced knowledge about the domain, or it can be pre-implemented by the creator of the artifact. System requirement #3 (c) can in my view only be solved by a system developer. This is because it would take a system developer to program the artifact to perform different tasks that would be triggered when the system recognize the

knowledge it is being feed. This includes programming it to present facts about relevant things in a user-friendly way. Since the SemanticGeoBrowser aims to be user-friendly, a set of vocabulary terms is added as knowledge by the developer.

In order to fulfill system requirement #3 (a), to "*help the user search for relevant things*", a pre-defined list over relevant things will be implemented in the control pane of the artifact. These relevant things should reflect the type of things the artifact knows about. The idea is that this knowledge should be used as a blueprint when searching for things. This can be done by implementing the characterization of a relevant thing in the search query as a RDF triple pattern. This should result in the exclusion of things with no matching vocabulary terms from the search result. The user will in this way end up with things that are in some ways relevant to the thing blueprint. Figure 9 shows the concept of a list where the user is presented with relevant things to search for within the domain of operation.

Figure 9: The concept of a list with relevant things to search for within the domain of operation

In order to fulfill system requirement #3 (b), to "*help the user recognize things that are relevant to the domain of operation*", the artifact should calculate the percentage of how many matching vocabulary terms each thing in the search result have compared to the blueprint of the type of thing that was used to recognize the things of interest. The more percentage a thing has the more matching RDF triples the thing will have. For example, if a type of thing contains four different RDF triple patterns and a thing are described with two of those RDF triples, the relevance will be calculated to a 50% match. Figure 10 shows the

concept of a list that display the percentage of matching vocabulary terms will display to determine the thing relevance to the user.

Name	Match
Bergen Offentlige Bibliotek	50%
http://dbpedia.org/data/Bergen.rdf	50%
http://dbpedia.org/data/Bergen.n3	50%
Apotek 1 Nygårdsgaten - Apotek 1	25%
"7-fjellsturen"	25%
Lille Lungeg00E5rdsvannet	25%
"Uitgebreide beschrijving van golfhot...	25%
"Uitgebreide beschrijving van golfhot...	25%

Figure 10: The concept of a list where the percentage of matching vocabulary terms will display to determine the thing relevance to the user

System requirement #3 (c), to "*present facts about relevant things in a user-friendly way*", was focused upon in the fourth iteration.

Constructing a search query for Sindice

System requirement #4: The SemanticGeoBrowser should make use of a semantic web index look-up service that provide

a. **access to a large amount of RDF datasets from the Web of data.**
b. **advanced geospatial query capabilities to be made within a selected area of interest.**

As discovered in iteration one, the semantic web index of Sindice seems like the perfect match for the SemanticGeoBrowser. Their service will provide the artifact with advanced query capabilities over a large amount of geospatial data from the semantic web. In order for the artifact to use this service, it will have to be able to generate search queries using their query language. Here is an example over how a search query for Sindice is constructed in the artifact.

In the scenario of a user conducting a search, the focus starts at the map pane. The user navigates the map to an area of interest. The focus continues to the

control pane. The user, which is operating within the domain of *tourism*, then selects *foot hike* as the thing of interest. Since the artifact only support one data source at this point, the user does not have to select one. The user initiates a search by clicking on the search button in the control pane.

The artifact starts the search process by constructing a search query for the data source, which in this case is the Sindice Search API. The search query is divided into three parts. All parts are merged together in an URL and sent as an HTTP request.

The first part contains the resource path, which is the address to the server in which the HTTP request is sent to. The other parts of the request contain the query parameters, specifying the search requirements.

The second part is where the area of interest is defined, which equals the rectangular shape of the map pane. The query parameters are defined by the current position of the map like described in the first iteration. Since this part is included in every query sent to the Sindice Search API, all searches constructed by the SemanticGeoBrowser are dependent on data from Literal Triples. This is because the location of things is described with this type of RDF triples.

The third part of the search query is where the thing of interest is defined. A search query constructed for Sindice Search API is designed to request one selected type of thing. This part is however not included in the query if the use have selected to search for “Everything”. By using the known URIs in which the selected type of thing are described, this part of the query is requesting documents with matching RDF triple patterns. This is archived by using the Ntriple Query notation in the Sindice Query language.

When the search query is constructed, the artifact continues by sending it as an HTTP request to Sindice. Listing 2 illustrate an example of a constructed search query, where “Hike” is selected as the type of thing of interest. The example is URL decoded in order to make it more human readable.

```
http://api.sindice.com/v3/search?q=
(geo:lat [60.38209922415584 TO 60.39855351612633]) AND
(geo:long [5.278172408813475 TO 5.3644322507324205])&nq=(
(* <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Individual>) AND
(* <http://data.sognefjord.vestforsk.no/resource/ontology#Path> *)
)&field=predicate&format=json&page=1
```

Listing 2: A search query constructed for the Sindice Search API

Requesting data from Sindice

System requirement #5: The SemanticGeoBrowser should avoid solutions that would trigger the web browser to reload a lot.

Requesting data from third party web services can be tricky. Especially when the request must be sent from code running on the client-side. The term “client-side” is used to describe the local computer used by an individual user. The SemanticGeoBrowser aims to be a web application that runs from a web browser on the client-side. In this way some of the processing load will be unloaded from the web server hosting the web application and over on the client-side of the user. Figure 11 illustrates the difference between the server-side and client-side.

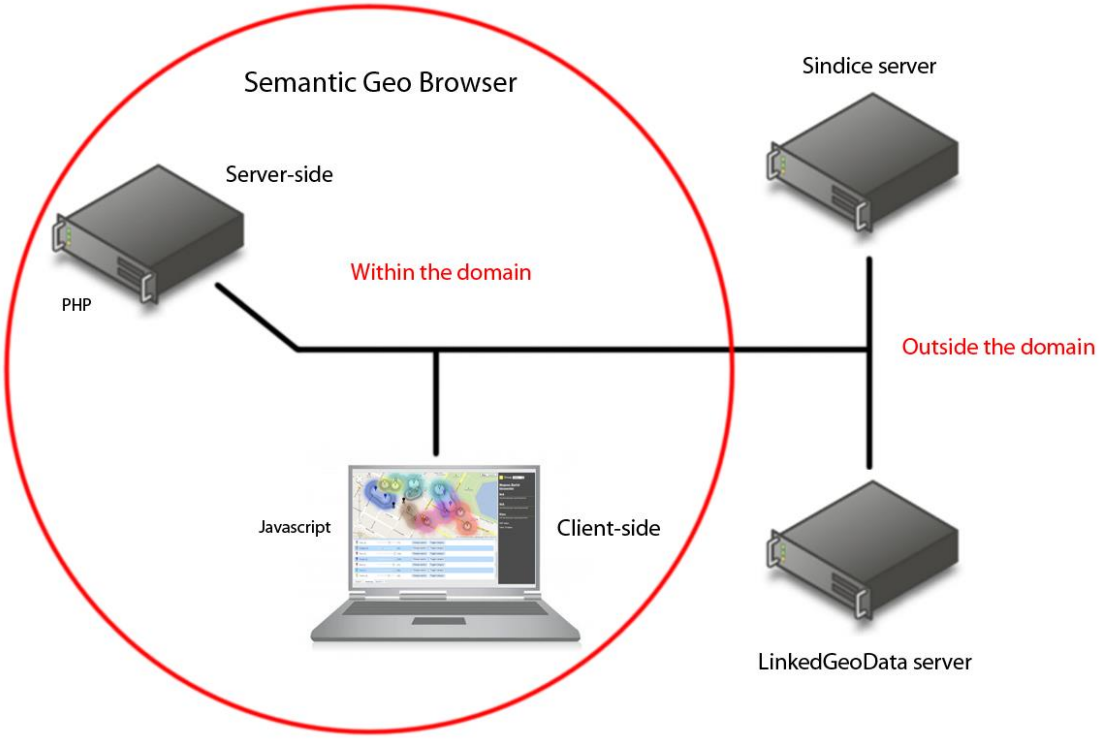


Figure 11: Illustration of the artifact's cross-domain communications

In order to prevent malicious web sites from requesting personalized data and perform other security threats, web browsers have implemented security features to limit the risk of browsing web sites with hidden agendas. The web browsers can however not completely block features like HTTP requests because it enables a lot of the dynamic behaviors seen in a major part of web sites and web applications today. Instead, web browsers have implemented security features like the "same-origin policy", which is made to prevent unrelated web sites to access each other's user sessions (Anon 2012b). The same-origin policy contains a mechanism for "XMLHttpRequest". The XMLHttpRequest specification defines a JavaScript API for client-side scripts to make HTTP requests to the same server in which the web page originated (Anon 2009). This will enable HTTP requests to be sent to a server that is under the website owner's control. The mechanism was originally envisioned to transfer XML data between a client and a server, but is today also used to transfer other data formats, like JSON, and "*serves as the foundation for much of the web 2.0 behavior of rapid UI updates not dependent on full-page transitions*" (Zalewski 2009).

In order to make the artifact avoid the behavior of a general web document, which would trigger the web browser to reload every time new data is requested, some security features in the web browser has to be broken. There are several ways to work around a web browser's security features, allowing the SemanticGeoBrowser to perform client-side cross-domain communications. The solution chosen for requesting data from the Sindice Search API is a feature provided by the Dojo Toolkit framework.

The framework has a module named "dojo.io.script" which provides access to JSONP resources. JSON with Padding (JSONP) was first introduced by Bob Ippolito in December 2005 (Ippolito 2005) and is a method for conducting cross-domain data fetching.

Listing 3 shows how the dojo.io.script module is used in order to request data from the Sindice Search API. The module requires a Uniform Resource Locator (URL), which in addition to the resource path must also contain the query parameters like described in the first iteration. According to Machi (2012) the server response is a "*JSON message wrapped in a callback function*". In the example shown in Listing 3, the callback function is handled by a custom function if the request is conducted successfully.

```
dojo.io.script.get({
      callbackParamName: "callback",
      url: url,
      handleAs: "json",
      load: function(sindice_reply){
            //...
      }
});
```

Listing 3: Dojo´s dojo.io.script module enables cross-domain communications between client and server

Because of this module, the SemanticGeoBrowser will be able to send the query constructed in the previous sub chapter.

Fetching data about each thing

System requirement #6: The SemanticGeoBrowser should be able to assemble and handle RDF data seamlessly behind the scenes.

After a search query has been sent to the Sindice Search API, the service will process the request and send a reply in return. If the query was processed successfully, the reply will contain an object with a lot of parameters. This object is described as a "result page" by the platform and contains metadata about the search result. An example of a result page is illustrated in Listing 4.


```
{
"totalResults":7879186,
"author":"Sindice.com",
"title":"Sindice search: hotel",
"itemsPerPage":10,
"startIndex":0,
"updated":"2013/02/01/ 19:33:18 +0000",
"search":"http://www.sindice.com/opensearch.xml",
"base":"http://api.sindice.com/v3/search?q=hotel&format=json",
"link":"http://api.sindice.com/v3/search?q=hotel&format=json&page=1",
"alternate":"http://sindice.com/search?q=hotel&page=1",
"first":"http://api.sindice.com/v3/search?q=hotel&format=json&page=1",
"last":"http://api.sindice.com/v3/search?q=hotel&format=json&page=787918",
"previous":"http://api.sindice.com/v3/search?q=hotel&format=json&page=1",
"self":"http://api.sindice.com/v3/search?q=hotel&format=json&page=1",
"next":"http://api.sindice.com/v3/search?q=hotel&format=json&page=2",
"cache_batch":"http://api.sindice.com/v3/cache?field=explicit_cont....Hotel-Services",
"entries":[
        {
                "link":"http://www.suburbanhotels.com/es/hotel-woodstock-georgia-
GA558",
                "cache":"http://api.sindice.com/v3/cache?field=explicit_content
&output=json....georgia-GA558",
                "updated":"2011/09/25",
                "formats":["MICRODATA","RDFA"],
                "title":[{
                        "type":"literal",
                        "value":"\"choicehotels.com/hotel/GA558\""
                }],
                "rank":1,
                "explicit_content_size":"16",
                "explicit_content_length":"2160"
        },
        {
                "link":"http://www.suburbanhotels.com/fr/hotel-woodstock-georgia-
```

```
GA558",
                "cache":"http://api.sindice.com/v3/cache?field=explicit_content
&output=json....georgia-GA558",
                "updated":"2011/09/25",
                "formats":["MICRODATA","RDFA"],
                "title":[{
                        "type":"literal",
                        "value":"\"choicehotels.com/hotel/GA558\""
                }],
                "rank":2,
                "explicit_content_size":"16",
                "explicit_content_length":"2174"
        },
        ....
],
"query":{
        "startIndex":0,
        "role":"request",
        "searchTerms":"hotel",
        "responseTime":1721
}
}
```


Listing 4: Example figure of successful reply from Sindice Search API

As discovered in iteration one, the search result from the Sindice Search API is divided into several result pages. Each result page contains metadata about maximum ten documents. These data source documents contain RDF triples about geospatial things that match the search query. The service has limited the number of results by only allowing access to the first 100 result pages.

