University of Bergen

Master Thesis

Metadata made easy - Adding metadata by means of natural language

Author:
Eivind Eidheim Elseth

Supervisor:
Csaba Veres

in the

Semantic and Social Information Systems

Department of Information Science and Media Studies

June 2013

"It depends upon what the meaning of the word 'is' is"

– Bill Clinton

UNIVERSITY OF BERGEN

Abstract

Faculty of Social Sciences

Department of Information Science and Media Studies

Masters degree

Metadata made easy - Adding metadata by means of natural language

by Eivind Eidheim Elseth

The rapid increase in the amount of information generated highlights the need for better strategies for searching for relevant content. Using Semantic Web technologies we can search using concepts and their properties and relations to each other. To enable the use of Semantic Web technologies documents on the Web need to contain metadata which can describe their content in a way readable by computers.

This thesis will describe the development of MaDaME, a prototype system that will help users add this metadata to their documents using only natural language. The tool will utilize WordNet to disambiguate natural language, and as an intermediary to find mappings into formal ontologies. The thesis will examine the feasibility of using the hypernym relation, and the hyponyms of hypernyms, for finding mappings to more general categories when no direct mapping exists.

Acknowledgements

I would first and foremost like to thank my supervisor Csaba Veres. Our talks during the course of my work on the thesis were inspiring. Your ideas and knowledge of the field was crucial during the development of MaDaME, and the tool would have been what it is without it. I would also like to thank you for your feedback and recommendations on writing this thesis.

Thank you, to all my friends on the master's course, for good technical and academic discussions and also for lunches, coffee breaks, and making the

Last but not least I would like to thank my wife, who bore with me when I was unbearable, and supported me through all my work.

Contents

List of Figures

List of Tables

List of Listings

Nomenclature

CSS	Cascading Style Sheets
HTML	HyperText Markup Language
IRI	Internationalized Resource Identifier
OWL	Web Ontology Language
RDF/XML	An XML representation of RDF
RDFa	Resource Description Framework in attributes
RDF	Resource Description Framework
SUMO	Suggested upper merged ontology
SUO	Standard Upper Ontology
Synset	A set of words with similar meanings.
URL	Uniform Resource Locator

Chapter 1

Introduction

The Internet is now an ingrained part of our everyday life, and the amount of content and services that are available through it is growing at an ever increasing rate. For all this information to be of use to humans it is necessary to have some interface through which to access the parts of it that are relevant to us. Shirky [2007] tells of the early attempts to structure the Web, using ontologies and hierarchies created by experts. This soon got clunky as the number of documents increased. Trying to organize the content of the Internet fell out of favor and was replaced by searching for information using keywords. It is this phase of organizing information we are in now.

Berners-Lee et al. [2001] suggested that we could do better than keyword matching. Keyword searches mean that users have to look though lists of results and compare documents to find the best match. It can also mean following leads and clicking through links to find the content one is after. Instead of forcing users to go through this process, this new idea was to enrich the documents we put on the Web with metadata that could be read and reasoned about by computers. By doing this we could move the tedious task of siphoning though Web sites looking for relevant information from users over to specialized software agents that could collect information on the topic and return the answer to the user.

This does create some extra work for content creators on the Internet. Adding metadata to content is not a trivial task. The W3C recommends using RDFa to add metadata [Pemberton et al., 2008]. But to use RDFa the user not only needs to know the syntax of RDFa itself, but also which ontologies exists that contain the meaning the creator wants

to convey, and knowledge about how those ontologies are structured to use them in the correct way.

Ontology is the philosophical discipline of finding out which things exist, the manner in which one can say that these exist and how these can be categorized. When talking about ontologies in the context of the Semantic Web, an ontology can be explained as a collection of things that one can describe using the ontology, and how these things can relate to each other.

1.1 Motivation

According to Gantz and Reinsel [2011] the amount of information in the world now doubles every two years. Should this trend continue, we are going to need better ways of searching through all the information that is generated. We are going to need to be able to search for content in a way that lets us search for concepts, not only keywords. What we want is to have linked semantic data that help move the burden of finding relevant content from us over to machines.

At the moment, adding semantic markup to Web sites is not feasible for most content creators on the Web. Adding proper metadata does not only mean that you need to know HTML, but also that you need to understand the concept of ontologies, and know of the different ontologies that exist. In addition you need to know of the implementation of RDFa, microdata or microformats as well as the content of the specific ontologies you intent to use on your site.

Adding metadata should not really be that hard. Humans are good at knowing what things are. We know what the things on Web sites mean, and which concepts they are meant to convey. Natural language is mainly ambiguous to computers, not human beings. I want to create a tool that lets users use their knowledge of natural language and the concepts it conveys to create metadata by disambiguating the language for the computer, and letting it create the mappings and the elements on the Web page.

The aim of this thesis will be to develop a prototype artefact that will let users add metadata to Web pages using natural language. The prototype should map to the schema.org

ontology, as using this ontology can increase the visibility of the Web page in search results in the most popular search engines. The tool should also be able to express meaning in other ontologies. One no longer believes that one single ontology can capture all of the concepts and relationships one wants to describe, and instead develops different ontologies to cover different domains. Creating a tool that can map to different ontologies then can help create a rich Web of interconnected data.

The prototype tool will be created using Node.js[1], a JavaScript platform based on Google Chromes JavaScript runtime, and using MongoDB[2] as a database solution. JavaScript is a good language to rapidly create prototypes, and the flexibility and simplicity inherent in using the same language server side, client side and in database communication, inspired me to try learning and using these tools for the thesis.

I choose to develop using node.js because it is a new and exciting technology that would make it possible to write the entire application using one programming language for front-end, server side and as the database query language.

Creating a Web application allows the work to consist both of back end server side components, and front-end work.

1.2 Research question

The main research question of this thesis is:

"Is it possible to create a tool which allows naïve users to easily add metadata to their Web sites using natural language?"

In this context we should understand "naïve users" as users who do not have any training in the use of Semantic Web technologies, and who do not have any knowledge about the ontologies used for testing in this thesis.

To answer this question I will develop a tool called MaDaME, a play on the goal "MetADAta MAde Easy". The tool will consist of a logical backend tied to a Web front-end, and the different modules will be created in an iterative way to get working proofs of concepts for testing early.

[1] `http://nodejs.org`
[2] `http://www.mongodb.org`

There are several sub questions that will need to examined to answer this question.

- How should users pick the parts of a Web page they want to add metadata to, and find the concepts it describes using natural language?
- Is WordNet suitable for representing disambiguated concepts from natural language in a way that will allow us to map these concepts to formal ontologies?
- How should an algorithm be implement to finds mappings from the natural language concept to types in formal ontologies in a way that preserves the semantic content of the concept?
- Is it possible to add metadata to Web pages in such a way that it does not change the way the page is rendered by browsers?

In addition I will also need to solve the technical problems of how to import and export Web pages from the artefact, and how to save the resulting documents on the server.

1.3 Target audience

The main goal of the work with MaDaME is to lower the barrier of entry for enhancing Web sites with metadata. I want to make metadata accessible to content creators on the Web that do not know enough about Semantic Web technologies to add this data them selves.

Due to availability of English natural language resources this thesis will focus development on mapping English text, and limit the target group to users who write content in English. I will also assume that the users control the HTML of their Web pages, and that being provided with a new HTML document will allow the user to update their documents.

Chapter 2

Theory

This chapter will give a review of some of the research done on user generated metadata and ontologies. It will also introduce the core vocabularies and tool will be used in the development of the artefact, and describe their origins and the state of research tied to them at the present.

2.1 Ontology and folksonomy

One of the central concepts in Semantic Web technology is that of the ontology. In philosophy Ontology is the branch dealing with the study of which things 'exists', and if it is possible to categorize these things. For artificial intelligence Gruber [1993] explained it as "an explicit specification of a conceptualization". That is, than one commits to a given conceptualization of the domain in question, and formalize how one describes and reason about these conceptualizations.

Guarino [1998] provided a description, which might be simpler, in which an ontology was described as a shared vocabulary concerning some specific domain where the assumptions regarding each term in the vocabulary was made explicit. Pretorius [2004] talks about information sharing and reuse as one of the advantages of using ontologies to capture conceptualizations. Tools that are committed to the same ontology can easily share their knowledge stores, since they talk about the same things using the same explicit definitions of those things. Reusing information becomes easier since one can use existing knowledge

stores by committing to the ontology they are described with, or by finding mappings from that ontology to the one you want to use.

Shirky [2007] criticizes the use of ontologies as a way of trying to enforce a structure on something that is by nature unstructured. He instead pushes the idea of common tagging. Part of the reason he criticizes the ontology approach is that it seem improbable that experts can know the needs of all the users a priori, and therefore that every ontology will prove to be inadequate. The artefact developed for this thesis will use an approach similar to that of tagging, in that it will let the user provide keywords which describe the text. These natural language keywords can then be used to map to ontologies.

While ontologies are formally constructed taxonomies, folksonomies are informal taxonomies generated by collecting tags or annotations from collaborative tagging systems on a given platform[Tang et al., 2009]. Mika [2005] has given a more formal definition of folksonomy where he sees a folksonomy as a set of tags T, $T \subseteq A \times C \times I$, where A is the set of users tagging, C is the set of tags, and I is the set of objects being tagged. Gruber [2007] suggested that one should add the source of the tag, and some kind of rating system to help filter out junk tags. Kim et al. [2008] on the other hand suggests removing the objects being tagged from the ontology, seeing that the objects that are being described are not part of the tool to describe them. Instead they like Gruber want to add the source of the tag, the number or times a tag occur, and the tagging behavior of each user. Bang et al. [2008] explains the difference between ontologies and folksonomies by classifying them as a priori and a posteriori annotations. That is, ontologies are created by experts as ways on conceptualizing a domain, folksonomies on the other hand are samples of how people speak or think about a domain. Folksonomies grew as a subject of research as it became popular for users to tag content on the Internet with keywords they felt were relevant.

For users tags are convenient, since adding additional tags can make it easier for humans to search and browse collections. This is especially true for multimedia content, which we do not yet have good tools for searching in [Weinberger et al., 2008]. Tags provide metadata about content in a way that makes sense to humans. From an information retrieval perspective this is interesting since it means that humans in some way add meaning to the content. There however are several problems with using tags as the basis of a Semantic Web. Tang et al. [2009] mention several. Tags are supposed to be written

in natural language, and natural language has words that are synonyms (words that are written in different ways, but mean the same), homonyms (words with different meanings that are written in the same way), or polynyms (a word that can have several meanings) making it unsuitable for computer reasoning since they are ambiguous [Passant and Laublet, 2008]. As Golder and Huberman [2005] mentions, users also operate on different levels of abstraction, which can make it harder to find interesting resources. In addition to this comes the problem of non-dictionary words, both new, or compound words, or simply words that have been misspelled[Tonkin and Guy, 2006].

There has been done a lot of research into how one can lift semantic data out of these unstructured tags. Golder and Huberman [2005] has done research into the statistical analysis of tags. The analysis done here show that there seems to develop vocabularies of frequently used tags. This might help diminish the effect of misspelled and nonsense words. Similar findings were also reported by [Shirky, 2007]

There has also been done research into automatic clustering. Mika [2005] created clusters by creating weighted graphs, and compared using tag concurrence and actor interest as weights. Brooks and Montanez [2006] has done work on categorizing blog entries by tags, to see if concurrence of tags indicated similar content. Using the most common tags did give some results, but only broad categories. The results were not better than extracting words that were given asserted to be relevant for the category.

[Tang et al., 2009] tries to go further that clustering tags, and tries to build an hierarchical model from a folksonomy. They use a probabilistic model that takes into account the frequency and concurrence of tags and try to generalize it to an ontology. The method does get good results in creating the hierarchy, but does also show some inappropriate sub/super category inferences.

Weinberger et al. [2008] suggests a method for removing ambiguity from tags, by suggesting additional tags to the user when the tag entered can belong to one of several distinct sets

Lifting semantic information from tags is an attempt at merging the. Veres [2011] suggested another method one could use to merge these two approaches. The idea presented involved disambiguating the tag at insertion time. This idea and the implementation of a system that disambiguates tags will be discussed later in the chapter.

2.1.1 RDF

On the Semantic Web, ontologies are represented using collection of RDF triples. These triples are in the form of <subject, predicate, object>, much like simple declarative sentences [Berners-Lee et al., 2001]. Let us suppose we wanted to say that I know my supervisor Csaba. An informal triple describing this could be

```
<Eivind, knows, Csaba>
```

"Eivind" is here the thing one talks about, the thing one wants to describe in some way. The predicate "knows" is a relation the subject of the triple has to some other thing. The object is the thing with which the subjects relates with respect to the predicate, in this case it is the thing that is known. When representing triples using RDF one has to use a more stringent form. The reason one describes things in RDF form is to give computers some way of reasoning about them, so there must be some way to separate this "Eivind" from the other "Eivind"s out there, there are after all a lot of people named Eivind. Disambiguation is achieved by using IRIs to represent the resources one wants to talk about. Using IRIs makes it possible to separate the thing one talks about from all the other similarly named things, since each should uniquely identify a single thing. The IRIs will also often be URLs that point to a location that describe the resource one talks about. But the only requirement for choosing an IRI is that it is a unique identifier.

An example using IRIs to describe each resource could be as such:

```
<
  "http://eivind.elseth.me",
  "http://xmlns.com/foaf/0.1/knows",
  "http://csabaveres.net"
>
```

Here the first IRI identifies whom we mean by "Eivind", namely "http://eivind.elseth.me". The second IRI identifies what we mean by knows, and the third who it is that Eivind knows. A machine parsing this can dereference the URLs to learn more information about the resources.

The subject and predicate of a triple must always be resources represented by IRIs. The objects can be either resources or literals, depending on what the predicate of the triple is. A predicate describing something like "has name" would likely point to a string literal containing the name of the subject[Hebeler et al., 2009].

2.2 WordNet

In a written dictionary the most efficient way to organize data is to list the words alphabetically, since the act of finding words is the most labor intensive part. As long as the dictionary is tied to an analog form it is hard to structure the content otherwise as it would make finding words to difficult. With the advent of computer dictionaries, the act of looking up words is no longer time consuming or difficult, and the possibility to experiment with new structures for the dictionary became possible.

There are several differences between ordinary dictionaries and WordNet. The motivation behind WordNet was to create a dictionary that categorized words by the concepts they represented. To accomplish this the structure was based on earlier psycholexicologic research[Miller et al., 1990].

One of the ways this psycholexicological background comes to sight is in that WordNet separates words into four syntactical categories: nouns, verbs, adjectives and adverbs[Miller, 1995]. This was based in part on work done by Fillenbaum and Jones [1965], which showed that participants would most frequently associate words with other words from the same syntactical category.

This way of categorizing the words is able to take advantage of the fact that the different syntactical categories have different semantic structures. This thesis will use the noun category of WordNet, and benefit from its topological hierarchy.

The word 'word' is ambiguous as it can be used to describe the representation of a word, and its underlying semantics. Natural language is built on conventions that tie together utterances or symbols, with some thing or idea. A symbol or utterance is the form of a word, while the thing or idea it represents is the words meaning. This idea can be represent as in table 2.1 (see page 10). The table shows a lexical matrix, which means to make explicit the relation between form F and meaning M. The forms and meanings

are linked using entries $E_{(x,y)}$ that would state that meaning M_x has the form F_y. It also shows how both a single form can refer to several meanings, and how a single meaning can be represented using several forms. Within the categories mentioned WordNet then tries to organize the words not by form, but by similarity of meaning.

Word Meanings	Word Forms						
	F_1	F_2	F_3	.	.	.	F_n
M_1	$E_{1,1}$	$E_{1,2}$					
M_2		$E_{2,2}$					
M_3			$E_{3,3}$				
.				.			
.					.		
.						.	
M_m							$E_{m,n}$

TABLE 2.1: A lexical matrix showing the relation between the forms and meanings of words, from Miller et al. [1990]

This organization is done by grouping words that are synonyms into sets. Figure 2.1 shows these groupings for the word "dog". These sets of synonyms (synsets) car be described by enclosing one or more word forms in curly brackets. The synset {dog, domestic dog, Canis familiaris} can be used to describe the common dog. When precision is not the point a short form like {dog} can be used to describe the same meaning. Later, when referring to a specific synset the form dog#n#1 will be used. The first part here refers to the form of the word. The second part is the grammatical group of the synset, in this case n for noun. The third part is the numbering of the synset, used to separate it from the other meanings of the word "dog".

The noun dog has 7 senses (first 1 from tagged texts)

1. (42) **dog**, domestic dog, Canis familiaris -- (a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds; "the dog barked all night")
2. frump, **dog** -- (a dull unattractive unpleasant girl or woman; "she got a reputation as a frump"; "she's a real dog")
3. **dog** -- (informal term for a man; "you lucky dog")
4. cad, bounder, blackguard, **dog**, hound, heel -- (someone who is morally reprehensible; "you dirty dog")
5. frank, frankfurter, hotdog, hot dog, **dog**, wiener, wienerwurst, weenie -- (a smooth-textured sausage of minced beef or pork usually smoked; often served on a bread roll)
6. pawl, detent, click, **dog** -- (a hinged catch that fits into a notch of a ratchet to move a wheel forward or prevent it from moving backward)
7. andiron, firedog, **dog**, dog-iron -- (metal supports for logs in a fireplace; "the andirons were too hot to touch")

The verb dog has 1 sense (first 1 from tagged texts)

1. (2) chase, chase after, trail, tail, tag, give chase, **dog**, go after, track -- (go after with the intent to catch; "The policeman chased the mugger down the alley"; "the dog chased the rabbit")

FIGURE 2.1: The possible meanings of the word "dog"

To organize words into synsets in this way one first needs to have a clear idea about what one means by synonymy. A strict definition of synonymy would claim that for form F and F' to be synonymous one must be able to replace one with the other in any sentence without changing the truth-value of that sentence. This is similar to using Leibniz law of identity to qualify synonymy. WordNet uses a weaker definition where one says that whether two forms are synonyms is dependent on the semantic context that they are in, and that two words can be synonyms with respect to this semantic context[Miller et al., 1990]. This weaker definition still necessitates that words must be from the same syntactic category to be synonymous, and explains together with Fillenbaum and Jones [1965] why it is useful to separate the syntactic categories.

2.2.1 Hyponymy and hypernymy

Hyponymy and hypernymy is the main way that nouns are organized in WordNet. Hypo- and hypernymy are a different type of relation between words than synonymy. While synonymy is a relation between different forms of a word, hypo- and hypernymy are relations between word meanings. The two are the inverse relation of each other and I will explain them by defining what makes something a hypernym. The concept M is the hypernym of some other concept M' if M is such that native speakers of English would agree with claims of the type "M' is a type of M". As an example: {animal} would be a hypernym of {dog}, since a native user of English would agree that "a dog is a type of animal". Hypernymy is transitive, meaning that if M is a hypernym of M', and M' is a hypernym of M'', then M is a hypernym of M''. So since we know that dog is a type of animal, and that Labrador is a type of dog, we also know that {animal} is a hypernym of {Labrador}. In addition to being transitive, hypernymy is also asymmetric. This means that the fact that M is a hypernym of M' entails that M' cannot be a hypernym of M. When we know that {animal} is the hypernym of {Labrador}, we also know that {Labrador} is not a hypernym of {animal}[Miller et al., 1990].

Hyponymy is the inverse relation of hypernymy. This means that if M is a hypernym of M', then M' is a hyponym of M. The fact that {animal} is a hypernym of {Labrador} means that {Labrador} is a hyponym of {animal}. Hyponymy is also transitive and asymmetric[Miller et al., 1990].

Another aspect of this relation with respect to nouns is the fact that one asserts some type of inherence from the more general hypernyms to its more specialized hyponyms. This means that if one knows that a concept has some properties, then the hyponyms of that concept will have all the same properties, in addition to the properties it has which distinguishes it from the concept[Miller, 1990].

This has some basis in psycholexicologic research. Collins and Quillian [1969] showed that subjects used less time to say that a concept had some property when that property was viewed as more prototypical for concept, than if it was seen a property of some more general concept. Figure 2.2 shows an example of such a hypothetical memory structure. The tests performed showed that subjects would spend less time on deciding that a shark was dangerous than that it had fins or that it breaths. This was in line with the hypothesis as being dangerous is more prototypical of a shark, while having fins or breathing is something which is associated with more general concepts.

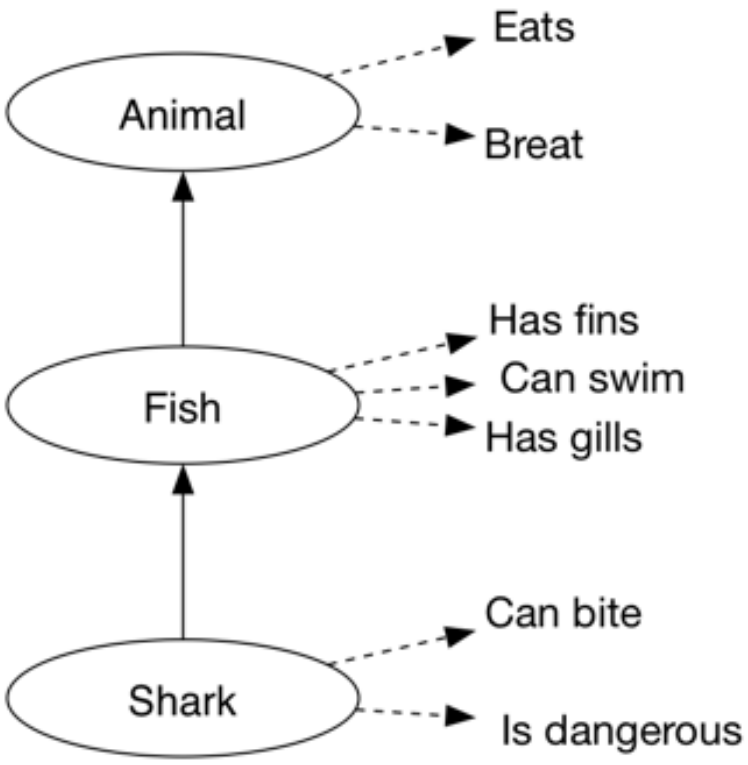

FIGURE 2.2: A hypothetical memory structure, adapted from Collins and Quillian [1969]

Nouns in WordNet are organized in a hierarchal fashion. One organizational issue then is if one should organize the nouns as one, or several hierarchies. WordNet started by organizing the nouns into 25 semantic prime categories. When these had been created one

realized that these contained natural groupings. At this point one decided to add some top-level synsets which tied these together, with the most general being {entity}[Miller, 1990].

2.3 DBPedia

DBPedia is an ontology based on data from Wikipedia. Wikipedia is a successful open collaborative information project, with about 4.2 million articles in English and 30 million all together [1]. The justification for the ontology is similar to the criticism mentioned by Shirky [2007] that it is difficult to know how a schema or ontology is going to be used before it is released on the Web. Instead of starting with an ontology then the work with DBPedia examine how one can extract unstructured data from Wikipedia and transform it into a machine readable ontology. The main contributions of the project has been[Auer et al., 2007]:

- Developing a framework for extracting information from Wikipedia to RDF
- Linking the items in the dataset to other ontologies
- Exposing the RDF dataset to the public
- Developing APIs for accessing the dataset

The extraction process is described in Auer and Lehmann [2007]. DBPedia extracts information from templates on the Wikipedia pages. The algorithm used looks through templates that have a high probability of containing structured information. These templates are parsed looking for pairs of attributes and values. The attributes correspond to predicates, and the values to objects of RDF statements. The subject of the statements would be the topic of the page itself, and it is given a unique URI in the DBPedia namespace by using its Wikipedia URL.

There is also some post processing of the results to enhance the data further. If the object of the statement is another Wikipedia resource, the same process that exists for the subject is used to find the targets URL and link it to that. In the cases where the object is a literal of a known type, then the data type information is added.

[1] As of 2013-04-16: `http://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia`

DBPedia is not a static ontology and in continuously being updated with data from Wikipedia, making sure it is up to date with the newest terms and information.

2.4 Lexitags

Veres [2011] marks one of the differences between the Web 2.0, or the Social Web and the Semantic Web as the fact that the Social Web is based on giving users the ability to create free form, unstructured data or content. The Semantic Web on the other hand is based on having a structure that allows for automatic machine processing of the content. The advantage of the Social Web is that it provides a very low threshold for content providers, but the lack of structure can provide long term difficulties since it can become hard to retrieve older content. The Semantic Web on the other hand has a higher threshold, but the structure makes retrieving the correct content easier, since machines can reason about the content and be made responsible for finding the correct content. The goal then would be to create a system that feels like a social system at input time, but which creates structured markup that can be searched like a semantic system at retrieval time.

Analysis of tags by Tonkin and Guy [2006] from delicious[2] and Flickr[3], two Web sites that are focused on user generated content and tags, showed that a large amount of the tags were "sloppy". Sloppy tags could mean that the tags were misspelled, that users had made compounded words my merging several words into one tag or that they added non-alphanumeric characters.