In order for the SemanticGeoBrowser to fetch data about each thing, it first has to fetch a complete list over all the data source documents in the result pages. Each result page contains metadata about the search itself and the other result pages. The artifact uses this metadata to navigate through all the result pages. The metadata about the data source documents can be seen in the parameter "entries", at page 59, in Listing 4. The most important information about each document is the URL stored in the parameter "link".

When the web application has fetched all the URL's made available from the Search API, the artifact uses each of them in a HTTP request to the Sindice Cache

API[15]. This API provides read-only access to the Sindice Data Store (Anon 2013d). When providing the API with a URL of a data source document in which the Sindice platform has indexed, the artifact get access to that document's latest data, which is cached by the Sindice crawlers. Here is an example on a query to the Cache API:

http://api.sindice.com/v3/cache?pretty=true&url=http%3A%2F%2Fsognefjord.vestforsk.no%2Fpage%2Fhike%2F101&output=json

Listing 5 illustrates the Cache API's JSON reply from the example query.

[15] http://sindice.com/developers/cacheapi

```
{"http://sognefjord.vestforsk.no/page/hike/101": {
  "explicit_content":   [

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://purl.org/dc/terms/title>
\"Mellingen-Rimmaskaret-Veten\" .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Individual> .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing> .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://www.w3.org/2001/sw/BestPractices/OEP/Time-Ontology#duration>
_:node16tto9idqx41007 .\n",

    "_:node16tto9idqx41007
<http://www.w3.org/2001/sw/BestPractices/OEP/Time-Ontology#minute>
\"60\" .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#Length>
_:node16tto9idqx41008 .\n",

    "_:node16tto9idqx41008
<http://data.sognefjord.vestforsk.no/resource/ontology#Kilometer>
\"2.5\" .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#Profile>
<http://sognefjord.vestforsk.no/resource/route-graph/hike101.png> .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#minimumElevation>
_:node16tto9idqx41009 .\n",

    "_:node16tto9idqx41009
<http://purl.oclc.org/NET/muo/ucum/meter>
\"94.7445042805\" .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#maximumElevation>
_:node16tto9idqx41010 .\n",

    "_:node16tto9idqx41010
<http://purl.oclc.org/NET/muo/ucum/meter>
\"484.149341852\" .\n",
```

```
    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#differenceInElevation>
_:node16tto9idqx41011 .\n",

    "_:node16tto9idqx41011
<http://purl.oclc.org/NET/muo/ucum/meter>
\"389.4048375715\" .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#heightIncrease>
_:node16tto9idqx41012 .\n",

    "_:node16tto9idqx41012
<http://purl.oclc.org/NET/muo/ucum/meter>
\"414.2001581891\" .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#heightDecrease>
_:node16tto9idqx41013 .\n",

    "_:node16tto9idqx41013
<http://purl.oclc.org/NET/muo/ucum/meter>
\"25.7197524276\" .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://www.w3.org/2002/07/owl#sameAs>
<http://tur.bt.no/tur/101> .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#StartOf>
_:node16tto9idqx41014 .\n",

    "_:node16tto9idqx41014
<http://www.w3.org/2003/01/geo/wgs84_pos#lat>
\"60.48739\" .\n",

    "_:node16tto9idqx41014
<http://www.w3.org/2003/01/geo/wgs84_pos#long>
\"5.32971\" .\n",

    "_:node16tto9idqx41014
<http://www.w3.org/2003/01/geo/wgs84_pos#altitude>
\"94.7445042805\" .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#EndOf>
_:node16tto9idqx41015 .\n",

    "_:node16tto9idqx41015
<http://www.w3.org/2003/01/geo/wgs84_pos#lat>
```

```
\"60.5017\" .\n",

    "_:node16tto9idqx41015
<http://www.w3.org/2003/01/geo/wgs84_pos#long>
\"5.32088\" .\n",

    "_:node16tto9idqx41015
<http://www.w3.org/2003/01/geo/wgs84_pos#altitude>
\"483.224910042\" .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#Path>
_:node16tto9idqx41016 .\n",

    "_:node16tto9idqx41016
<http://data.sognefjord.vestforsk.no/resource/ontology#GoogleEncodedPath>
\"e}dpJu}o_@OTOZBd@NXLZNJNVJZLVH`@H`@F`@J^LXJ\\\\LXNTNPLXSCSCSBQ@QIUCQAQGSAQDQDQHQNOPQ
LOVM\\\\I`@KXMXOXQLQLMTSNOLORMXQLOJOXMTMVOPOXOPOROTMXMTOTMVQROVMVOVMZMXMTK\\\\K`@Ib@UH
Il@OXOXMb@MXOTORQNMVORQTORMXMXOZMVMZKZMTOPORQLM\\\\K\\\\I\\\\MVMZM\\\\KZIb@Af@I`@QRQJQ
NOPK\\\\Gj@K\\\\ONOTQPM\\\\G`@KZGb@M^QLOXMZOROJS@SBUBSFWTSNQJONOXQMQPMTQIOPOROTMVQGSEQ
AS?QKM[SAIa@Be@Ae@MYSBQ@QJOJS@SLK^O`@QH?e@G_@Gc@Ce@Ka@QMNQ?g@Cc@MYH_@Ek@@c@Ae@Ga@Cc@?e
@Ec@Bg@?e@OQOSQKEg@OQQ?KXSDM[KYK]MQK[M[QISFS?QAOUQMQKI_@Ga@KYEg@Cc@OWQMQEQCS?QBQ?QKMVO
NQDQCQMK]Ea@SFMROVMVOXSKQKMVQFQOAH\" .\n"

  ],
  "implicit_content_length": "2705",
  "implicit_content_size": "22",
  "data_source": "SIGMA",
  "predicate":   [
    "http://purl.org/dc/terms/title",
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
    "http://www.w3.org/2001/sw/BestPractices/OEP/Time-Ontology#duration",
    "http://www.w3.org/2001/sw/BestPractices/OEP/Time-Ontology#minute",
    "http://data.sognefjord.vestforsk.no/resource/ontology#Length",
    "http://data.sognefjord.vestforsk.no/resource/ontology#Kilometer",
    "http://data.sognefjord.vestforsk.no/resource/ontology#Profile",
    "http://data.sognefjord.vestforsk.no/resource/ontology#minimumElevation",
    "http://purl.oclc.org/NET/muo/ucum/meter",
    "http://data.sognefjord.vestforsk.no/resource/ontology#maximumElevation",
    "http://data.sognefjord.vestforsk.no/resource/ontology#differenceInElevation",
    "http://data.sognefjord.vestforsk.no/resource/ontology#heightIncrease",
    "http://data.sognefjord.vestforsk.no/resource/ontology#heightDecrease",
    "http://www.w3.org/2002/07/owl#sameAs",
    "http://data.sognefjord.vestforsk.no/resource/ontology#StartOf",
    "http://www.w3.org/2003/01/geo/wgs84_pos#lat",
    "http://www.w3.org/2003/01/geo/wgs84_pos#long",
    "http://www.w3.org/2003/01/geo/wgs84_pos#altitude",
    "http://data.sognefjord.vestforsk.no/resource/ontology#EndOf",
    "http://data.sognefjord.vestforsk.no/resource/ontology#Path",
    "http://data.sognefjord.vestforsk.no/resource/ontology#GoogleEncodedPath"
  ],
```

```
  "label": ["\"Mellingen-Rimmaskaret-Veten\""],
  "checksum": "92487ed420195df5022ade11e27f807bd23d083e",
  "format":   [
    "TITLE",
    "RDFA"
  ],
  "ontology":   [
    "http://dublincore.org/2010/10/11/dcterms.rdf",
    "http://www.w3.org/1999/02/22-rdf-syntax-ns",
    "http://www.w3.org/2002/07/owl",
    "http://www.w3.org/2003/01/geo/wgs84_pos",
    "http://dublincore.org/2010/10/11/dcam.rdf",
    "http://dublincore.org/2010/10/11/dcelements.rdf",
    "http://dublincore.org/2010/10/11/dctype.rdf",
    "http://www.w3.org/2000/01/rdf-schema",
    "http://www.w3.org/2003/g/data-view",
    "http://www.w3.org/TR/skos-reference/skos.html",
    "http://xmlns.com/foaf/0.1/accountProfilePage",
    "http://usefulinc.com/ns/doap",
    "http://www.rddl.org/purposes/",
    "http://www.w3.org/2000/10/swap/pim/contact",
    "http://www.w3.org/2003/06/sw-vocab-status/ns"
  ],
  "url": "http://sognefjord.vestforsk.no/page/hike/101",
  "size": "29",
  "timestamp": "2012-05-23T20:38:16.000",
  "length": "3835",
  "domain": "sognefjord.vestforsk.no",
  "implicit_content":   [

    "_:node16tto9idqx41014
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing> .\n",

    "_:node16tto9idqx41015
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing> .\n",

    "<http://sognefjord.vestforsk.no/page/hike/101>
<http://purl.org/dc/elements/1.1/title>
\"Mellingen-Rimmaskaret-Veten\" .\n",

    "<http://www.w3.org/2002/07/owl#Individual>
<http://www.w3.org/2000/01/rdf-schema#subClassOf>
<http://www.w3.org/2000/01/rdf-schema#Resource> .\n",

    "<http://tur.bt.no/tur/101>
<http://www.w3.org/2002/07/owl#sameAs>
<http://sognefjord.vestforsk.no/page/hike/101> .\n",

    "<http://tur.bt.no/tur/101>
```

```
<http://purl.org/dc/terms/title>
\"Mellingen-Rimmaskaret-Veten\" .\n",

    "<http://tur.bt.no/tur/101>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Individual> .\n",

    "<http://tur.bt.no/tur/101>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing> .\n",

    "<http://tur.bt.no/tur/101>
<http://www.w3.org/2001/sw/BestPractices/OEP/Time-Ontology#duration>
_:node16tto9idqx41007 .\n",

    "<http://tur.bt.no/tur/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#Length>
_:node16tto9idqx41008 .\n",

    "<http://tur.bt.no/tur/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#Profile>
<http://sognefjord.vestforsk.no/resource/route-graph/hike101.png> .\n",

    "<http://tur.bt.no/tur/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#minimumElevation>
_:node16tto9idqx41009 .\n",

    "<http://tur.bt.no/tur/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#maximumElevation>
_:node16tto9idqx41010 .\n",

    "<http://tur.bt.no/tur/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#differenceInElevation>
_:node16tto9idqx41011 .\n",

    "<http://tur.bt.no/tur/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#heightIncrease>
_:node16tto9idqx41012 .\n",

    "<http://tur.bt.no/tur/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#heightDecrease>
_:node16tto9idqx41013 .\n",

    "<http://tur.bt.no/tur/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#StartOf>
_:node16tto9idqx41014 .\n",

    "<http://tur.bt.no/tur/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#EndOf>
_:node16tto9idqx41015 .\n",
```

```
    "<http://tur.bt.no/tur/101>
<http://data.sognefjord.vestforsk.no/resource/ontology#Path>
_:node16tto9idqx41016 .\n",

    "<http://tur.bt.no/tur/101>
<http://purl.org/dc/elements/1.1/title>
\"Mellingen-Rimmaskaret-Veten\" .\n",

    "<http://www.w3.org/2002/07/owl#Individual>
<http://www.w3.org/2002/07/owl#equivalentClass>
<http://www.w3.org/2002/07/owl#Individual> .\n",

    "<http://www.w3.org/2002/07/owl#Individual>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Class> .\n"

  ]
}}
```

Listing 5: Example of result from the Cache API

Each result page from the Cache API is stored in the artifact for further processing. For this, the artifact makes use of "Dojo Store", a feature included in the Dojo Toolkit framework with the purpose of making it easier to store data and query it afterwards.

Handling data from Sindice

When all result pages from the Cache API are stored in the artifact, the next step is to analyze the fetched data. The artifact will conduct the analysis by looking for patterns that are programmed into the web application. The outcome of this process will decide what the artifact can do with the fetched data. The more data the artifact recognizes, the more abilities the artifact will have to do something interesting with it.

The analysis process starts by decoding the part of the fetched data that are in the form of RDF triples. These are the RDF triples fetched by the Sindice crawlers in addition to some extra RDF triples added by Sindice. Additional RDF triples is a result of reasoning conducted by the Sindice platform. Listing 6 illustrates this by displaying a RDF triple that have been added by the platform after detecting geospatial triples in the crawled RDF data.

```
<http://sognefjord.vestforsk.no/page/hike/101>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing> .
```

Listing 6: The Sindice platform conducts reasoning

Additional statements can for example make it easier to query the Search API for things that are geospatial. The RDF triples in the Cache API is however encoded in the form of N-Triples. Since the artifact's implemented Dojo Store is not able to query RDF triples encoded in N-Triples, each RDF triple is decoded by extracting the statement's subject, predicate and object. These are stored in the artifact's Dojo Store in a way that makes it possible for the system to query the RDF triples.

The artifact starts to analyze the decoded RDF triples by detecting geospatial things described in each result document. This is done so the SemanticGeoBrowser can operate on the level of things. Some documents contain descriptions of multiple things, but every individual detected is treated as an independent thing. Each thing might contain multiple geo locations, in which the artifact supports.

As a result of this process, all the geospatial things in the search result are detected. Moreover, an *index card* is being created for each of the detected things in the artifact's Dojo Store. The index cards are used to store the results of the analysis, which are further used to quickly look up and present data in the system.

The first data stored in an index card is the URI of an identified individual. Same as in RDF data, the URI is used as a unique identifier for the detected individuals throughout the system and is further used in the analysis process. When the artifact recognizes a set of properties that can be used by the system, the URIs of the property owners are used to find back to each of the individual's index card where the property facts are stored. Since the geo points are the criteria that identify the geospatial things, they are stored in the same process as the URIs. The geo points come in form of a longitude and latitude.

In order to identify individuals with property facts, the system query the decoded RDF triples using a set of RDF triple patterns. These patterns are programmed into the system. Listing 7 shows an example of some of the RDF triple patterns the system is looking for.

```
var find_profile_property = {
        predicate:
        "http://data.sognefjord.vestforsk.no/resource/ontology#Profile",
        attribute_name:          "hike_profile"
};

var find_duration_property = {
        predicate:                       "http://www.w3.org/2001/sw/BestPractices/OEP/Time-
Ontology#duration",
        find_sub_property:       [
                {
                        predicate:
        "http://www.w3.org/2001/sw/BestPractices/OEP/Time-Ontology#minute",
                        attribute_name:          "hike_duration_in_minutes"
                }
        ]
};

var find_start_of_property = {
        predicate:
        "http://data.sognefjord.vestforsk.no/resource/ontology#StartOf",
        find_sub_property:       [
                {
                        predicate:
        "http://www.w3.org/2003/01/geo/wgs84_pos#lat",
                        attribute_name:          "hike_start_of_lat"
                },
                {
                        predicate:
        "http://www.w3.org/2003/01/geo/wgs84_pos#long",
                        attribute_name:          "hike_start_of_long"
                },
                {
                        predicate:
        "http://www.w3.org/2003/01/geo/wgs84_pos#altitude",
                        attribute_name:          "hike_start_of_altitude"
                }
        ]
};
```

Listing 7: Some of the RDF triple-patterns the system is looking for

Listing 7 illustrates three examples of property facts that the system looks for. The list, which is written in JavaScript, defines the RDF triple patterns through objects. Each object contains the parameter *predicate* that contains the URI of the

predicate that describes the desired property fact. Every object also contains the parameter *attribute_name*, which contains the name of the parameter used to describe the attribute when storing the discovery in an individual's index card. In order to describe patterns of RDF statements that consist of multiple triples, which are possible by linking triples together using blank nodes, the parameter *find_sub_property* is used to describe triples that use a blank node as subject.

Figure 12 shows an example of an index card that describes the SemanticGeoBrowser's knowledge of an individual after the analysis process is conducted.

```
> console.debug(store.data[6]);
▼ Object {id: "search_nr_1_result_nr_1", name: ""7-fjellsturen"", uri: "http://sog
  nefjord.vestforsk.no/page/hike/104", location: Array[2], type: "searchResult"…}
    checkbox: true
  ▶ detected_wikipedia_articles: Array[0]
    dojo_store_id_number: 1
    hike_difference_in_elevation_in_meters: "598.30423177555"
    hike_difference_in_elevation_in_meters_status: true
    hike_difficulty: "Hard"
    hike_difficulty_status: true
    hike_duration_in_minutes: "480"
    hike_duration_in_minutes_status: true
    hike_end_of_altitude: "29.1816536925"
    hike_end_of_altitude_status: true
    hike_end_of_lat: "60.39201"
    hike_end_of_lat_status: true
    hike_end_of_long: "5.3307"
    hike_end_of_long_status: true
    hike_height_decrease_in_meters: "2627.2345240311"
    hike_height_decrease_in_meters_status: true
    hike_height_increase_in_meters: "2629.548002548"
    hike_height_increase_in_meters_status: true
    hike_length_in_kilometers: "33.4"
    hike_length_in_kilometers_status: true
    hike_maximum_elevation_in_meters: "605.0"
    hike_maximum_elevation_in_meters_status: true
    hike_minimum_elevation_in_meters: "6.69576822445"
    hike_minimum_elevation_in_meters_status: true
    hike_profile: "http://sognefjord.vestforsk.no/resource/route-graph/hike104.png"
    hike_profile_status: true
    hike_start_of_altitude: "26.8681751755"
    hike_start_of_altitude_status: true
    hike_start_of_lat: "60.39233"
    hike_start_of_lat_status: true
    hike_start_of_long: "5.25482"
    hike_start_of_long_status: true
    id: "search_nr_1_result_nr_1"
  ▶ location: Array[2]
  ▶ location_markers: Array[2]
    name: ""7-fjellsturen""
    number_of_detected_wikipedia_articles: 0
    number_of_matching_rdf_triples: 3
    number_of_predicates: 21
    number_of_rdf_triples: 51
    percent_matching_rdf_triples: 100
  ▶ polyline_markers: Array[1]
  ▶ predicates: Array[21]
    selected_thing: true
  ▶ sindice_cache_api_reply: Object
    type: "searchResult"
    uri: "http://sognefjord.vestforsk.no/page/hike/104"
  ▶ __proto__: Object
```


Figure 12: An object representing an index card with facts about a selected thing

After the artifact has analyzed the fetched data, the system calculates how many percentages each individual matched the search that was made. This is already presented as the solution of system requirement #3 (b) earlier in this iteration.

The searching process ends by displaying the geospatial things from the search result on the map. A list of the result, illustrated in Figure 10, is also generated. The data is now ready to be explored by the user.

Evaluation

While the two first iterations consisted of foundational preparations, the third iteration started the development phase of designing the artifact itself. Moving into this development phase, the evaluation phase that will end each of the next iterations will be focused on the fulfillment of the system requirements. This is because the result from an evaluation phase is intended to provide essential feedback to the following construction phase in the next iteration (Hevner et al. 2004, p.85). The iterations will continue until the system requirements are fulfilled. As mentioned by Hevner et al. (2004, p.85) in guideline three, a design artifact is complete and effective when it satisfies the requirements and constraints of the problem it was meant to solve.

In order to fulfill system requirement #1, the artifact will have to operate at the level of "things" (instead of at the level of documents) and treat them as first-class citizens in a user-friendly interface. The user interface that was implemented in the third iteration did however not work as intended. Even though the layout manages to put things in the center of the user interface, it still had the look and feel of a web document in use. The main reason for this, in my experience, is that the map disappears from the screen when scrolling the page to see the list of search results. In order to fulfill this requirement, the user interface will have to be redesigned in the next iteration. The new interface should have a map that does not disappear from the screen when exploring the search result.

System requirement #2, of implementing a map into the artifact, is considered fulfilled. The Google Maps API was easy to work with and contained all the necessary features for completing the job. For example, the API made it easy to select an area of interest by seamlessly returning the required coordinates when interacting with the map and it was just as feasible to handle the things on the map through the use of JavaScript. The result of the implemented map is an artifact that is easier to interact with.

System requirement #3, of making the artifact knowledge-based, is in this iteration considered partly fulfilled. Point (a), to "*help the user search for relevant things*", is considered fulfilled through the implementation of a list of things that is relevant to the domain of operation. The list represents the knowledge over things the artifact knows about, which helps the user with alternatives over things to search for. Point (b), to "*help the user recognize things that are relevant to the domain of operation*", is also considered fulfilled through the implementation of a list that indicates the relevance of the search result to the user. Point (c) is not considered fulfilled as it is postponed to the fourth iteration.

System requirement #4, to "*make use of a semantic web index look-up service*", is considered fulfilled. Through the use of the web services of Sindice, their APIs provide "*access to a large amount of RDF datasets from the Web of data*" and "*advanced geospatial query capabilities to be made within a selected area of interest*", which fulfills point (a) and (b).

System requirement #5, to "*avoid solutions that would trigger the web browser to reload a lot*", has been archived in the start of the construction phase and is therefore at this time considered fulfilled.

System requirement #6, to "*assemble and handle RDF data seamlessly behind the scenes*", has been somewhat archived and is at this time considered fulfilled. The reason for calling it *somewhat archived* is because requesting data from third party services is conducted synchronous in the SemanticGeoBrowser. Requesting data from third party services is archived through the use of AJAX. According to Chapman (2012), one of the biggest advantages of using AJAX in web pages is that it can be used to access data from a server without having to reload the web page. There is two ways that AJAX can communicate with a server. The first way is running the request synchronous, which stops the JavaScript until the server has replied to the request. This solution has a big down side because it can make the web page appear frozen while it is awaiting the reply, which is negative for the user experience. The second way is running the requests asynchronous, which lets multiple functions be processed at the

same time. In the time of coding I decided to run the requests synchronously because I thought it was easier to structure and build up the script on code that runs synchronously. Because the artifact appears frozen while waiting on the search result, I would have chosen to code the browser to run asynchronous if I were to recode the artifact.

Fourth Iteration

In this fourth iteration, the goal is to redesigning the artifact's interface, implement user-friendly fact box, and make the artifact conduct reasoning.

Designing a more user friendly interface

In order to improve the user interface from iteration three, the layout has been redesigned so the map does not disappear from the screen when the user is scrolling the interface to explore the list of search results. By designing a fixed user interface where the map is in the center at all times, the SemanticGeoBrowser aims for a look and feel of a web application, instead of a web document. Figure 13 illustrates the improved user interface.

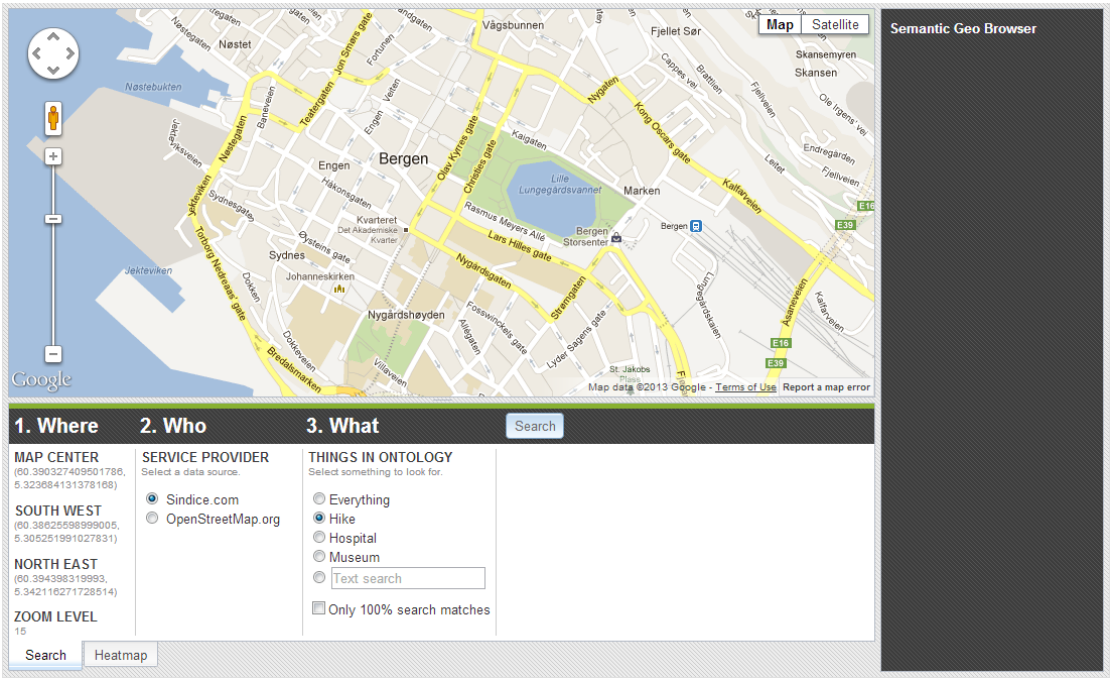


Figure 13: The second layout of the SemanticGeoBrowser

The artifact´s improved user interface is divided into three panes with different purpose. The map pane is located in the upper left corner and gives users the ability to navigate and zoom in on any location on the earth in order to select the area of interest. When a search is completed, the map is used to visualize and able interaction with the geospatial results. The control pane is located in the lower left corner and divided into tabs at the bottom of the pane. The first tab is the control panel for the search and is where the search criteria can be changed. Other tabs are visible when searches are made where each tab represents the control panel for an individual search. The focus pane that is located on the right side provides information about things that are selected.