The lexitags system is developed to work with tags and provides a solution to the problem of sloppy tags. Lexitags solves the disambiguation problem by providing the user with a list of suggested senses for the tag provided. These senses can be in the form of a synset or the description of the sense. The user can then pick which sense that corresponds with the meaning the user wanted to convey[Veres, 2011]. The sense the user chooses will then be stored as a reference to one of the vocabularies that lexitags uses. The vocabularies used by lexitags are WordNet and DBPedia. Since WordNet mainly contains dictionary words, DBPedia was chosen to complement it to cover things like names, places, or compound terms.

[2] `https://delicious.com`
[3] `http://www.flickr.com`

The lexitags server is used in MaDaME to find mappings from the natural language concepts to WordNet. The artefact will treat the keywords that the user selects on the page as tags, and use the lexitags interface to find disambiguated synsets describing the concepts they represent.

2.5 Suggested upper merged ontology

The suggested upper merged ontology (SUMO) was created based on work from the Standard Upper Ontology (SUO), and is one of the proposed candidates for SUO. The work with SUO was started to fulfill the need for a large, free, public standard ontology created by cross discipline consortium of contributors.

The idea behind a standard upper ontology is to simplify generating new knowledge and databases by providing high level concepts that one can use to build upon, and which can help with interoperability with other systems. It could also ease the burden of working with legacy databases and knowledge stores, as one can just map them to this common ontology, and use the language of the ontology to communicate. It can also help combine different domain specific ontologies. By having a common set of terms, even if just a subset, it becomes easier to integrate the different ontologies[Niles and Pease, 2001].

The process involved in creating SUMO consisted of several steps. To start with it was necessary to identify high lever ontologies which where not hampered by licensing issues. As the name can imply, the basis of SUMO was to merge these ontologies together into one unified ontology, using the upper concepts of these ontologies. The upper levels are shown in figure 2.3 (page 16). Physical describes all those things that have some position in time and space, things like atoms, dogs and galaxies. The other main category, Abstract, covers those things that do not have some physical extension, like numbers, justice, and song. In addition to these types there is a unifying top-level type called entity, which encompasses all things physical and abstract.

Before the merge could start, the different ontologies were translated into the same knowledge representation format, to make sure they conformed to the same syntax[Niles and Pease, 2001]. SUMO is written in the SUO knowledge interchange format, but is also translated to the Web Ontology Language (OWL)[Benzmüller and Pease, 2012]. Then the ontologies that contained the highest-level concepts were merged by unifying those

concepts that were equal, and positioning the other concepts in relation to them. This upmost level was then used as a basis for aligning the other lower level ontologies[Niles and Pease, 2001].

Merging the lower level terms fell into four general cases. In the simplest case the term did not correspond to terms in the other ontologies, but was useful and suitable, and did not break with any of the philosophical decisions made.

Other terms were found to be unsuitable for SUMO. Since SUMO was supposed to be a top-level ontology terms that were to narrow, or which had little practical use were discarded.

In other cases a term would refer to the same meaning as some other term in the ontology. In these cases where it was only the term chosen to convey the meaning that differed one can map both term to the same term in the final ontology. In the last case the terms has overlapping meaning. In these cases one either kept both the terms, or picked which one to use in the merged ontology.

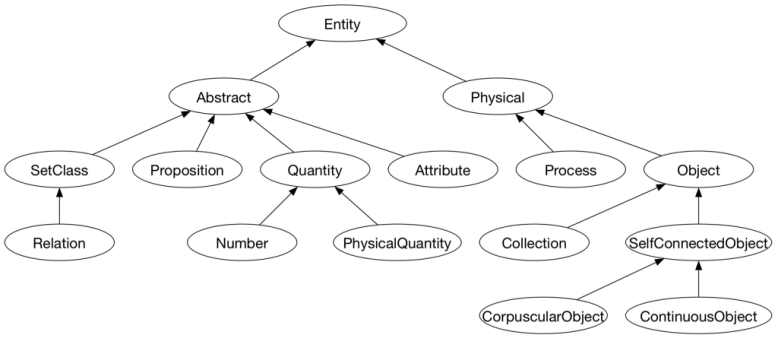

FIGURE 2.3: The upper levels of SUMO, adapted from Niles and Pease [2001]

2.6 Schema.org

Schema.org[4] is a joint venture by Google, Yahoo, and Bing, with the expressed goal of creating richer search results by getting content creators add metadata to their pages. In addition to giving the search engines extra metadata about the content of the page, the search engines also use them to provide information for rich snippets[Guha, 2011]. Rich snippets are small pieces of HTML which in some way can add information to a search

[4] `http://schema.org`

result, showing the number of stars a product got in a review, a picture of the author, part of the multimedia content or some other enriched content[Mayer and Menzel, 2009]. Figure 2.4 shows how the search result for the MaDaME homepage has been made more prominent by adding an image of the author of the page. Schema.org is tied closely to

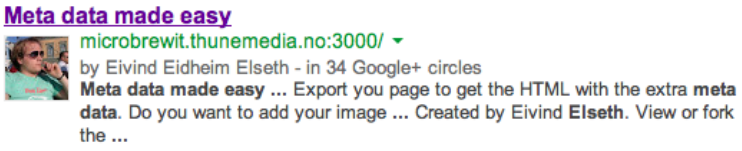

FIGURE 2.4: The rich snippet displayed for the MaDaME homepage

the microdata format[5], but does at the moment still allow RDFa.

Schema.org is not a general or upper level, or a domain specific ontology, but can be regarded as a middle level ontology. That is to say that it does not try to be neither an all-encompassing ontology, nor an ontology which covers everything within a domain. The role as a midlevel ontology involves covering the most common use cases without having the entire hierarchy modeled. The top levels of schema.org are shown in figure 2.5.

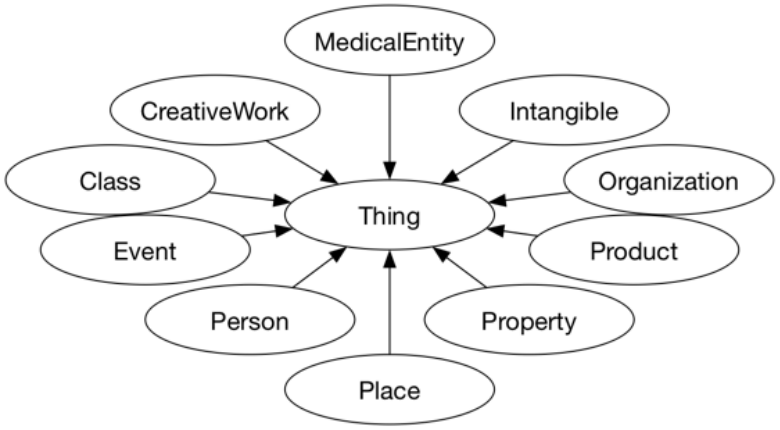

FIGURE 2.5: The top-level entities of schema.org

Inspecting the ontology shows that it is clearly geared toward search engine and commercial usage, both in the concepts that are selected, and in the hierarchal structure of the ontology[Ronallo, 2012]. It is also a fairly small ontology with only 577 types, excluding the 10 data types.

[5] `http://dev.w3.org/html5/md-LC/`

If schema.org does not contain the term needed they suggest extending the scheme by adding sub-types or sub-properties. The mechanism they provide for extending the vocabulary is string concatenation. Schema.org defines person using the following URL:

```
http://schema.org/Person
```

If you need to say that this person is a minister you would then extend Person by adding Minister at the end:

```
http://schema.org/Person/Minister
```

The intention is to have the extension mechanism working in a way that will give search engines some information, even when they come across unknown terms. It is also intended as a way to let the ontology grow organically, as terms which become common can be included in the schema.

This extension mechanism does however not conform to the microdata specification as Tennison [2011] explains. The specification says that URIs should be opaque identifiers, and that one should not infer meaning through string parsing. The mechanism also comes into difficulties when it comes to disambiguation. It is impossible to infer from the URI itself if Person/Minister refers to a clergyman, or to a minister of state. This problem would also persist when incorporating the term in the schema, and might make markup that used to be correct say something that was unintended.

Chapter 3

Methodology

The design research methodology is utilized to perform the research in this thesis. The guidelines provided in Hevner et al. [2004] will be followed in the work related to the thesis to ensure that the process is rigorous. The guidelines provided in this article are:

1. Design as an Artifact
2. Problem Relevance
3. Design Evaluation
4. Research Contributions
5. Research Rigor
6. Design as a Search Process
7. Communication of Research

The first guideline says that a design research project should produce some artefact. This thesis focuses on the development and testing of MaDaME. The development of the artefact will be described in chapter 4. It is also the testing of this artefact which is the topic of chapter 5, and the conclusion will describe if the results of the tests allow us to answer the research question positively.

In the introduction the motivation for the thesis, and why it is a relevant contribution to design science was described. The case was made for the need for computer readable

metadata for information retrieval, and the difficultly inherent in the creation of this data for new users. The work performed in completing this thesis can help lower the barrier of entry for generating semantically enriched Web sites. If the research question is successfully answered, the results can hopefully be used to both increase the value of computer search in general, and increase the visibility and discoverability of the Web pages that use tools that build on the work done.

Design evaluation is also mentioned and this guideline stresses the need for rigorous evaluation of the artefact that has been developed. To evaluate the success of the project I will look at the artefacts ability to map natural language to ontologies. The goal is to generate metadata that is of the level of quality as existing metadata of a comparable type. Whether the tool can add the metadata to the Web page using RDFa, without changing the way the Web page is rendered by browsers will also examined.

The main research contribution of this project will be MaDaME, which will contribute to solve the problem of how to get users to create semantic content. This prototype system can serve, either as a starting point for development, or as an inspiration as to how one can create a system that makes it easier to create semantic content on the Web.

The need for research rigor, which is mentioned in guideline 5 will be followed by following the multimethodological approach suggested in Chen et al. [1990] and Nunamaker Jr and Chen [1990]. Chen et al. [1990] proposes four activities for the design process that interact in the development of information systems (see figure 3.1 on page 21). The paper suggests using a multimethodological approach where one moves between different research activities: Theory building, experimentation, observation and systems development. Using these different approaches should help with catching important facets that might otherwise have been missed or over looked.

Guideline 6 says that design research should be a search process. In this context that means that one should explore the possible implementations of the artefact by iterating through phases of generating prototypes and testing these prototypes against the requirements of the project (as seen in figure 3.2, page 22). To attain this type of cycle I choose to use a system development methodology that utilizes multiple iterations of building and testing.

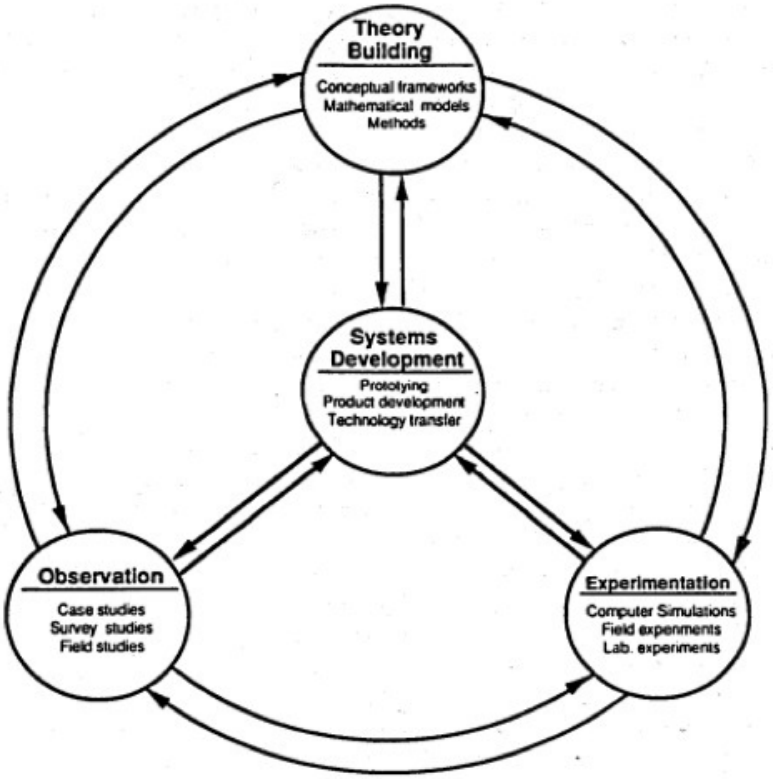

FIGURE 3.1: A multimethodological approach to IS research, from Chen et al. [1990]

The last guideline proposed has to do with clear communication of the results of the research. It is further proposed that one should take care to have several channels of communication with different levels of technical detail. The reasoning is that one needs to convince both the technical and the managerial communities.

To conform to the seventh guideline one should communicate the results of ones research in such a way that it is accessible for the intended target audience. The primary way of communicating the research will be this thesis. The thesis will target an academic audience and describe the process of development, and the evaluation of the artefact.

In addition the work done on the project is made available to the developer community, with hopes that others can build on the work done in relation to this thesis. All the source code from the project is open source, developed and shared on GitHub[1], and released under the MIT license. Releasing the source code of the project means that others can learn from the system in another way than what they could just by reading this theses, as most of the details of implementation are outside the scope of an academic thesis. It also means that all those who wish to improve upon the work, either by resolving issues

[1]The project can be found at: `https://github.com/EivindEE/Madame`

with the codebase or by adding new features, can do so in a way that can enrich the tool for all those who wish to use it.

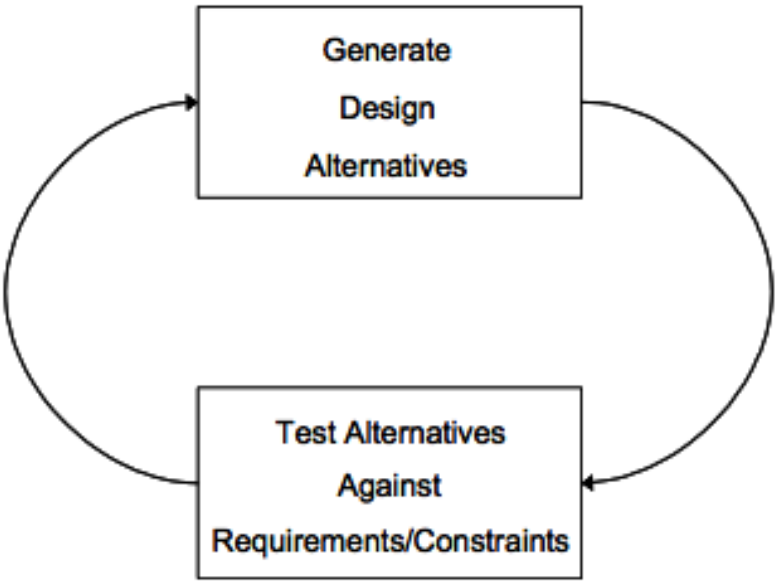

FIGURE 3.2: The generate/test cycle, from Hevner et al. [2004]

Chapter 4

Development

This chapter will describe the process of development, and which parts of MaDaME which were developed during the different stages of development. It will then go on to give a high level overview of the system. After giving an outline of the development process and the system, the main parts of the system will be described in greater detail.

4.1 The different phases of development

This section will outline the main sequence in which work on MaDaME was done. The overview of the iterations will explain when the initial development of the different parts were started to give an short explanation of the development process.

4.1.1 Iteration 1

The first iteration started with a technical peak exploring different tools and frameworks for development with JavaScript. I wanted to find tools that could reduce development time by reducing the amount of repetitive work done in the project.

It was decided that the Web page should use Twitter bootstrap[1] as a grid system. Using a standard grid system makes it easier to create a Web page that looks OK, and reduces the amount of time needed to set up a scaffold. Since the visual aspect of the Web page

[1] `http://twitter.github.io/bootstrap/`

was not an important aspect of the work a simple grid system like bootstrap eased the development burden.

It was decided to use SASS[2] to write the style sheets. It has a syntax that is similar to that of CSS. In addition to the regular syntax of CSS it also includes the possibility of using variables. This makes it possible to make global changes while only modifying one line in the style sheet. SASS compiles to standard CSS.

JavaScript lacks a good IDE so it would be helpful to have some tool that would check the code to make sure that the code was consistent and that the syntax was correct. I decided to use JSLint[3] to validate the code. Using a linter eases development as it enforces a particular coding standard for the project. The way the project was set up, the project would not build without it passing the linting test.

Since the project depended on compiling the style sheets and linting the code it was decided to use grunt.js[4] to automate this process. Grunt.js is a task manager that can run certain tasks when specified events are triggered. The project was set up so that each time a JavaScript or SASS file was saved the task manager compiled the SASS document, and concatenated the public JavaScript files. Concatenating the JavaScript files meant that it was unnecessary to update the HTML when a new JavaScript file was created or deleted, since the Web page only needed to include that one file.

Node.js was chosen as the development platform of the project. Setting up a basic server in node.js can be done in a single line of code. Using this basic server requires a lot of low level handling of requests and responses, including parsing request URL to find the correct handler. To abstract these low level concepts away MaDaME uses express[5], a Web application framework which simplifies routing requests to the correct handler and handling static files. It was also decided to use the jade[6] templating language to generate the HTML that was used on the Web site. Using a templating language made the distinction between structure and content clearer, and enabled reuse of structure and content between different versions of the Web site during development.

[2] `http://sass-lang.com`
[3] `http://www.JSLint.com`
[4] `http://gruntjs.com`
[5] `http://expressjs.com`
[6] `http://jade-lang.com`

The technological peak also included getting to know the DOM, the API used for manipulating HTML documents. The problem that was tried to solve during this technical peak was how to find minimal legal ranges. MaDaME has to create separate tags to add the metadata it created, and that the system was going to let users select which portions of the text that carried the semantic content. Since the user selection might not be a legal HTML range I worked on an algorithm to find the smallest range that was both a legal range, and which contained the whole of the selected text.

4.1.2 Iteration 2

The next step consisted of actually generating metadata tags with the correct RDFa syntax that also enclosed the user selection. One of the tasks was giving each selected section a unique id so that they could be referenced externally. It was decided that this id should carry some information about the content of the tag where possible, to increase the readability for the users. The mapping files had not been created at this point so the tags contained mock data.

In this iteration work also began with generating the hypernym chains that were needed to find mappings. The initial approach was to use an existing triple store and query the information with SPARQL. The triple store used an early version of SPARQL that did not allow recursive queries. Since the system needed to find the transitive closure of the hypernym relation, using this source would require multiple and sequential queries and would take to long to be tenable. JavaScript does not have a mature library for querying the WordNet database, so it was not possible to not create JavaScript module to perform the task. The solution used was instead to use Perl to write a script that could query a local WordNet instance. This again required a technical peak to learn the basics of the Perl programming language but gave a satisfying solution.

This iteration was also used to find WordNet mapping files and convert them to a format more suitable for the system. I had a mapping file between WordNet and SUMO in the WordNet database format, and one between WordNet and schema.org in RDF. Both of these files were translated into JavaScript objects.

4.1.3 Iteration 3

In the third iteration the actual best-fit algorithms that my supervisor and I had proposed were implement. The algorithms will be discussed further in section 4.7. The two strategies used by these algorithms were to either look at all the hypernyms of the synsets before looking at other senses, or to look alternately at the direct hypernym and then the siblings of the synset.

This was also the iteration where development started on a mechanism for fetching HTML from other Web sites so that they could be marked up using MaDaME.

4.1.4 Iteration 4

The export module was developed in the fourth iteration. This module would take the Web page that the user had imported into the system and create a new HTML document. This document would be stored in the database, and would be accessible for the user through a URL which was provided when the page was exported.

Adding properties to entities was also developed during this stage. This was only implemented for schema.org properties, both to keep the number of properties at a manageable level, and because extracting all the allowed properties for all SUMO classes would be a difficult task.

In this iteration the ability to add author information was also added. This is not closely tied to the problem of translating natural language to formal ontologies, but it is tied to allowing users to add metadata in a simple way without knowing about the formal underpinnings.

4.1.5 Iteration 5

The last iteration consisted of testing and refactoring the system. I wrote some test scripts that would run the best-fit algorithms that had beed created over a set of synsets, and print out the resulting mappings along with information about whether the algorithms gave corresponding results, and the measure of quality of the mapping. The results of these tests will be discussed further in section 5.1.

4.1.6 Overview of the completed system

The artefact was divided into several modules which function independently of each other. The modules were loosely coupled so that each module could be changed or modified without affecting the other modules. The modules that were created were:

- The Web front end
- A module that fetched and processed the HTML from the sites that the user wanted to mark up
- A module for fetching possible disambiguating terms from lexitags
- A module for finding the best-fit mappings for SUMO and schema.org
- A module for finding the best-fit mappings from DBPedia to schema.org
- A module that puts the document back in its initial state and saves it to a persistent storage

4.2 Interaction method

When first faced with the problem of how to let users interact with the Web application I thought that using the highlighting or selection of text to be the most intuitive approach. This also appeared to be the simplest interaction method to implement. The simple case of taking selected text and displaying it in the console can be implemented as in listing 4.1.

```
1  var logSelection = function () {
2    console.log(window.getSelection().getRangeAt(0).toString());
3  };
4  document.addEventListener('mouseup', logSelection);
```

LISTING 4.1: Logging selected text

The "mouseup" event is triggered every time the user has pressed and then released the mouse pointer on the Web site. This event is not available on mobile and tablet type browsers, but adding listeners for corresponding events should not be difficult if there should be a need for this at a later stage.

In the Web application the selected text is sent to the Web server to find possible disambiguations for the text. The lexitags service that the system uses for disambiguation now is targeted towards disambiguating single words. It does handle some composite terms and some proper nouns, but this is outside the normal use case. The lexitags service is designed to disambiguate single tags, not large paragraphs of text. To accommodate this MaDaME does a simple check for the length of the text string. If the length of the string is more than the longest allowed length, the query is instead replaced with a shorter query indicating that the selection is of a larger section of text.

The response from the server is a JSON string containing the terms that were found to be possible senses of the selected text. Each of these senses have an explanation property which gives a short textual description of the meaning of the sense. These senses are displayed as a list on the Web site. The senses that are returned come from different sources. They can come either from WordNet and take the form of WordNet synsets, or they can be resources from DBPedia. Both of these can be used to generate mappings, but the most interesting mapping for this thesis lies in generating mappings from WordNet so the synsets are given preference in the list. DBPedia does not have a standardized way of finding the super type of a resource, so if the resource does not have a schema.org mapping it is difficult to find generalizations that do. Lexitags also returns the top-level resources from schema.org as shown in figure 2.5 (page 17). These are sorted to the bottom of the list as these are fallbacks for when none of the of the terms suggested for the word turn out to be reasonable suggestions for the selected text.

A special case for images was also developed, as these are object on a Web page that were assumed users would be interested in marking up, but do not contain any text that can be disambiguated. This special case kicks in when the selection highlighted contains an image tag, but no text.

4.2.1 Adding types

To select one of the terms from the list as the meaning of the selected text the user clicks the sense in the list see in figure 4.1. When a sense is clicked it is sent to the server to be mapped. The servers uses the best-fit algorithm described in 4.7 (page 40) to find the ontology references that fits the sense best and attaches these references to the selection using RDFa. The algorithm used to attach the metadata to the selection can handle

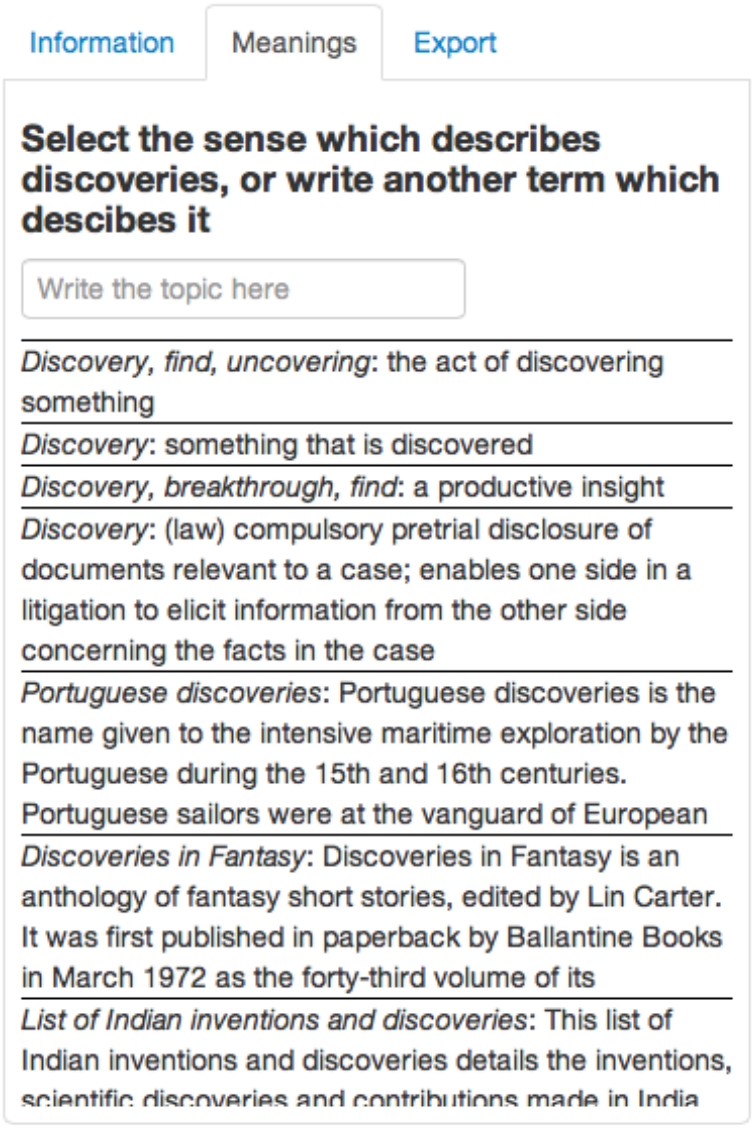

FIGURE 4.1: The list of possible interpretations of "discoveries"

an arbitrary amount of namespaces and references. If there is duplication of references these will be dropped.