Designing an user-friendly fact box

In order to fulfill system requirement #3 (c), to "*present facts about relevant things in a user-friendly way*", the artifact is designed to look up the index card of a thing that is selected on the map or in the result list and display the facts found on the card in a fact box. This is done by considering how each type of fact should be presented to the user and designing an individual layout for it. By this, the artifact is able to display facts from all vocabulary terms that are recognized in the analysis process of RDF statements. Figure 14 shows an example of how the artifact presents facts about one of the hikes added to Sindice.

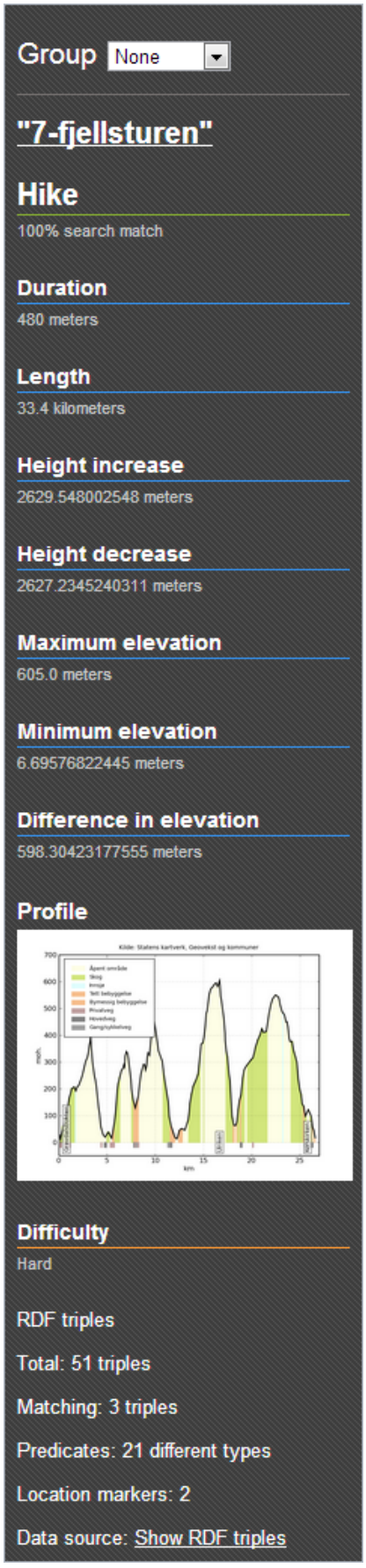


Figure 14: A fact box presenting facts from recognized RDF triples that describes a hike

Reasoning with property facts

System requirement #7: The SemanticGeoBrowser should be able to draw conclusions from facts described in the properties of things.

Knowledge about data within the domain of operation opens up the possibility of conducting reasoning on the discovered property facts. The term "reasoning" is used to describe the powerful mechanism to draw conclusions from facts (Verborgh 2012a). Another term that is commonly used to describe reasoning is "inference", which is the act or process of deriving logical conclusions from premises known or assumed to be true, hence the act of reasoning (Anon 2013c). A good reason for implementing reasoning into the artifact is to let the machine assist the human to draw conclusions from recognizable facts identified in machine-readable data. By creating rules, the machine can conduct reasoning.

In order to enable a reasoning feature into the artifact, I have chosen to implement an open source version of the EYE (Euler YAP Engine)[16], which is an inference engine that is built to supports logic based proofs.

EYE was selected as the artifact's reasoning engine because of the initiative aim to "*provide a user-friendly reasoning experience in current Web browsers and applications*" (Verborgh 2012a). In order to bring reasoning to the Web, the EYE reasoner consists of two parts, a **reasoner server** and a **browser widget**. Figure 15, which is created by Verborgh (2012a), is an illustration on how the server and browser communicates. The reasoner server part is the EYE reasoning server itself, made accessible to anyone on the Internet through an API. This part has a public API, but can also be downloaded from Ruben Verborgh's "EyeServer" project page[17] at Github and installed on a private server. The browser widget part is a demonstrator that exemplifies how a web application can communicate with the EYE reasoner server. The demonstrator can be downloaded from Ruben Verborgh's "EyeClient" project page[18] at Github.

[16] http://eulersharp.sourceforge.net/
[17] https://github.com/RubenVerborgh/EyeServer
[18] https://github.com/RubenVerborgh/EyeClient

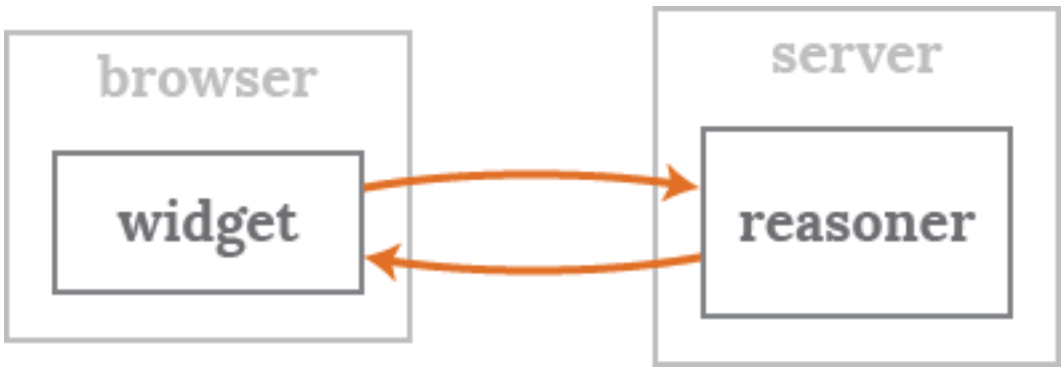


Figure 15 – How the browser widget and reasoner server communicate

In order to implement EYE into the artifact, I chose to use the publicly available reasoner server because it was the fastest solution. When it came to implementing the EYE reasoner on the client side, I studied the source code of the browser widget in order to see how it worked. The same technique used to communicate with the reasoner server was then applied to the artifact.

The SemanticGeoBrowser demonstrates how reasoning can be used when it recognize the properties of things in which the user is browsing. Each time a user selects a thing, the browser checks if the thing has all the properties that would qualify for reasoning. If the properties of a selected thing provide all the facts that are required for a set of rules, supported by the artifact, the reasoning process starts.

The reasoning process starts by building an object that contains all the parameters that are required by the reasoner server. Listing 8 shows how this object is built.

```
var     options = {}; //object
        options.data = [];
        options.data.push(n3_data_input);
        options.data.push(rules_input);
        options.path = "http://eye.restdesc.org/";
        options.query = "{ ?a ?b ?c. } => { ?a ?b ?c. }.";
```

Listing 8: An object that contains all the parameters that are required by the reasoner server

The EYE reasoner server requires three inputs, which is *facts*, *rules* and a *query* (Verborgh 2012b). The facts include all the RDF triples found about the selected

thing. The array “explicit_content”, in Listing 5, shows an example of RDF triples that would be sent as facts to the reasoner server. The RDF triples must be added in the N3 syntax.

The rules included are custom made for the things that contains the right facts. Listing 9 shows an example of how rules can be written to say something about the difficulty level of a hike. This example includes the difficult levels of “Medium” and “Hard”. The first rule conclude that if the thing has a duration, in minutes, greater than 15 minutes and less than 60 minutes, then the difficulty level must be “Medium”. The second rule is made to capture the gap that occurs between less than 60 minutes and more than 60 minutes. The third rule captures all the hikes that have a duration greater than 60 minutes, which is concluded to be “Hard”. Even though the process of setting the difficult level on hikes would require a lot more properties to be justifiable, this example demonstrate how rules can help the concept of an SemanticGeoBrowser to draw conclusion from facts appearing in the search result.

```
@prefix sf_ont: <http://data.sognefjord.vestforsk.no/resource/ontology#> .
@prefix owl-time: <http://www.w3.org/2001/sw/BestPractices/OEP/Time-Ontology#> .
@prefix math: <http://www.w3.org/2000/10/swap/math#> .

{
        ?a      owl-time:duration      ?x .
        ?x      owl-time:minute        ?y .
        ?y      math:greaterThan       15 .
        ?y      math:lessThan          60 .
}
=>
{
        ?a      sf_ont:Difficulty      """Medium""" .
}.

{
        ?a      owl-time:duration      ?x .
        ?x      owl-time:minute        ?y .
        ?y      math:equalTo           60 .
}
=>
{
        ?a      sf_ont:Difficulty      """Medium""" .
}.
```

```
{
        ?a      owl-time:duration       ?x .
        ?x      owl-time:minute         ?y .
        ?y      math:greaterThan        60 .
}
=>
{
        ?a      sf_ont:Difficulty       """Hard""" .
}.
```

Listing 9: How rules can be written to say something about the difficulty level of a hike

The query included in the Listing 8 says what the EYE reasoner server should do. In this example, the server is told to conduct deductive reasoning.

"*Deductive reasoning, also called deductive logic, is the process of reasoning from one or more general statements regarding what is known to reach a logically certain conclusion*" (Sternberg 2009) cited by (Wikipedia 2012).

"*Deductive reasoning involves using given true premises to reach a conclusion that is also true*" (Wikipedia 2012).

"*An example of a deductive argument:*

1. *All men are mortal.*
2. *John is a man.*
3. *Therefore, John is mortal.*"

(Wikipedia 2012)

The object also contains a fourth parameter, which is the path to the public reasoning server. This is used by the function that is sending the object from the client to the server. The object is sent as a HTTP request.

The SemanticGeoBrowser now conducts reasoning each time it recognizes all the required facts that are needed by a rule to draw a conclusion. In order to avoid sending a lot of requests to the EYE reasoner at the same time, the artifact will check a thing for required facts when the user selects it in the user interface. The

system will store the result of the reasoning and only conduct reasoning the first time the thing is selected. At this time, the artifact will recognize some facts in the hikes that were added to Sindice. When one of those hikes is selected, the recognized facts will be sent to the EYE reasoner and its conclusion will be presented in the selected thing's fact box in a user-friendly way. Figure 16 illustrates an example of how the result of reasoning is presented to the user.

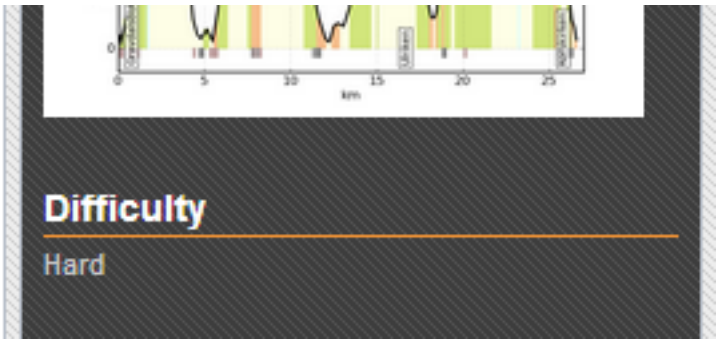


Figure 16: How the result of reasoning is presented to the user

Evaluation

The fourth iteration starts with redesigning the artifact to fulfill system requirement #1. The new layout has a setup where it is not possible to scroll the map out of the screen, which I think is a more user-friendly solution. The artifact does now have the look and feel of a web application that operates on the level of things, contrary to a general web document. System requirement #1 is therefor considered as fulfilled.

System requirement #3 (c), which was postponed in iteration three, is in this fourth iteration considered fulfilled.

System requirement #7, of enabling the artifact to "*draw conclusions from facts described in the properties of things*" was accomplished in this iteration and is therefore considered fulfilled. Even though the implemented example of a reasoner was simple and small, it is a concrete example of how reasoning can be implemented and how works in a semantic web application.

Fifth Iteration

The fifth iteration is about the implementation of a second data source into the SemanticGeoBrowser. The second data source is not a semantic web index like Sindice, but instead a set of datasets that are controlled by the data provider. These datasets are accessible through a single SPARQL endpoint and are based on open data from the OpenStreetMap project[19]. The RDF triples is not provided or generated by OpenStreetMap themselves, but rather a result of the LinkedGeoData project[20]. As explained in chapter 2, the LinkedGeoData project "*uses the information collected by the OpenStreetMap project and makes it available as an RDF knowledge base according to the Linked Data principles*" (Stadler 2012).

The decision to implement a second data source was based on the idea that thing browsers should be able to help users in the task of finding specific types of things, within a selected domain, on the Web of data, that in the starting point are unknown to the system. Since the task of finding something specific requires the knowledge of *what* characterizations to look for, a thing browser without this knowledge would not be able to find anything specific, unless it was able to acquire this knowledge from somewhere. While search engines operating on the Web of documents are based on their users entering words from a language and vocabulary they master, users of the Web of data should not be required to feed thing browsers with RDF triple patterns in which only advanced users would know the meaning of. Instead, thing browsers should focus on providing its users with user-friendly options. In iteration three, knowledge was coded in to the artifact by the developer. In this iteration, the SemanticGeoBrowser should try to fetch and apply external knowledge by coding *where* and *how* to acquire it.

Implementing a second data source

System requirement #8: The SemanticGeoBrowser should support different data sources and apply knowledge from an external ontology.

[19] http://www.openstreetmap.org/
[20] http://linkedgeodata.org/About

In order for the artifact to implement knowledge from an external data source, it must be accessible. By accessible, I mean in a standard place where it is easy to find and use. In addition to provide geospatial RDF data generated through the use of OpenStreetMap data, the LinkedGeoData project have derived a lightweight ontology from the same lifting process (Stadler et al. 2012). The LinkedGeoData ontology is accessible through the same SPARQL endpoint as their RDF data. This makes it possible for the SemanticGeoBrowser to fetch both knowledge and RDF data through the use of SPARQL queries.

When a user is going to search for something in the second data source, the search process starts in the control pane where the user selects the OpenStreetMap data source. This selection triggers the artifact to query the LinkedGeoData ontology for all its knowledge about types of things. By this, the browser fetches the ontology's knowledge about characterizations that can be used to find specific types of things in the SPARQL endpoint. The acquired knowledge is presented to the user in form of a list where a thing of interest can be selected. Instead of presenting the characterizations in form of RDF triple patterns to the user, the browser uses the Norwegian labels that were found in the ontology. At the time of coding there were no English labels to be found.

The user continues the search process by selecting a type of thing from the generated list. If the user does not select anything, the search will query for all geospatial things in the area of interest. If the user do select a type of thing, it is also possible to add an extra filter for the search by selecting another type of thing and a radius area in kilometers. The search will then only search for things that is within the radius of the second type of thing. When the search criteria are set, the search is started by a click on the search button. Figure 17 shows an example of a search criteria set in the control pane.

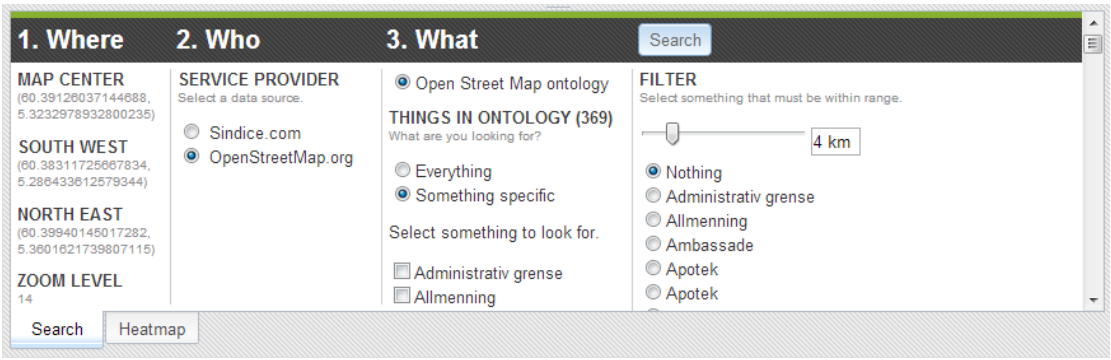


Figure 17: A search criteria set in the control pane

The artifact starts the search by generating a SPARQL query with inputs from the control pane and the map pane. While the selected RDF triple pattern from the control pane can be inserted directly into the search query, the coordinates from the southwest and northeast corners of the map must be processed in order to get a square area of interest that equals the rectangular shape of the map pane. The reason for this is the software powering the SPARQL endpoint. Because the version of Virtuoso in the time of coding did not contain a query function for filtering geospatial things within a square shaped area, the browser would have to calculate the coordinates for the area of interest itself. The solution was found in a forum post written by Claus Stadler, which presented a formula in JavaScript code[21]. This code was implemented in the browser. Listing 10 shows an example of a SPARQL query that is generated with input from the implemented algorithm.

```
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
SELECT count(?thing_uri) as ?thing_count
FROM <http://linkedgeodata.org>
WHERE {
       ?thing_uri          geo:geometry  ?geo .
       Filter(
              bif:st_intersects(
                     ?geo,
                     bif:st_point(5.313792144531249, 60.3919271497259),
                     2.544099138178441
              )
```

[21] https://groups.google.com/forum/?fromgroups=#!topic/linked-geo-data/BXuH45-IXdU

```
        ) .
        Filter(
                bif:st_x(?geo) > 5.270833884948729 &&
                bif:st_x(?geo) < 5.356750404113768 &&
                bif:st_y(?geo) > 60.38338236409244 &&
                bif:st_y(?geo) < 60.40047193535936
        ) .
}
```

Listing 10: A query generated for the LinkedGeoData SPARQL endpoint

This example is used to count all geospatial things in a square shaped area of interest, which in this case is the city of Bergen. It shows how a square shaped area of interest is shaped in a SPARQL query by using two filter clauses. In order to illustrate how a SPARQL query like this works, Figure 18 was constructed. The first filter clause filters away all geospatial things that are found outside a radius-selected area. This is illustrated in Figure 18 and would result in all the markers inside the black circle. The second filter clause shapes the square selected area by filtering away all things from the radius selected area that falls outside the square area. This is also illustrated in Figure 18, where all the blue markers will be filtered away, leaving the red markers left as the query result. It is the coordinates used in these two filter clauses that are calculated by the functions from Claus Stadler.

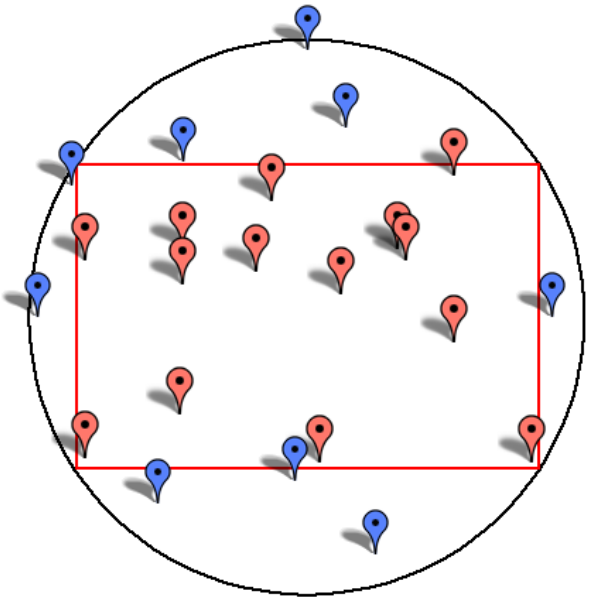


Figure 18: How filtering geospatial things within a square shaped area is happening through the LinkedGeoData SPARQL endpoint

Even though removing the first filter clause from the SPARQL query would result in the same search result, it is not redundant. The reason for this is that the first filter clause represents an outer circle, of the rectangular area, that would likely optimize the query's performance (Stadler & Knibbe 2011). Since it is believed that the second filter clause will take more time to process than the first filter clause, the first filter is applied to make a fast and rough trim down of possible result items. The second filter clause is however the filter that is defining the square shape of interest.

After the search query is generated, it is sent to the SPARQL endpoint as a HTTP GET request. The querying process is divided into two steps. The first step consists of gathering a complete list of all URI's to the things in the search result. Then, the second step consists of requesting data about each thing using the URI's from the first step. The number of queries sent to the SPARQL endpoint will therefore depend on the number of thing in the search result. This must be done because of a default setting in the SPARQL endpoint that limits the maximum number of one thousand RDF triples for each reply on a request. The artifact must therefore start the first step by requesting a count of RDF triples in the search result. If the number is under or equals one thousand RDF triples, a single SPARQL query will be enough to fetch a complete list of URI's. Otherwise, a count larger than one thousand RDF triples requires a separate SPARQL query to be sent for each thousand RDF triples in the count.

When the search result is fetched from the SPARQL endpoint, it is stored in the artifact's data store, powered by Dojo Toolkit. This makes it possible for the artifact to query the fetched data in an analysis process.

The analysis process consists of querying the gathered data for RDF triple patterns. When a RDF triple is recognized, the thing described with that triple will be indexed in order for the system to use the acquired knowledge after the analysis process is completed.

After the analysis process is conducted, the artifact presents the discovered things on the map and result pane. At this time, the result presented in the result pane will not display each discovered thing in a list, but rather display a category list with type of things discovered. This list enables the user to toggle what group of things that are displayed on the map. This feature should help the user to focus on one type of things at a time when exploring a result with many different types of things. Figure 19 illustrates a category list generated in a search for anything in the area of Bergen.

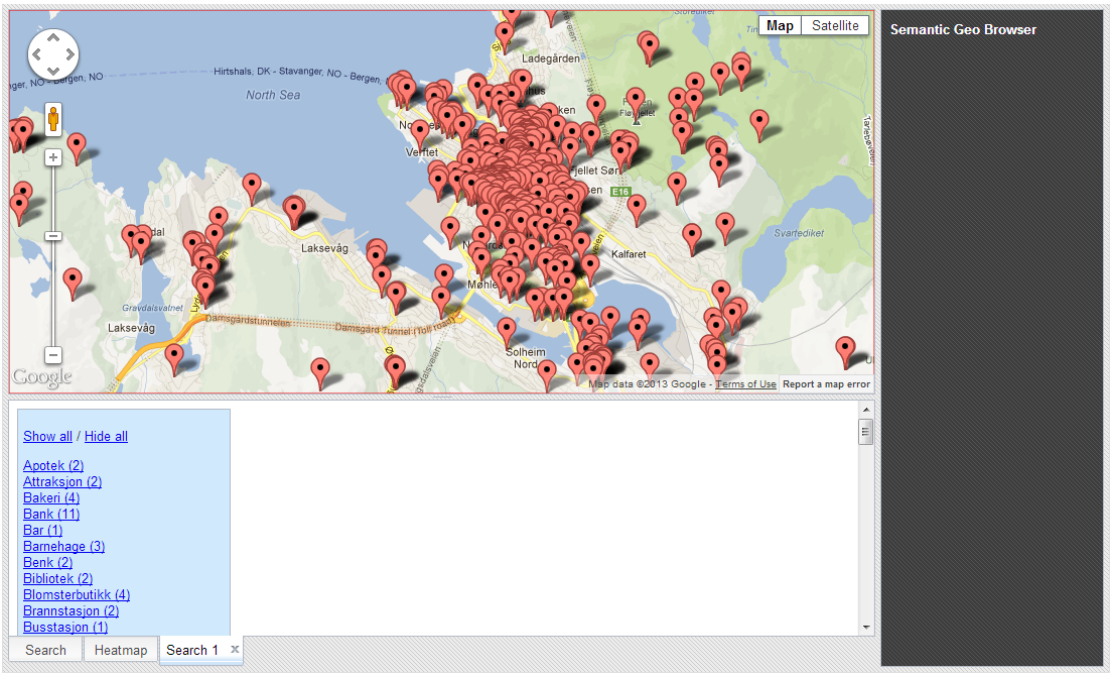


Figure 19: A category list generated in a search for anything in the area of Bergen

As can be seen in Figure 19, exploring a popular area of interest can be difficult if there are too many things to explore on the map at once. Figure 20 illustrates the same example as Figure 19, but with the exception that the user has selected the category “Minibank (10)” from the category list.