When the metadata element is created, the schema.org type is sent back to the server to get a list of the possible properties of that type. The server uses a JSON representation of the schema.org ontology to find the properties that a type can have. The representation that is used on the server was retrieved from `http://schema.rdfs.org`, which is a support site that tries to promote the use of linked data. For each of the types in schema.org the JSON file provides the properties specific to that type, and all the properties it has inherited from its super types. The information about which properties

a type can have is combined with a short description of what the property represents, and information about the range of the property, that is the range of values that are valid values of the property. The resulting JSON object is then returned to the Web site. These properties are then added to the metadata element as separate elements, with the description and range stored as data attributes of the element.

4.2.2 Adding properties

Users can click text that has been tagged with metadata. This triggers a popover view that displays the properties that the element can have, and which types of values are allowed for each property as shown in figure 4.2. If the property allows other schema.org

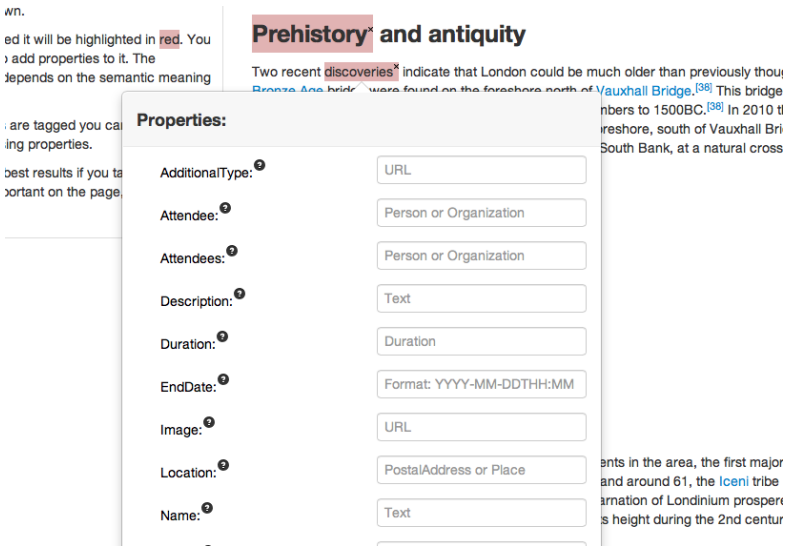

FIGURE 4.2: The properties popover for schema:Event

types as their value the Web application does a scan of the content of the Web site to check if there are elements of the correct type on the page. Elements of the correct type that are found are put in a combo box and can be selected as values for the property. In all cases the user is presented with the option of writing the value of the property in an input field. If the property already has a value it will be displayed in the input field when the popover is displayed.

4.3 Maintaining the well-formedness of the document

One of the difficult aspects of adding the metadata to the selections was making sure that the HTML was well-formed after the metadata was inserted.

Special care needs to be taken with the ranges of the selection. Since the system need to insert the metadata into tags it cannot just use the user range without some checks. If we consider the simple case:

```
<p>some text in a p</p> <p>‘ plus some other text </p>
```

If the Web application simply inserted a tag it would produce invalid HTML since it would create overlapping elements.

```
<p>‘some text <tag> in a p</p> <p> plus some </tag> other text </p>
```

One way to get a well-formed HTML document would be to close and open the HTML tags.

```
<p>some text </p><tag><p> in a p</p> <p> plus some </p></tag><p> other text
</p>
```

Solving the problem in this manner would change the structure and rendering of the Web page, as it would create new block elements. This is not a tenable solution, as one of the goals of this thesis is to not change the way the document is displayed. The solution that was found was to instead expand the range of the selection so that it covered a larger part of the document:

```
<tag><p>some text in a p</p> <p> plus some other text </p></tag>
```

The interaction method consists of letting the user make arbitrary selections and then adding metadata to the selection. This could easily lead to overlapping elements in the HTML as shown above. To avoid this malformed HTML the system checks the range of the selected text and see if surrounding it with a tag containing metadata would result in a well-formed document, and modify the range if needed. If the start and end of the selection are in the same element then adding metadata is safe. If that is not the case then the Web application will find the smallest change that can be made to the scope of the range to make it safe. There are two general cases her, when either the start or the

end of the range is in a descendant element of the other, and when both start and end are descendants of a common element but not of each other.

In the first case the system needs to find out which element is the descendant of which. This is done by following the ancestor links of each element. When the artefact finds that either the start or end element has the other as an ancestor it uses the ancestor which is the direct child of the other element and place the start or end tag right before or after that tag, depending on which element descended from which.

When the elements are not descendants of each other then the only option is that they must be descendants of some other element. If nothing else they must both be descendants of the root element. The approach for this is similar to what is done for elements that are descendants of each other, but instead of finding the ancestor which is the direct child of the other node the Web application finds the one which is a child of the closest common node. These elements are then surround with the metadata tag. This metadata element is styled with a distinct color to let the user know that the text is tagged.

4.4 Importing Web pages

One of the issues that came up early was how to import and display the Web sites that the user wanted to add metadata to. Early attempts tried using the iframe[7] element of HTML. Embedding the target Web site in an iframe was the first try as it would allow the page to display in the same manner as it does when accessed directly. Using the iframe element in this manner would however constitute cross-domain communication and is not allowed. The solution that is used in the system now uses a module on the server to fetch the HTML of the document the user wants to enrich. The HTML that is imported is now appended to a content element in the Web application. The JavaScript events used in the Web application are attached to this element, to keep the interaction confined to the content the user is meant to tag. The id of this tag is also used to tell if a certain element is a part of the user content section by checking if it is the child of the element.

The user writes the URL of the Web page into an input field on the top of the Web application as shown in figure 4.3. The server will fetch the document stored at that

[7] http://www.w3.org/TR/html5/embedded-content-0.html#the-iframe-element

FIGURE 4.3: The header of the MaDaME Web page

location. I created a proxy to get the html from other Web sites. The proxy would take a URL and get the resource located there. If the resource is not of the right type i.e. not a HTML document with a body element, the proxy would return an error. When the resource has been downloaded the content of the body element of the document is parsed to comment out code that could be harmful, that could make the page display incorrectly, or that could disrupt the way the Web application functions. The parts of the page that are comment out are the script, iframe and comment elements. The script elements could disrupt the way the Web application works, either by overwriting the behavior of functions, or otherwise alter the behavior of the Web application. In addition the script elements are not visible on the page so interacting with them would be hard. Removing them was also convenient during development. The scripts in the tags would often rely on variables or functions that should have been loaded elsewhere, and they would clutter the console making it more difficult to find the relevant information. Iframes were excluded since they could make the Web application appear to be inconsistent. The iframes look like a part of the page, but it would not be possible to alter their content since the HTML belongs to a different document. The difference between the document and the iframe would be invisible to the user, so it was decided that it would be better to remove the element all together. The comment elements were removed to make sure that the page would display the way it should. Since the system uses comments to remove iframes and scripts, and since these elements could them selves contain comments, the content of the comments would sometimes be displayed as text elements on the page.

MaDaME is only designed to add metadata to the body element of the document, so the page is split into body and head parts before it is returned to the Web application. The document is returned as a JSON string with one property containing the processed body, and one containing the content of the head.

4.5 Generating mapping files

An important part of the early work was to generate files that the tool could use to find mappings from WordNet synsets into the ontologies which were used. The ontologies that were chosen for the project, SUMO and schema.org, already had these kinds of mapping files. The mapping files were however in different formats and in formats that were unsuitable for use in a JavaScript based application. The format that was chosen was one that provided the option to ask if there was a mapping for a given WordNet synset, in other words what was chosen was a dictionary. JavaScript objects are themselves dictionaries, making it natural to choose to them as the representation. The structure of the objects is very simple, the synsets are used as the keys to entities in the target ontologies as seen in listing 4.2.

```
1 exports.mapping = {
2   "abstraction#n#6": "Intangible",
3   "accounting_firm#n#1": "AccountingService",
4   "address#n#6": "PostalAddress",
5   // The rest of the mappings
6   "work_unit#n#1": "Energy"
7 };
```

Listing 4.2: Excerpt from the WordNet to schema.org (wn2schema.js) mapping file

Since MaDaME uses JavaScript objects to represent the mappings, querying a mapping object to check if it contains a mapping for a certain synset would then just consist of checking if the object has a property with the given name as in listing 4.3.

```
1   if (mapping['synset-to-check']) {
2     // it contains a mapping
3   } else {
4     // it does not contain a mapping
5   }
```

Listing 4.3: Testing if a mapping exists

The mapping files were to large to translate to JavaScript objects by hand, with the SUMO mapping file alone covering more than 82.000 mappings, so I used regular expressions to modify the files globally. The schema.org mapping file[8] that was used as the

[8] https://github.com/mhausenblas/schema-org-rdf

basis for the WordNet to schema.org mapping file is written in RDF/XML. The first step in creating the mapping file was to remove the parts of the file that were irrelevant to the problem at hand. The original file contains triples describing schema.org properties, the schema.org entities and the hierarchal ordering of entities. This information is handled at other places in MaDaME and could be safely removed. Only the elements mapping synsets to schema.org terms were kept, and all other information except the names were removed. At this point the XML tags were stripped so only the synset reference and the schema type remained. Since the correct elements were now at the correct place it was only necessary to surround them with the correct quotation marks, separate them with a colon and add a comma at the end to separate it from the next property.

The SUMO[9] mapping followed a similar pattern, but the original format was different as it used the WordNet database format. The WordNetMappings30-noun file was used as the basis so for this mapping so removing non-noun phrases was unnecessary. The fact that each line in the file corresponded to a line in the mapping file made the process of translation easier, there was however one caveat in that the file used the offset to identify the synset, instead of the word#category#number format which is used for the other mapping file. I put some work into finding a way of substituting the offset for the synset id, but found that it was simpler to find the offset for the synsets that were queried. Unifying the way the mapping object properties are named is one of the things that can simplify the generalization of the process at a later stage.

4.6 Building the hypernym chain

Many synsets that the users will select as representing the concepts in the Web application will be more specific than the ones used in ontologies MaDaME maps to. This means that the tool needs some way of generalizing the synsets. This thesis examines two approaches to finding mappings from the synset to the ontologies. One approach is to use the hypernym chain, a term used in Veres [2011] to describe the list of all the hypernyms of a synset. The other approach examined is to use the sibling senses of the synset and the synsets in the hypernym chain.

[9] `http://sigmakee.cvs.sourceforge.net/sigmakee/KBs/WordNetMappings/`

JavaScript does not have a library for querying the WordNet database, so this part of the project needed to be coded in another language. Had there been a JavaScript library for querying WordNet, it would be natural to generate the hypernym chain while searching for the mappings from the synset to the ontologies. Having to use a different programming language and calling it outside the program, it is easier and more efficient to split the process. The system will first generate the entire hypernym chain and add the siblings of all of the hypernyms, and then examine the hypernym chain to find the best mappings. This section will describe the first part of this process, generating the hypernym chain.

The system uses a Perl script[10] to generate the hypernym chains. The script is called by using the node child_process.exec[11] function, which calls a command line utility and uses the output stream as the input to the callback function.