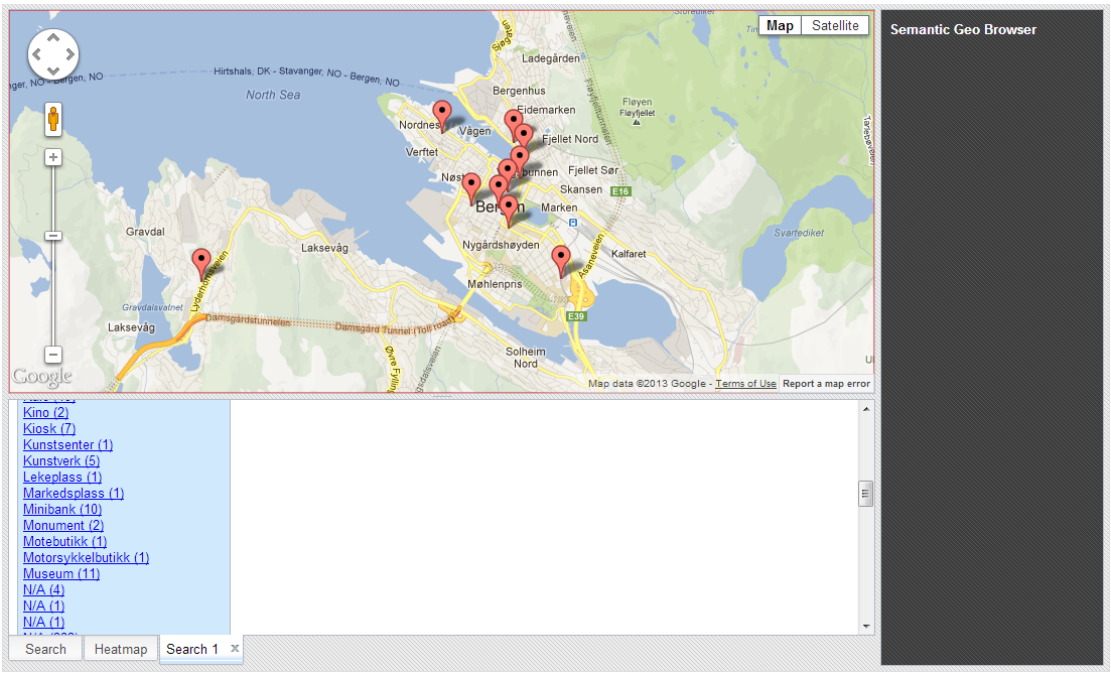


Figure 20: After an option in the category list is selected

The word “minibank” is Norwegian and stands for automated teller machine (ATM). Because the labels in the dataset were not available in English at the time of coding, the artifact is displaying the Norwegian labels that were available. The number behind the labels in the category list indicates the number of things in the category.

When selecting a thing from the OpenStreetMap / LinkedGeoData data source, the fact box will display the categories the selected individual is a member of. Figure 21 illustrates a fact box displaying information about a selected thing.

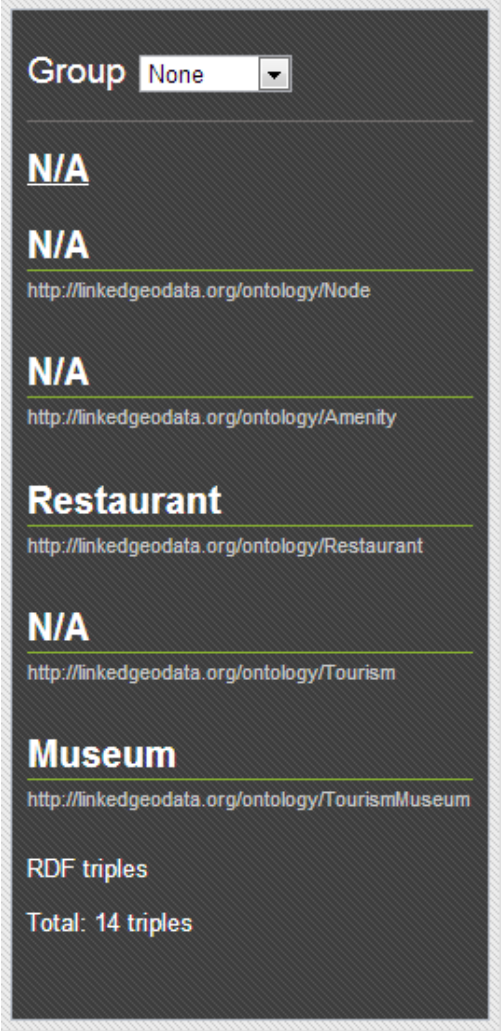


Figure 21: A fact box displaying information about a selected thing

As seen in Figure 21, a label will display "N/A" if there is no label available to display.

Evaluation

The fifth iteration focused on one system requirement alone. System requirement #8, to "*support different data sources and apply knowledge from an external ontology*", was in this iteration archived and is therefore considered as fulfilled.

Sixth Iteration

In the sixth iteration, the goal is to make it easier to discover and see new patterns in geospatial data loaded in the SemanticGeoBrowser. Also, if the user experience a scenario where relevant thing characterizations are not available, it should be possible to conduct a free text search.

Implementing categorization and heat map feature

System requirement #9: The SemanticGeoBrowser should help the user to discover patterns shaped by the coordinates of geospatial things.

In the process of letting the user find and explore geospatial things in the SemanticGeoBrowser, the artifact should contain features to distinguish things that are relevant to the user. In order to solve this task, a categorization and heat map feature was implemented.

The categorization feature is designed to make it easy for the user to sort the geospatial things in the exploration process. Since the desired outcome of this process is for the user to get an overview and discover interesting things in an area of interest, the categorization feature should provide the user with a method to sort things after the user's perception of the individual. Figure 22 illustrates the category feature.

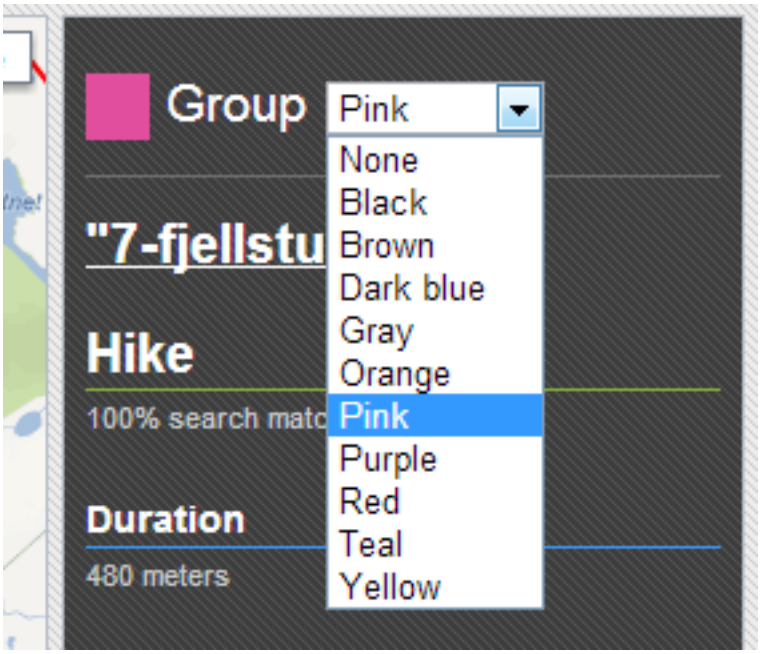


Figure 22: The category feature

The categories are predefined by colors. In this way the user will not have to create any categories on his/her own and brainstorm creative group names. The

meaning of a category is created by the user's perception of the things that is added to the category, which is remembered by the category color.

When a thing is added to a category, its map icon is changed to the color of the category. If the user uses the categorization feature to mark all things that have been explored, including the things that are not of interest, the things on the map will start to show a pattern. Things that have yet to be explored will also be revealed by the pattern. Figure 23 illustrates the categorization and heat map feature in use.

Figure 23: Categorization and heat map feature in use

The heat map feature extends the categorization feature by highlighting the things that are categorized on the map. The highlighting reflects the color of a thing's category and fills the area around the thing in a radius. The icon and heat map pattern should make it easier to discover relations between things and it works across dataset from different sources.

The heat map feature has its own control panel that appears by clicking on the "Heatmap" tab. In this control panel the user can adjust the radius, color strength and toggle the heat map on / off. Figure 24 illustrates the control pane of the heat map feature.

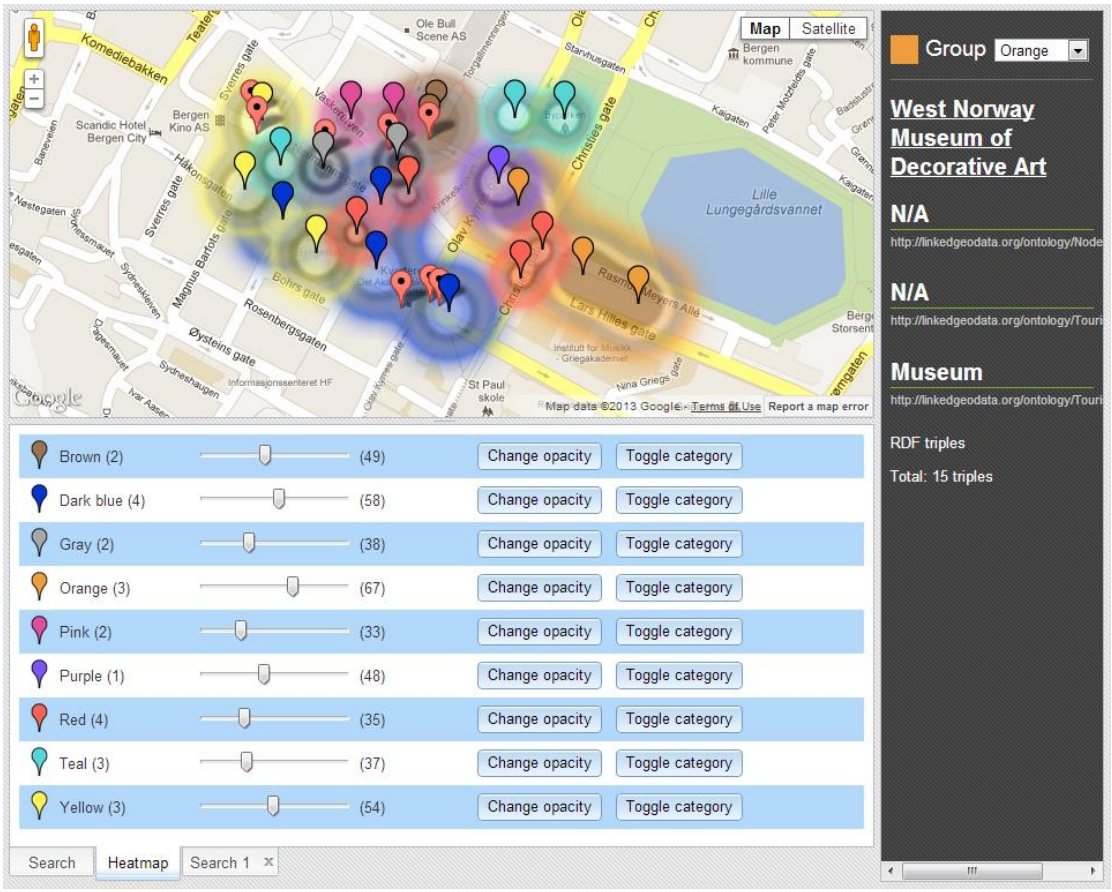


Figure 24: The control pane of the heat map feature

When the heat map radius of things from the same category overlaps on the map, the overlapping area merges. The heat map functionality is provided by Google Maps API[22].

Implementing text search feature

System requirement #10: The SemanticGeoBrowser should allow users to conduct text searches when available thing characterizations aren't enough.

Querying for things in RDF data can be hard if the URIs describing the things are unknown. Since the Sindice Search API supports search by words used in URIs and literal values in RDF triples, an extra search feature has been implemented in this last iteration. The SemanticGeoBrowser now supports search by human text input in the Sindice data source. Figure 25 illustrates the text search field in the control pane for the search.

[22] https://developers.google.com/maps/documentation/javascript/layers#JSHeatMaps

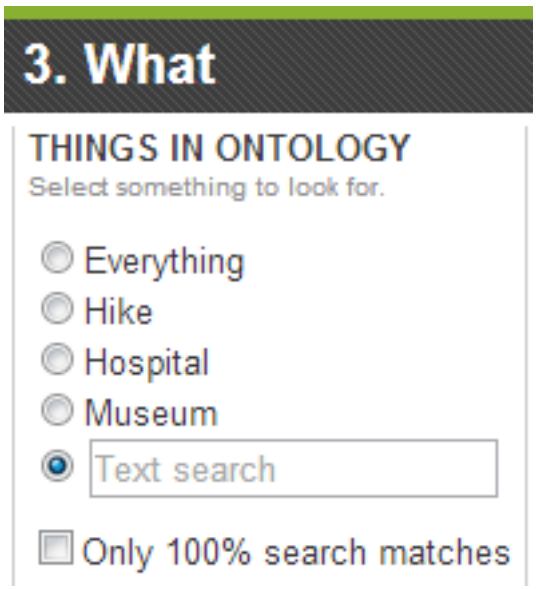


Figure 25: The text search field in the control pane for the search

Evaluation

In the sixth iteration, the two last system requirements were fulfilled.

System requirement #9, to "*help the users to discover patterns shaped by the coordinates of geospatial things*", was achieved through the categorization and heat map feature, and is thereby considered fulfilled.

System requirement #10, to "*allow users to conduct text searches when available thing characterizations aren't enough*", was in this iteration implemented to support the Sindice platform and it by this considered fulfilled.

Chapter 5

5 Evaluation and Discussion

This chapter will present an evaluation made by the completed artifact, followed by a discussion of this study's result.

Descriptive Evaluation

In order to evaluate the designed artifact, Hevner et al. (2004, p.86) advice the use of methodologies from the knowledge base. The evaluation methods that are presented in the knowledge base of design-science are well executed and strengthen the rigorousness of the research, which Hevner et al. (2004, p.87) emphasize through guideline five.

The selected evaluation method for the SemanticGeoBrowser is in this project going to be "descriptive". Hevner et al. (2004, p.86) points out that the selection of evaluation methods must be matched appropriately with the designed artifact and the selected evaluation metrics. It is also pointed out that descriptive methods of evaluation should only be used for especially innovative artifacts for which other forms of evaluation may not be feasible. After having reviewed the different methodologies presented by Hevner et al. (2004, p.86), I have concluded that the descriptive evaluation method is the best choice at the time of writing. I also consider the designed artifact to be innovative enough for the descriptive evaluation method.

Hevner et al. (2004, p.86) describes these two types of descriptive evaluation:

- Informed Argument: Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artifact's utility.
- Scenarios: Construct detailed scenarios around the artifact to demonstrate its utility.

In this project, the designed artifact is going to be evaluated both by informed arguments and scenarios.

Informed Arguments

This section will present informed arguments to demonstrate the proof of concept artifact's utility. The arguments are cited from the article "*How Will We Interact with the Web of Data?*" by Heath (2008), which presents arguments that this study is based upon. The focus of his article is to "*discuss some ways in which our interaction with the Web of data might differ from how we interact with the established Web of documents*".

The following arguments discussed by Heath (2008) can be used to justify the utility of the SemanticGeoBrowser:

- Discussing the Web of machine-readable data: …"*without a human somewhere in this process to reap the rewards of these new capabilities, the endeavour is meaningless.*"
- "*If we're to fully exploit the challenges and opportunities of a Web of data, we need to move beyond the initial phase and work to understand how this changes the Web's user interaction paradigm.*"
- "*In the Semantic Web, you can't assume you have control over how the information you publish will be presented — it's just data.*"
- …"*concentrate first on publishing relevant, high-quality data, and let others build the views they want rather than those that someone else assumes they need.*"
- …"*in the Web of data, no one can control with any degree of certainty the sources with which their data is integrated — enabling serendipitous reuse is exactly the point!*"
- …"*data published in the Web in a reusable form enables new views that have value beyond the sum of the parts and that the original creators might not have anticipated in advance.*"
- …"*The machines' job is then to assemble this data into a coherent view, ready for human consumption.*"
- "*Of far greater relevance than the documents themselves are the things described in those documents — the people, places, and concepts.*"
- "*It's at the level of "things" that browsers for the Web of data should operate. Providing simple browsers for RDF triples, and the documents*

in which they're published, is one option for enabling people to interact with this information space."

- Discussing the earliest Semantic Web browsers: *"it rather misses the point. The one-page-at-a-time style of browsing, which we know well from the Web of documents, would make nothing of the potential we now have for integrated views of data assembled from numerous locations.*"
- "*Semantic Web browsers must not simply echo the underlying representation of the data. Instead, they must treat "things," in the broadest sense, as first-class citizens of the interface. A particular thing of interest should take center stage, with the browser assembling relevant information seamlessly behind the scenes.*"
- …"*the thing of interest is of greater importance, and specific documents simply supply fragments of data that together make up a broader picture.*"
- "*Conventional browsers have largely failed to deliver on the original vision of the Web as a read/write medium.*"
- "*Browsers for the Semantic Web, which I suggest we call "thing browsers," have an opportunity to enable a far greater degree of direct manipulation in their interfaces. Different types of objects afford different types of actions, and knowing the type of object on which the user is focused should let browsers provide a menu of actions specialized for this object type, and perhaps even adapt these according to the context.*"
- Discussing a Semantic Web browser's ability to conduct action: …"*without any of these functions having been explicitly listed as actions that can be invoked on these individuals. Instead, the Semantic Web at large can provide the necessary knowledge and services on which to offer such functionalit…*"
- "*Clearly, a Web of data can't offer direct manipulation of real-world things, such as cars and dogs, which are not, and never will be, online. However, in a Web where we can explicitly reference anything, not just documents, there's great potential to reduce the degree of indirection in Web interfaces. We no longer have to refer to Web pages about things but can refer to the things themselves.*"

- "*In case there was any doubt, this is no overnight endeavor but a trend that will take years to be realized and could take many different forms.*"
- "*Accepting the shift from document to thing, and from predefined views to those assembled dynamically, won't just require completely new interfaces but also several changes to the interaction widgets in interfaces with which we're already familiar.* "
- "*All the data available on the Web about London can't feasibly be presented in one interface; users will need to decide which sources to add in depending on their current task or context, or will need the browser to make this decision intelligently for them...*"
- "*… This functionality becomes even more critical if automated reasoning is carried out on Semantic Web data, creating knowledge that wasn't previously explicit in any of the individual data sources. How to manage the assembly of these data sources becomes a critical issue.*"
- "*Key to developing Web of data browsers will be look-up services such as Sindice, which provide a means to find other RDF documents on the Semantic Web that mention a particular thing. This kind of service might help ensure that the user experience is coherent — that is, that it includes all data the user expects it to. However, ensuring that a particular view of data is useful is another issue.* "
- "*Any system aiming to integrate heterogeneous data on an ad hoc basis and present this to users will need to adopt sophisticated models of relevance, quality, and trust that are sensitive to the user's current task and its context. How that might be achieved is a question for another day.*"

Scenarios

This section presents two scenarios to demonstrate the artifact's utility.

Scenario one

A tourist is traveling around the world on a cruise ship. In ten minutes time the cruise will arrive to the city of Bergen in Norway. The ship is after its schedule with an entire day so the captain decides to make some last minutes changes.

The passengers are informed that they will only have three hours to spend in the city. Our tourist knows that he does not have one minute to waste ashore. He decides to drop his plans about sightseeing in random city streets and only focus on reaching a nearby mountain top to get an overview over the location. To reach his goal, there will be no time to stand in line at a local tourist information office. He decides to find a suitable hiking route superfast on his personal internet device. Instead of searching the Web of documents for web pages with different presentations of hikes, he recently discovered a web application that lets him browse the Semantic Web for tourist information in any area of his choice. The application was already bookmarked because of its support of recognizing things within the tourist domain. He recognizes the user interface as the SemanticGeoBrowser appears on the screen. The map is navigated to the area of interest and the predefined option "Hikes" is selected as the type of thing to look for. A search is conducted and a result is presented. He examines the visual result on the map and taps on some hikes to study their details in the informative fact box. The suggested degree of difficulty is helpful and makes him take a quick decision. A suitable hike is selected and the ship hasn't even docked yet.

Scenario two

Our tourist from scenario one has reached his goal and has now taken a short break on a view point on his way back down to the city of Bergen. While he rests, he figures he has time to visit a café before getting back on the cruise ship. He flips out his internet devise and enters the SemanticGeoBrowser once again. This time, he selects a specific data source that is known for its updated tourist information. A list is presented to him with types of things to search for. He selects "Café", but remembers at the same time that he should send a post card to each of his sisters while still in Bergen. The web application supports the option of finding things that is close to other types of things, which enables him to search for all cafés that is nearby a post office. A search is made and a result is presented. A category feature is used to mark the relevant options with colors. Cafés are marked with the color red and the post offices are marked with blue. He hides the items from the first search result and conducts a new one. How

many pubs area there in the Bergen area? He is just curious. A result is presented. "That many…" he thinks before starting to walk again. Maybe he has time to visit one of them as well.

As described through the two scenarios above, the tourist draws logical conclusions from premises presented on the map like can be archived through the design of the SemanticGeoBrowser. The tourist would have struggled to draw the same conclusion by reviewing the same content presented as plain RDF triples.

Discussion

In this design-science research study, I have made an effort to answer the research question "*how can we build a user-friendly Semantic Web browser that enables its users to discover and explore geospatial things described in the Web of data?*" by constructing a proof of concept artifact. In order to support this research question, a hypothesis was formed, which claims that "*a thing browser, like the SemanticGeoBrowser, will make it easier for humans to discover and explore geospatial things described in the Web of data*". In the scope of the research question and the hypothesis, a set of system requirements was formulated to estimate what it would take for the proof of concept artifact to solve the problem.

In order to determine this study's and the artifact's level of success, I would like to review the results in the light of the requirements for effective design-science research. These was introduced through the seven guidelines by Hevner et al. (2004) in chapter 3, named "Method".

In the light of guideline number one, "*Design as an Artifact*", the outcome of this design-science research has resulted in viable artifact. Like pointed out in this guideline, the outcome of the conducted research has not resulted in a full-grown information system that are ready for use in practice, but rather a proof of concept artifact that is capable of doing what it is intended to do. The SemanticGeoBrowser was designed with the purpose of presenting an innovative

solution to the problem addressed by the research question of this study. All though the idea of a Semantic Web browser, or a thing browser, is a well-known idea within the Semantic Web community, I have attempted to design an innovative artifact by putting existing technologies and ideas into practice in order to present my own contribution on the matter.

In the light of guideline number two, "*Problem Relevance*", the problem addressed by the research question of this study is a *real* problem. It is both theoretically and practically addressed by other researchers and is therefore to be considered as relevant to the Semantic Web community. As expressed in chapter 1, named "Introduction", in subchapter "Background", the problem of making it easy for humans to interact with the Semantic Web, is a wide problem. The focus of this study has however been influenced by questions like the one raised by Heath (2008), "*What should a Semantic Web browser look like?*", which narrows the scope of a foundational problem. The research question and hypothesis has been defined with the goal of contributing to the Semantic Web community's ongoing search process of finding solutions to a difficult problem. While the research question represents the before state of this study, the hypothesis represents the goal state of the search process.

In the light of guideline number three, "*Design Evaluation*", the SemanticGeoBrowser was evaluated in terms of functionality in which it was able to demonstrate. The functionality is represented by a set of system requirements, which was proposed by me in chapter 1. During the iterative development process of the artifact, the evaluation phase (in each iteration) was based on the fulfillment of the proposed system requirements. The iterations continued until all the system requirements were fulfilled. According to Hevner et al. (2004, p.85), "*a design artifact is complete and effective when it satisfies the requirements and constraints of the problem it was meant to solve.*" The proof of concept artifact was then evaluated by one of the methodologies proposed by Hevner et al. (2004, p.86), which is available in the design-science knowledge base. The designed artifact was evaluated after the "descriptive" evaluation method. This is firstly because I consider the SemanticGeoBrowser to be

innovative enough to justify this selection, but this research project was also limited on time at the point of the conducting the evaluation. The descriptive evaluation method is considered to be sufficient when it comes to demonstrate the SemanticGeoBrowser's utility.

In the light of guideline number four, "*Research Contributions*", the outcome of this study has resulted in one clear and verifiable research contribution, namely the design artifact. The SemanticGeoBrowser, which applies existing knowledge in a new and innovative way, represents a contribution to the Semantic Web community, which has a wide heretofore unsolved problem on its agenda.

In the light of guideline number five, "*Research Rigor*", the theoretical foundations, research methodologies, and technology has during this study been selected at the best of my ability. The selected scientific literature has been relevant to the topic of this thesis and the methods applied to this design-science research is believed to be rigorous and relevant as they have been suggested by Hevner et al. (2004). When it comes to the construction of the artifact, the selected technology and the code written has proven to be sufficient for solving the tasks represented by the system requirements.

In the light of guideline number six, "*Design as a Search Process*", the SemanticGeoBrowser was created through the use of the Generate/Test Cycle, described in this guideline. Even though there were too many system requirements in this project to make it feasible to focus on each of them through several iterations, the repetition of measuring the artifact against the system requirements gave a sense of progress.