The structure of WordNet as a whole can be described as a directed graph, as shown in figure 4.4. The graph must also be acyclical since the hypernym relation is asymmetric as mentioned in section 2.2.1. Each synset can be described as a node in this graph,

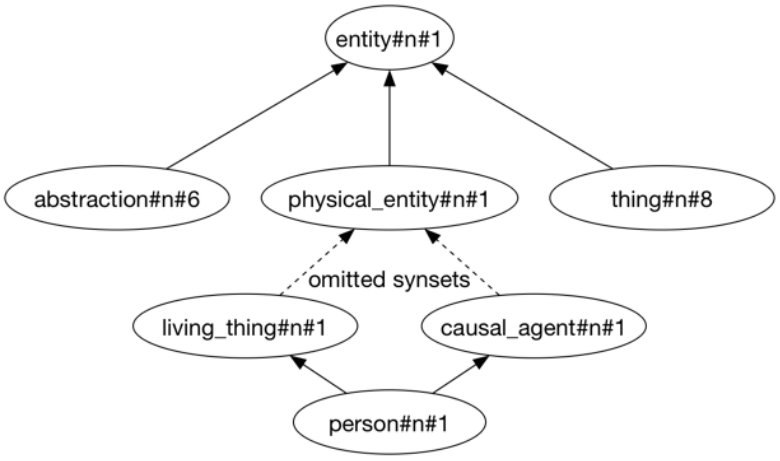

FIGURE 4.4: Part of the WordNet graph

with a directed edge connected to its direct hypernym. For each node in the graph it is also true that following one of the directed edges will lead to the synset entity, which is the most general concept in WordNet.

[10] `https://github.com/EivindEE/Madame/blob/master/scripts/parents.pl`

[11] `http://nodejs.org/api/child_process.html#child_process_child_process_exec_command_options_callback`

```
{
  "chain": [
    {
      "synset": "enrollee#n#1",
      "siblings": [ { "synset" : "self#n#2", "offset" : "09604981" ...},
      ],
      "offset": "10059162"
    },
    {
      "synset": "person#n#1",
      "siblings": [],
      "offset": "00007846"
    }
  ],
  "synset": "student#n#1",
  "siblings": [],
  "offset": "10665698"
}
```

Listing 4.4: Excerpt from the hypernym chain for student#n#1

The hypernym chain the script generates is a list of objects. The objects contain the hypernym itself, its offset in WordNet, and a list of all its siblings. Each sibling is stored as an object containing the synset, and its offset. An example of how the hypernym chain is represented as JSON can be seen in listing 4.4.

Generating the hypernym chain is done in two steps. The first step is to create a list of all the hypernyms, while the second is to add all the siblings. To generate the list of hypernyms, the script does a breadth first traversal of the graph as described in listing 4.5. For the graph in figure 4.4 the algorithm would add {living thing}, and {causal agent} before adding {physical entity} and {entity} to the list. If a synset is encountered multiple times, it is ignored in subsequent encounters. The hypernyms are added to the end of the list when they are found. This means that the synsets that are at the start of the list, are the ones that are at the closest level of abstraction to the synset for which the chain is built.

After the list of hypernyms has been created it is iterated over to add the siblings of each of the hypernyms. For each synset in the hypernym chain the script will find the synsets hypernym, and then find all the hyponyms of the hypernym. These sibling senses are then put in a list along with the synset.

```
my @hypernym = hypernym($synset);
my @hypernyms;
while (@hypernym){
  my $hypernym = shift @hypernym;
  push(@hypernym, hypernym($hypernym));
  push(@hypernyms, $hypernym);
}

my @hypernyms_with_siblings;
push(@hypernyms_with_siblings, {"synset" => $_ , "offset" => $wn->offset(
    $_) ,"siblings" => [siblings($_)]}) for @hypernyms;

my %result_map = ("chain" => [uniq(@hypernyms_with_siblings)], "synset"
    => $synset, "offset"=> $wn->offset($synset), "siblings" => [siblings(
    $synset)]);
my $json = JSON->new->utf8->pretty->encode(\%result_map);
print $json;

sub hypernym {
  my @hypernym = $wn->querySense($_[0], "hype");
  if ($#hypernym > 0) {
    return ();
  }
  return @hypernym;
}
sub siblings {
  my $synset = $_[0];
  my @hypernym = hypernym($synset); # Find hypernym of synset
  my @siblings;
  my @sibling_and_offset;
  push(@siblings, $wn->querySense($_, "hypo")) for @hypernym;
  for my $sibling (@siblings) {
    push (@sibling_and_offset, {"synset" => $sibling, "offset" => $wn->
    offset($sibling)});
  }
  return @sibling_and_offset;
}
```

Listing 4.5: Excerpt from the Perl script that generates the hypernym chain

4.6.1 Problems with multiple hypernyms

When starting work on the best-fit algorithms it was discovered that some hypernyms lead to incorrect mappings. The problem was noticed when mapping the synset quotation#noun#2, described as "a short note recognizing a source of information or of a quoted passage", which was mapped to the schema.org term MusicRecording. This error was caused by the fact that the hypernym chain of quotation#noun#2 contains section#n#1 which is a hyponym of both music#n#1 and writing#n#2, shown in figure 4.5.

The synset music#n#1 is described as "an artistic form of auditory communication incorporating instrumental or vocal tones in a structured and continuous manner". The

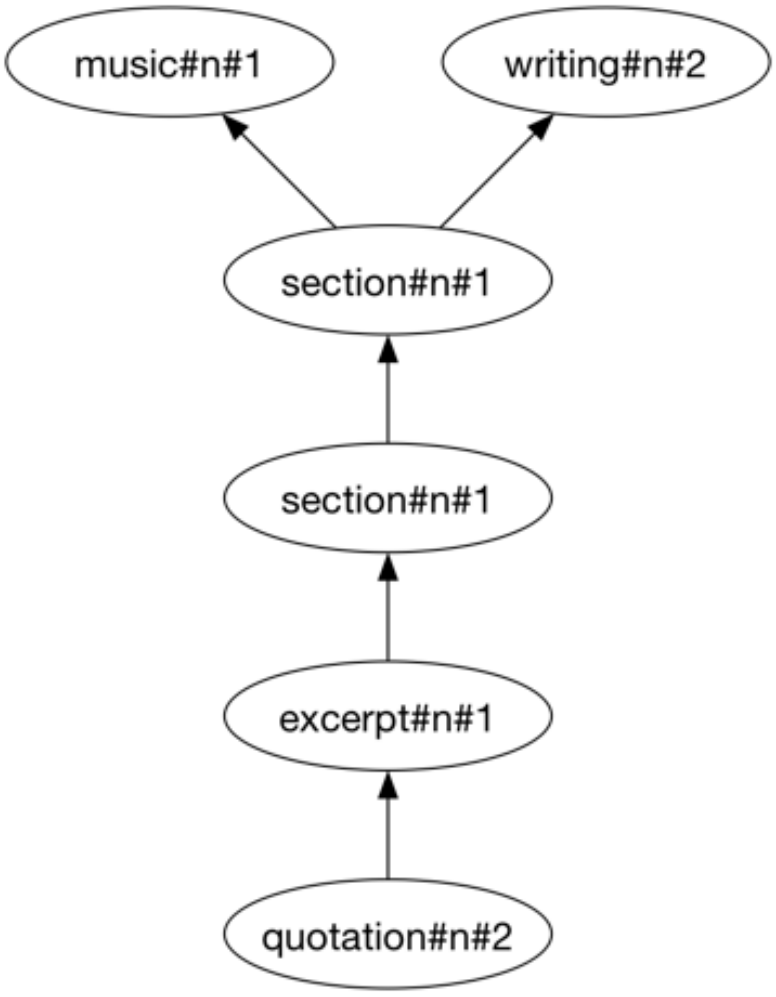

FIGURE 4.5: The start of the hypernym chain of quotation#noun#2

fact that quotation#noun#2 is a hyponym of music#n#1 violates the rules for hypernymy discussed in 2.2.1. Hypernyms are supposed to be generalizations of its hyponyms, and they should be in a "is a" relationship to each other. A native user of English would not say that a quotation is a type of music in the senses of the words are described above. This error has been reported to the maintainers of WordNet and will be fixed in the next public release (C. Fellbaum, personal communication, May 1).

The main problem this caused was that it made it unclear if both hypernyms would maintain the meaning of a synset, when the synset had more than one hypernym. It is not possible to choose a strategy where one only picks one of the hypernyms since the system has no way of knowing which of the hypernyms to choose.

The system now handles cases where a synset has multiple hypernyms by aborting the generation of the hypernym chain, and only using the hypernyms found up to that point.

This cutoff can be seen in the hypernym subroutine in the Perl script in listing 4.5. This is not an optimal solution as it discards a lot of synsets that would give good mappings. It was however chosen to avoid getting extra incorrect mappings when testing the system. The solution chosen was also simpler to implement than check against hypernyms that were found to be wrong, and a more sophisticated solution to choosing hypernyms was decided to be outside of the scope of the research problem.

4.7 Best-fit mapping

I will now describe the two algorithms that were developed to find mappings between the WordNet synsets and the ontologies. Both algorithms will start by looking at the synset itself. The system uses the direct mappings of the synset should they exist as there is no reason to believe that one can do better than a direct mapping. Given that there are ontologies that have not been mapped to the system uses the best-fit algorithms to find the missing mappings.

4.7.1 Hypernyms first

The hypernyms first algorithm traverses the hypernym chain by first looking at the direct hypernym of the synset for mappings. It will then follow the hypernym chain upward as long as no mapping has been found. This means that, except in cases where the generation of the hypernym chain is aborted because of multiple hypernyms such as mentioned in section 4.6, the algorithm will always map to the most general type of the corresponding ontologies instead of looking at the siblings. Looking at figure 4.6, the hypernyms first algorithm would follow the links upward to the top. If there are no mappings in the hypernym chain the algorithm will start to look at the siblings, and them looking at them in the same order as the hypernyms. It will start looking at the siblings of the synset and see if any of them have a mapping. As long as there are ontologies without mappings it will then follow the hypernym chain upward again, this time looking at the siblings of the hypernyms. If the mapping that is found belongs to a sibling then the algorithm map to the super-type of the type that it maps to.

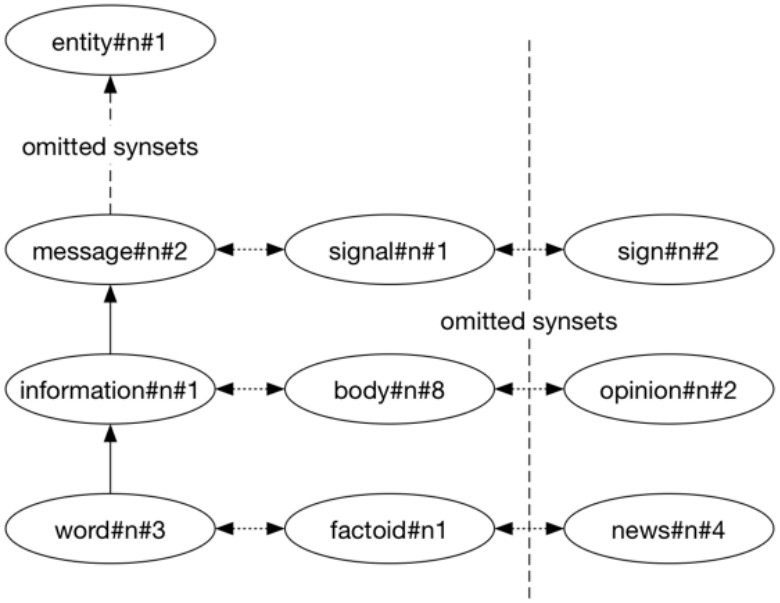

FIGURE 4.6: A hypernym chain with siblings

4.7.2 Hypernym then siblings

The hypernym first algorithm is conservative. Since hypernymy is a "is a" relationship there is little chance that the resulting mappings can be mistaken. It does however move quite quickly up the hypernym chain leading to mappings to types that are at higher level of abstraction than the synset. The other try at creating an algorithm tries to stay at a closer level of abstraction by looking at the siblings earlier in the traversal.

The algorithm also starts by looking at the hypernym. When a mapping is found from a sibling we use the super-type of the mapped type, not the type it self. Since WordNet has a very large vocabulary, it is reasonable to assume that a mapping from the hypernym would be closer or at least as close in meaning as a mapping to the super-type found through a sibling.

The hypernym then siblings algorithm diverges from the hypernyms first by looking at the siblings of the synset after looking at the hypernym. It will then follow the same procedure up the chain, first looking at each synsets hypernym, then looking at its siblings before moving up the chain and repeating the process. Looking again at figure 4.6 the algorithm will at each node first look at the node above, and then at the nodes to the right, before moving one node up the tree. The manner in which the algorithm chooses which sibling to map to is at the moment fairly naïve. The algorithm will use the

first match it finds, meaning that if two or more siblings have mappings to an ontology the first one found by the algorithm will be used. As shown in listing 4.5 the order of the siblings in the JSON is the same as the order that they are given by querying the WordNet database. For the system this is equal to using an unordered list since the order is unknown. No reason could be found which lead to the assumption that there was a reason to prefer one mapping over another, since there was not at this point any way for the artefact to know the meaning of the synset. Further refinement is likely possible of this assumption does not hold.