In the light of guideline number seven, "*Communication of Research*", the outcome of this design-science research study will be made available to use by anyone with access to the Web. This master thesis document will make the results available to non-technical audiences, while the source code of the proof of concept artifact will be made available for audiences with more technical skills. The result of this study will therefore be available for practitioners to take

advantage of the benefits offered by the SemanticGeoBrowser, and researchers will have the opportunity to use this work for further extension and evaluation.

Chapter 6

6 Conclusion and future work

This chapter will present a short summarization and a conclusion of this design-science research study.

Conclusion

This design-science research study has demonstrated a concept of a design artifact which addresses a problem recognized by the Semantic Web community. The problem is that it is difficult for humans to browse the Semantic Web with a general purposed web browser, which was originally designed for browsing a Web of interlinked hypertext documents. Because the Web of data is woven together with RDF triple statements that are designed for computer processing, there is a need for Semantic Web browser tools that can help humans to explore and make sense of the data.

The SemanticGeoBrowser is an effort to join the Semantic Web community in a search for effective designs that can contribute to solving this issue. In order to narrow the scope, the design artifact focuses on browsing RDF statements that describe things as geospatial data.

The artifact was developed and evaluated through an iterative design-search process, following the requirements for effective design-science research, presented by Hevner et al. (2004). When it fulfilled a set of proposed system requirements, the iterative search process ended, and the designed artifact was evaluated after a “descriptive” evaluation method from the design-science knowledge base.

In order to determine this study’s and the artifact’s level of success, the outcome of the research was reviewed against the seven requirements for effective design-science research, presented by Hevner et al. (2004). In regards to this review, I have concluded that the conducted research meets these seven

requirements, and this study is therefore considered as a valid design-science research.

In regards to the designed artifact's level of success, the SemanticGeoBrowser meets the proposed system requirements that were created in order to answer the research question. As the two descriptive evaluations made suggest that the proof of concept artifact demonstrates its utility design, which also align with the supporting hypothesis made in this study, the research question is hereby considered answered. The SemanticGeoBrowser demonstrates a prototype solution on how to build a user-friendly Semantic Web browser that enables its users to discover and explore geospatial things described in the Web of data.

Future work

This section will present some points on future work or research:

- The Semantic Web community will need more proof of concept artifacts in order make progress on this issue.
- While the SemanticGeoBrowser is a working prototype, it is in need of optimization and further development. The source code is open for anyone to use and made publically available through GitHub.com.
- The SemanticGeoBrowser can also be subject for further evaluation by other evaluation methods from the knowledge base from design-science research.
- The Web of data is in need of more quality data, which describe things with property facts.

References

Aaberge, T., 2012a. Interpretations of Formal Languages. *Western Norway Research Institute (WNRI)*.

Aaberge, T., 2011. Ontology Construction: Background and Practices. *Western Norway Research Institute (WNRI)*.

Aaberge, T., 2012b. *Semantisk Sognefjord.no*, Sogndal. Available at: http://www.vestforsk.no/rapport/semantisk-sognefjord.no.

Amundsen, G., 2009a. Kritisk til kartverkets frislipp. *Aftenposten*. Available at: http://www.aftenposten.no/digital/Kritisk-til-kartverkets-frislipp-5590538.html [Accessed November 16, 2012].

Amundsen, G., 2009b. Statens kartverk frigir kartene sine. *Aftenposten*. Available at: http://www.aftenposten.no/digital/nyheter/Statens-kartverk-frigir-kartene-sine-6623090.html [Accessed November 16, 2012].

Andersen, R., 2009. About us. *Norwegian Mapping Authority*. Available at: http://www.statkart.no/eng/Norwegian_Mapping_Authority/ [Accessed November 15, 2012].

Anon, 2013a. About Sindice. *Sindice.com*. Available at: http://sindice.com/main/about [Accessed January 14, 2013].

Anon, 2013b. Axiom. *Wikipedia, the free encyclopedia*. Available at: http://en.wikipedia.org/wiki/Axiom [Accessed January 25, 2013].

Anon, 2012a. Encoded Polyline Algorithm Format. *Google Developers*. Available at: https://developers.google.com/maps/documentation/utilities/polylinealgorithm [Accessed November 21, 2012].

Anon, 2013c. Inference. *The Free Dictionary*. Available at: http://www.thefreedictionary.com/inference [Accessed February 4, 2013].

Anon, 2012b. Same Origin Policy. *World Wide Web Consortium*. Available at: http://www.w3.org/Security/wiki/Same_Origin_Policy [Accessed October 29, 2012].

Anon, 2013d. Sindice Cache API. *Sindice.com*. Available at: http://sindice.com/developers/cacheapi [Accessed February 1, 2013].

Anon, 2013e. Sindice Search API. *Sindice.com*. Available at: http://sindice.com/developers/searchapiv3 [Accessed February 1, 2013].

Anon, 2013f. SIREn: Efficient semi-structured Information Retrieval for Lucene. *Sindice.com*. Available at: http://siren.sindice.com/ [Accessed March 4, 2013].

Anon, 2013g. XML. *Wikipedia, the free encyclopedia*. Available at: http://no.wikipedia.org/wiki/XML [Accessed January 20, 2013].

Anon, 2009. XMLHttpRequest. *World Wide Web Consortium*. Available at: http://www.w3.org/Security/wiki/XMLHttpRequest [Accessed October 29, 2012].

Anon, 2013h. XSLT. *Wikipedia, the free encyclopedia*. Available at: http://en.wikipedia.org/wiki/XSLT [Accessed January 20, 2013].

Berners-Lee, T., 2006. Linked Data. *World Wide Web Consortium*. Available at: http://www.w3.org/DesignIssues/LinkedData.html [Accessed December 7, 2012].

Berners-Lee, T. et al., 1998. RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax. *The Internet Society*. Available at: http://www.ietf.org/rfc/rfc2396.txt [Accessed December 17, 2012].

Berners-Lee, T., The WorldWideWeb browser. *World Wide Web Consortium*. Available at: http://www.w3.org/People/Berners-Lee/WorldWideWeb.html [Accessed January 2, 2013].

Berners-Lee, T. & Cailliau, R., 1990. WorldWideWeb: Proposal for a HyperText Project. *World Wide Web Consortium*. Available at: http://www.w3.org/Proposal.html [Accessed December 2, 2012].

Berners-Lee, T., Hendler, J. & Lassila, O., 2001. The Semantic Web K. Aberer et al., eds. *Scientific American*, 284(5), pp.34–43. Available at: http://www.nature.com/doifinder/10.1038/scientificamerican0501-34.

Brombach, H., 2012. Frigitte kartdata bare egnet for papirkart. *Aller Media AS*. Available at: http://www.digi.no/894644/frigitte-kartdata-bare-egnet-for-papirkart [Accessed November 16, 2012].

Chapman, S., 2012. Ajax - Asynchronous or Synchronous. *About.com*. Available at: http://javascript.about.com/od/ajax/a/ajaxasyn.htm [Accessed November 9, 2012].

Chng, P., 2008. Decoding Google Maps Encoded Polylines using PHP. *Unitstep.net*. Available at: http://unitstep.net/blog/2008/08/02/decoding-google-maps-encoded-polylines-using-php/ [Accessed November 23, 2012].

Cyganiak, R. et al., 2012. RDF 1.1 Concepts and Abstract Syntax. *World Wide Web Consortium*. Available at: http://www.w3.org/TR/2012/WD-rdf11-concepts-20120605/ [Accessed December 29, 2012].

Cyganiak, R., 2011. URIs vs. IRIs in Semantic Web standards and tools. *Semanticweb.com*. Available at: http://answers.semanticweb.com/questions/13076/uris-vs-iris-in-semantic-web-standards-and-tools [Accessed December 29, 2012].

Denning, P.J., 1997. A New Social Contract for Research. *Communications of the ACM*, 40(2), pp.132–134. Available at: http://dl.acm.org/citation.cfm?id=253755.

Engeland, S., 2012. Kartverket frigir mer kartdata. *Norwegian Mapping Authority*. Available at: http://www.kartverket.no/Kartverket+frigir+mer+kartdata.d25-SwZHQ0x.ips [Accessed November 15, 2012].

Genesereth, M.R. & Nilsson, N.J., 1987. *Logical Foundations of Artificial Intelligence*, San Mateo, CA: Morgan Kaufmann Publishers.

Gruber, T.R., 1993. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2), pp.199–220.

Hausenblas, M., 2009. Exploiting Linked Data to Build Web Applications. *IEEE Internet Computing*, 13(4), pp.68–73. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5167270.

Heath, T., 2008. *How Will We Interact with the Web of Data?*, IEEE. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4620099.

Heath, T. & Bizer, C., 2011. *Linked Data: Evolving the Web into a Global Data Space* 1st ed. J. Hendler & F. Van Harmelen, eds., Morgan & Claypool. Available at: http://linkeddatabook.com/editions/1.0/.

Hevner, A.R. et al., 2004. Design Science in Information Systems Research. *MIS Q.*, 28(1), pp.75–105. Available at: http://dl.acm.org/citation.cfm?id=2017212.2017217.

Hirsch, T., 2011. WPS-services at Statens kartverk. *Norwegian Mapping Authority*, p.5. Available at: http://www.statkart.no/filestore/Landdivisjonen_ny/Kart_og_produkter/tvisningstjenester/WPS_en_EN.pdf.

Ippolito, B., 2005. Remote JSON - JSONP. *bob.ippoli.to*. Available at: http://bob.ippoli.to/archives/2005/12/05/remote-json-jsonp/ [Accessed October 29, 2012].

Karger, D. & Schraefel, M.C., 2006. The Pathetic Fallacy of RDF. *Semantic Web User Interaction Workshop (SWUI 06)*. Available at:

http://swui.semanticweb.org/swui06/papers/Karger/Pathetic_Fallacy.html [Accessed January 7, 2013].

Machi, D., 2012. Getting Jiggy with JSONP. *Dojo Toolkit*. Available at: https://dojotoolkit.org/documentation/tutorials/1.6/jsonp/ [Accessed October 29, 2012].

Simon, H.A., 1996. *The Sciences of the Artificial*, MIT Press. Available at: http://books.google.com/books?id=k5Sr0nFw7psC.

Stadler, C., 2012. About. *Agile Knowledge Engineering and Semantic Web (AKSW)*. Available at: http://linkedgeodata.org/About [Accessed October 29, 2012].

Stadler, C. et al., 2012. LinkedGeoData: A Core for a Web of Spatial Open Data. *Semantic Web Journal*. Available at: http://linkedgeodata.org/Publications.

Stadler, C. & Knibbe, F., 2011. Bounding Box geographical filter. Available at: https://groups.google.com/forum/?fromgroups=#!topic/linked-geo-data/BXuH45-IXdU [Accessed May 5, 2012].

Sternberg, R.J., 2009. *Cognitive Psychology*, Wadsworth.

Tsichritzis, D. & Metcalfe, R.M., 1998. *"The Dynamics of Innovation" in Beyond Calculation: The Next Fifty Years of Computing*, New York: Copernicus Books.

Tummarello, G., Delbru, R. & Oren, E., 2007. Sindice.com: Weaving the open linked data K Aberer et al., eds. *Lecture Notes in Computer Science The Semantic Web*, 4825(4825), pp.552–565. Available at: http://iswc2007.semanticweb.org/papers/547.pdf.

Verborgh, R., 2012a. Bringing reasoning to the Web. *Multimedia Lab, Ghent University*. Available at: http://reasoning.restdesc.org/ [Accessed October 29, 2012].

Verborgh, R., 2012b. Semantic Web Reasoning With EYE, Executing rules. *Multimedia Lab, Ghent University*. Available at: http://n3.restdesc.org/rules/executing-rules/ [Accessed October 29, 2012].

Wikipedia, 2012. Deductive reasoning. *Wikipedia, the free encyclopedia*. Available at: http://en.wikipedia.org/wiki/Deductive_reasoning [Accessed November 13, 2012].

Zalewski, M., 2009. Same-origin policy for XMLHttpRequest. *Google*. Available at: http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy_for_XMLHttpRequest [Accessed October 29, 2012].

Appendices

The source code of the SemanticGeoBrowser has been made publically available for anyone to use through GitHub.com, a web-based hosting service for open source projects. This code can be found and reviewed at the following URL: https://github.com/hustveit/SemanticGeoBrowser

A live test version of the SemanticGeoBrowser can be found at the following URL: http://SemanticGeoBrowser.com

The code of the scripts programmed for lifting data in this master thesis project, and the lifted data, is also available for review through GitHub.com, by following this URL:
https://github.com/hustveit/SemanticGeoBrowser.Data

The lifted data is also available through this SPARQL endpoint:
http://sognefjord.vestforsk.no:8890/sparql