4.8 Exporting the document

To export a page after adding metadata the user clicks on the export tab in the sidebar and on the export button. This will provide the user with a link the Web site with the metadata added as shown in figure 4.7. The exported site is parsed to put the

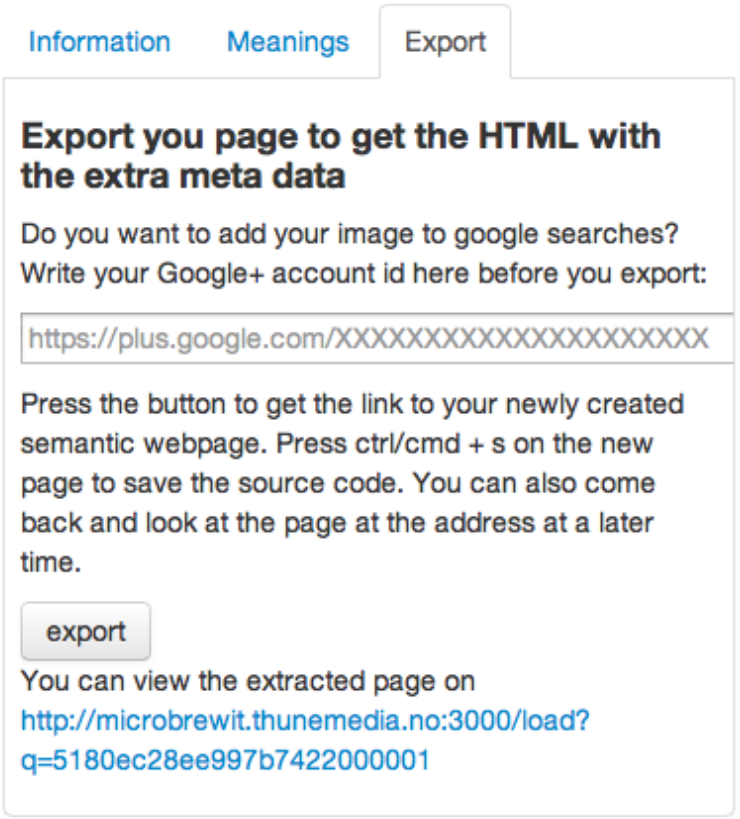

FIGURE 4.7: The export tab after exporting the current page

JavaScript and iframes back in so that the Web site look the same as it did before it

was imported into MaDaME. On the export pane the user is given the option to add a Google plus id to an authorship tag on the page. This is treated as a separate task as it pertains not to the content of the page, but to the page itself. It is also not a part of the schema.org ontology, but is included as it is a natural way to increase the visibility of the Web site in search results using metadata.

MaDaME uses MongoDB[12] as a database for storing the Web pages after the user has added metadata. MongoDB is a NoSQL database that uses JavaScript as its database interaction language. MongoDB uses a dynamic schema for storing data, meaning that each document in a collection does not need to have the same structure. This makes it suitable for projects with a rapid development style since the schema does not have to be finalized at early stages of development. The artefact uses mongoose[13] as a tool for modeling the objects. Mongoose allows one to abstract a lot of the database logic away and simplifies connecting with the database. When receiving a document to store the module removes all the extra comments that were added when the document was imported, and puts the document back in its original state. The database also stores the namespaces used in the document and the date on which the document was created. When the document has been stored successfully the URL of the newly created Web page is returned to the user as shown in figure 4.7 on page 42. If there was an error during the process the user is alerted about what went wrong.

[12] `http://www.mongodb.org`
[13] `https://github.com/learnboost/mongoose`

Chapter 5

Analysis and Discussion

This chapter will contain an analysis of the different parts of the system and discuss their appropriateness in solving the problems tied to the research question. The chapter will start with comparing the mapping algorithms to see if one of them performs better then the other. It will then try to answer the questions of whether MaDaME was able to generate metadata of equal or better quality than that which is already available, and if the system is capable of adding the metadata without changing the way the Web page is displayed. It will end with an overview of the usage data the system has received at this point and see how it corresponds with the expected use of the system.

5.1 Comparing the algorithms

To compare the two algorithms a list of English nouns was generated. This list was sent through the lexitags server to get synsets that corresponded to the meanings of each word. Both these lists were preprocessed to remove duplicates and to format them as JavaScript objects [1]. The final list of synsets contained 4350 unique synsets. A short script was written that was used to run the synsets through the best-fit algorithms, and to write a report of the results. For the schema.org version of the test the script wrote the average depth of the mapped type, as well as the total number of times the two algorithms had the same and different mappings. The depth was calculated as the distance from the root node in the tree, i.e. schema:Thing itself had a depth of 0, schema:Person which

[1] `https://github.com/EivindEE/Madame/tree/master/testing`

inherits directly from schema:Thing has a depth of 1 and so on. For the SUMO version the depth of each type was unavailable, so the test results from this test only show the agreement between the algorithms. The full results from the tests can be found at the URL `https://github.com/EivindEE/Master-thesis/tree/master/AlgComparison`.

As one can read from the numbers in table 5.1 the results from the SUMO test display no difference between the two algorithms when mapping from WordNet to SUMO. The two algorithms return identical mappings in 100% of the test cases. This indicates that there must have been a mapping either directly from the synset, or from the direct hypernym of the synset for each of the 4350 synsets in the test. This makes it hard to say anything relevant about the algorithms from these results.

	Schema.org	SUMO
Total number of synsets tested	4350	4350
Number of identical mappings	3262 (75%)	4350 (100%)
Number of different mappings	1088 (25%)	0 (0%)
No result found	598 (13.7%)	0 (0%)

Table 5.1: The testing results

The schema.org results are more interesting. The two algorithms still perform fairly equally. Reading from table 5.1 we can see that the algorithms are equal in 75% of the cases, but we can examine the 25% that are different and see which perform better in those cases. We can also see that the algorithms were unable to find a mapping in 13.7% of the cases. These cases cannot tell us much about which algorithm should be preferred, but could be a good starting point for finding parts of the WordNet to schema.org mapping which could be enhanced.

The prediction made beforehand was that the hypernyms first approach would have fewer incorrect mappings, but would give results at a more shallow depth. The last prediction was the easiest to test, as we know the schema.org hierarchy and can calculate the depth of each type. For each mapping the script running the test registered the depth of the type in the schema.org hierarchy. These depths were averaged over the total number of mappings made.

As seen in table 5.2 both algorithms map fairly high in the hierarchy. The hypernyms first approach maps to a type at level 0.69 on average when considering all the synsets, or to a type at level 0.72 when ignoring the cases where the two algorithms gave the same result. As predicted the hypernym then siblings approach does a little better, though not

	Hypernyms First	Hypernym then siblings
For all mappings		
Avg. depth total	0.688506	0.804138
For different mappings		
Avg. depth different	0.721507	1.183823
Mappings to schema:Thing	449	334
Mappings to schema:Intangible	481	81
For the 250 examined mappings		
Correct mappings	235	67
Correct mapping rate	94%	26.8%
Unclear	10	27
Unclear rate	4%	20.8%
Errors	5	156
Error rate	2%	62.4%

TABLE 5.2: Comparison of the mapping algorithms

much, mapping to types at level 0.8, or 1.2 when excluding identical mappings. Looking at the data it is obvious that the hyponyms first approach much more frequently leads to mappings to schema:Thing and schema:Intangible. The hypernym first algorithm maps to schema:Thing 449, and schema:Intagible 481 times, while hypernyms then siblings maps to schema:Thing 334 and schema:Intangible 81 times. Neither schema:Thing nor schema:Intangible are very interesting mappings in the ontology. As described in section 2.6, schema:Thing is the most general category, meaning that every concept belongs to this category. On the other hand, schema:Intangible is described as "a utility class that serves as the umbrella for a number of 'intangible' things", and does not have any special properties in the ontology.

These results were a bit disappointing. One should however keep in mind the fact that schema.org is not a general ontology. As mentioned in section 2.6 the schema.org ontology is geared towards things that are relevant to search engines. The synsets that were used in this test covered a wide variety of topics. The top-level categories of schema.org were shown in figure 2.5 on page 17 When looking at the results it is the necessary to keep in mind that synsets that do not belong to any of those 10 categories cannot have a better mapping than schema:Thing in the ontology.

5.1.1 Correctness of the algorithms

To find which algorithm performed best it was decided to check the results where the algorithms gave different mappings. In the cases where the mappings were the same, the two algorithms obviously performed equally well since they mapped to the same concept. The results from when the two algorithms gave different mappings they were checked manually to judge their correctness. The process consisted of looking up each synset that was mapped, and the types it had been mapped to and check if the two corresponded. The results were divided into three categories. If it was clear that the synset and type corresponded, they were marked as correct. If it was clear that they did not correspond they were marked as incorrect. There were also some cases where it was unclear whether or not a mapping was correct.

It was decided that it was necessary to include a category for unclear mappings to highlight the fact that some of the categories are fuzzy and require some more documentation, or that they might entail things than seem unnatural. One of the instances were it was unclear if a mapping should be judged to be correct was for the mapping of dairy#n#1, "a farm where dairy products are produced", which maps to schema:FoodEstablishment. Schema:FoodEstablishment is described as "[a] food-related business", which a dairy most certainly is. The sub typing of schema:FoodEstablishment however seems to indicate otherwise. The sub types of schema:FoodEstablishment are:

- Bakery
- BarOrPub
- Brewery
- CafeOrCoffeeShop
- FastFoodRestaurant
- IceCreamShop
- Restaurant
- Winery

This seems to indicate an establishment where private customers come to purchase goods, making a more industrial venue seem out of place. Another schema.org term that provided some difficulty was schema:Place, which has the description "[e]ntities that have a somewhat fixed, physical extension". Again the sub types seem to indicate that it should be used for geographical sections. From the description of the type it is unclear if it can be used to describe things like borders and edges of things. This would depend on what the thing should be fixed with regard to.

Since the manual inspection of the mappings was a time consuming process I decided to only inspect 250 mappings, and see if the results of checking these would be sufficient to say anything about the algorithms. Mappings to schema:Thing and schema:Intangible were excluded as one could normally argue reasonably for these.

A higher error rate in the hypernym then sibling algorithm had been predicted, but the difference in the error rate between the algorithms was much larger than anticipated. Again pointing to table 5.2 we can see that the hypernyms first algorithm made correct mappings in 94% of the test cases, while giving incorrect mappings in 2.0% test cases and questionable mappings in 4% of the cases. The hypernym then sibling algorithm on the other hand gave a correct mapping in only 26.8% of the cases checked. It gave incorrect mappings in 62.4% and unclear mappings in 20.8% of the cases. The fact that it gave correct mappings at a rate of close to 25% of the instances where the results were different was very surprising. This high degree of incorrect mappings indicates that using sibling synsets as the basis for mapping in unfruitful. The causes of the incorrect mappings will be examined in the next section, to see if they are caused by weaknesses in the algorithms, or in WordNet or the mapping from WordNet to schema.org.

5.1.2 Analyzing the sources of error

The fact that the hypernyms first algorithm gave incorrect mappings at all is a bit alarming. As described in section 2.2.1 about hyper- and hyponymy, hypernymy should be a transitive "type of" relation. Each hypernym should then be a more general type of the synset provided and not break the semantics of the synset. The mappings from WordNet to schema.org indicate that the semantic content of the synset are equal to the semantic content of the schema.org type mapped to. Since both hypernymy and mappings should preserve the semantics of the synsets, the mappings should be correct.

The fact that the hypernyms first algorithm gives incorrect mappings indicate that either the integrity of the hypernym relation is broken, or that the mappings are incorrect.

All the cases where the hypernyms first approach was deemed to have incorrect mappings were instances where the synset mapped to schema:Quantity via their hypernym measure#n#2 {measure, quantity, amount}. We can see in figure 5.1 how the different synsets are related to measure#n#2.

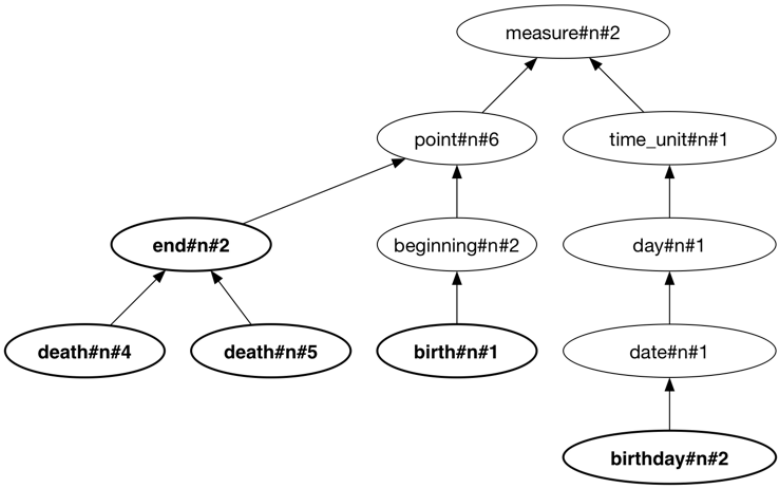

FIGURE 5.1: How the synsets with incorrect mappings are related to measure#n#2

In schema.org the type schema:Quantity is described as "[q]uantities such as distance, time, mass, weight". The synset measure#n#2 is described as "how much there is or how many there are of something that you can quantify". The mapping between these appears reasonable and do not seem to be the cause of error.

What makes the mappings seem unreasonable is that the synsets that were mapped incorrectly represent events that happen at a single instance in time. The problem in the chains seem to be that one goes from a synset which represents something that happened at a single instance in time, to a synset which describes a quantity of things. Using the "is-a" test, it does not seem reasonable that an english user would say that a birth is a type of quantity. The cause of the errors then is an inconsistency within the hypernym relation in WordNet.

I have reported these errors to the maintainers of WordNet, and they will be fixed in a future public release of WordNet (C. Fellbaum, personal communication, May 13, 2013).

The results where the hypernym then siblings algorithm mapped incorrectly were also analyzed. The distribution of incorrect mappings can be seen in figure 5.2 (page 50).

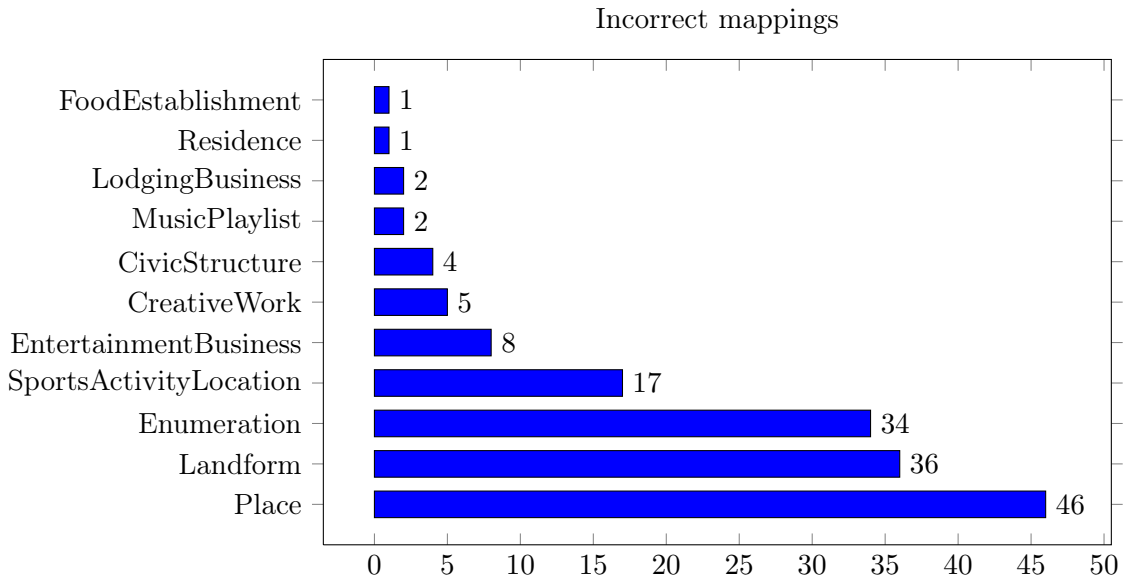

FIGURE 5.2: Distribution of incorrect mappings in hypernym then siblings algorithm

The two most common incorrect mappings were picked as objects of further study. I went through all the cases of incorrect mapping to schema:Place. This examination showed that in 45 of the 46 cases the mapping to schema:Place came as a result of the synset being a hyponym of whole#n#2, which has the sibling geological_formation#n#1 which again has a mapping to schema:Landform, a sub type of schema:Place. The last instance is the mapping from attention#n#3 which has the sibling tourist_attraction#n#1 mapping to schema:TouristAttraction.

For the mappings to schema:Landform all of the incorrect mappings came as a result of the synsets being hyponyms of part#n#3. That synset has the hypernym thing#n#12, which has the hyponym body_of_water#n#1. Body_of_water#n#1 maps to schema:BodyOfWater, which again is a sub type of schema:Landform. Common for all these mappings is that the hypernyms are reasonable. There is no reason to believe that these are cases where the hypernym chain breaks the semantics of the synset. The mappings from WordNet to schema.org also seem to be sound.

The analysis seems to exclude both the hypernym relation and the mappings from WordNet to schema.org as causes for the incorrect mappings. That leaves the possibility that using the siblings caused the error. Using the mappings from the siblings were wrong in

these cases, as the sibling had some other semantic value than the synset for which the algorithm tried to find a mapping.

There were instances of siblings both of the original synset and of the hypernyms, so it does not appear that closeness to the synset needs to have any influence on whether or not the sibling gives an accurate mapping for the synset.

5.1.3 Choosing an algorithm

From the previous discussion and from the results of the testing it seemed clear that the hypernyms first algorithm is the dominant strategy for mapping synsets to schema.org, and that there is no difference between the two when mapping to SUMO. The algorithms performed very differently when mapping to the different ontologies.

It might be that an increase in the number of mappings would change the results. One could then try to refine the hypernym then siblings algorithm to analyze the mappings of its siblings, and try to find some more sophisticated way of choosing which mapping to select.

It would also be interesting to try mapping to a more general ontology than schema.org. As mentioned in section 2.6 schema.org is a small ontology with only 577 types, and with a bias towards commercial interests. It might be that an ontology that was larger, and which had a more balanced approach to the world would result in better results for the algorithm.

As it stands it is clear that the hypernyms first algorithm out performed the hypernym then siblings algorithm, and is the one that will be used in the artefact. The fact that its mappings were incorrect or questionable in 6% of the instances analyzed is unfortunate. The cause of the errors were however found to be outside the system that has been developed for this thesis, and they have been reported. This gives us reason to hope that the error rate of the system will go down as the tools it depends on are improved. It also suggests that some error reporting mechanism should be created to give feedback to the maintainers when incorrect mappings or hypernym relations occur.

5.2 Testing against existing markup

One of the success criteria was that the artefact should be able to mark up HTML as well or better than what is on the Web now. To test if MaDaME managed this I used Web pages that were already marked up with schema.org markup, and tried to generate similar markup using MaDaME before comparing the results to see if the metadata created by the tool was of a similar quality.

5.2.1 Method for comparing the results

The method that was used to mark up and compare the sites followed a simple process. Web pages were picked from a list of sites using the schema.org ontology for metadata that is available on GitHub [2].

The markup of the Web pages that were picked was examined to find which things that were marked up, and which schema.org types they were marked up as. This process was selected to make sure that it would be possible to compare the results.

The pages were imported into the artefact, and the markup would be recreated by selecting text, and if necessary supply keywords to disambiguate the content. For some sections supplying keywords were required as the text did not contain keywords that signaled the meaning of the content. There were several reasons for this. In some cases the keyword was displayed on the Web page as an image, so instead of having the term "review" in a header or in text it appeared as part of an image on the page. It could also be that the individual words on the page did not represent the concept. In other cases the difficulty was tied to mapping to functional concepts. For example is being for sale not an inherent property of thing it self, making mappings to products difficult using descriptions of the thing.

Metadata was added as faithfully as possible to make comparison simpler. When the markup process was complete, the Web pages would be exported and the resulting Web page would be analyzed. The original was used as a gold standard which the marked up pages could be compared to.

[2] `https://github.com/LawrenceWoodman/mida/wiki/Sites-Using-Microdata`

The analysis of the metadata would be performed by using Google's structured data testing tool [3], and W3s RDFa 1.1 distiller and parser[4]. The structured data testing tool shows the metadata that is read and extracted by Google. It was used to check if the metadata available to Google was the same. The tool was used on both the original page and the page marked up by the artefact so that one could judge if the extracted metadata was the same.

The RDFa distiller could not be used to compare the documents as it is created to distill RDFa not microdata, which was used to include metadata on the pages tested against. The RDFa distiller parses the Web page and extracts the RDFa, displaying it as some RDF format. It was used to see that the markup that was created was valid RDFa and that it could be translated to RDF.

The pages that were marked up using this process were:

- A restaurant review from the Telegraph
- A tour operators customer feedback page
- A tourist agency home page
- The home page of a marketing company
- A movie review sites review of a film

The full HTML of the Web pages before and after they were marked up using MaDaME can be found at `https://github.com/EivindEE/Master-thesis/tree/master/WildTesting`.

Most of the metadata on the Web pages that were marked up was metadata about larger sections of the page. The artefact is targeted towards disambiguating single words as described in section 4.2. When marking up the text this meant one had to decide whether to markup the same section of the text as used in the original page, or if one should select a single word in the section describing its content. Should one for example select the header "Review", or the entire review itself? Choosing the first option would give the metadata a different structure, while selecting the second option would require the user to provide a keyword describing the topic of the section. There is little practical difference

[3] `http://www.google.com/webmasters/tools/richsnippets`
[4] `http://www.w3.org/2012/pyRdfa/`

between the two strategies. Selecting a section might make it easier for humans looking at the markup to find out which part of the page the type refers to, but MaDaME does not use the content of the tag to add information so there is no difference in the semantic meaning of the metadata that is added. Neither should there be any difference in how the Web pages are displayed after the metadata is added. Marking up an larger section does however increase the amount of places where an error could occur. A combination of these two strategies was tried when marking up the pages.

5.2.2 An issue with the testing tool

An issue that was discovered when analyzing the resulting markup with the structured data tool was that it returned error messages saying that some of the elements that had been marked up were missing required properties. When analyzing a schema:Product without a name value the error message reads:

```
Warning: Missing required field "name (fn)".
Warning: Incomplete rdfa with schema.org.
```

The documentation on the schema.org homepage does not give any indication that schema types have required properties. The initial reaction to this was annoyance that these fields were required by RDFa but not by microdata. Closer inspection of the unmodified Web pages however revealed that when using the microdata format these types were ignored without warning. It is positive that the RDFa created by the artefact gives an error message instead of failing silently, but it is not good that the testing tool describes a field as required when this is not mentioned in the documentation. The fact that the validator requires the properties does not in itself mean that the markup is incorrect, or that the syntax is wrong. The validator is targeted towards the generation of rich snippets, and it might be that the warning is intended to warn that the amount of metadata is insufficient to generate rich snippets. The warning that required fields are missing might then refer to the fact that they are required to create a snippet, not required for the metadata to be valid. The metadata generated in these instances turned out to be of equal quality as that of the original documents, but also showed that the tool could do more to promote attributes that are required by search engines.

5.2.3 Comparison of the results

During testing it was realized that there are some schema.org types that the system was not able to reach. In particular it was discovered that the system does not have a way to reach the aggregated schema.org types. At the time of writing the aggregate types in schema.org are schema:AggregateRating and schema:AggregateOffer. The type schema:AggregateRating is meant to represent the average rating that a rated object has received. The type schema:AggregateOffer on the other hand represents a collection of offers for a given product. The difficulty posed by both these types is that what separates them from their super types schema:Rating and schema:Offer is that they represent the plurality of the super type. The system uses WordNet to represent and disambiguate words, and as its basis for mapping natural language to ontologies. WordNet is a dictionary type system, which does not separate between the different grammatical numbers of a given word, as they all represent the same concept. This could be an issue for the system as a whole since it means that the language used as an intermediary between natural language and the ontologies does not capture all of the complexity of the ontologies the system is supposed to map to.

The schema.org types that were used on the Web pages were (the number of times used in parenthesis):

- Review (11)
- Product (11)
- ImageObject (11)
- TravelAgency (3)
- Article (3)
- Rating (1)
- Movie (1)
- Person (1)
- People (3 - non-existing type)

The types that the Web pages mapped to were so high level that they all had natural direct mappings, meaning that the best-fit algorithm did not have to be used. Adding the properties to these were simple, as the tool provided a list of the properties that the type was able to have.

It was found that one of the pages used illegal markup. The page with the movie review it was found that the author of the HTML had illegal types and properties. The movie review page referred to people as schema:People, while the correct type in schema.org is schema:Person. The page used this pseudo-type to refer to multiple people by providing a name property to the type for each person. This usage in not possible in MaDaME as it does not allow multiple properties. The Web page also used the property name "publishdate" while the correct name of the property is "datePublished".

Mistakes like these are simple to make since it is natural for humans to think that people and person, or publishing date and date published are two ways of saying the same thing. This is however the ambiguity that we want to remove for computers by using metadata. An advantage of using a tool like MaDaME is that one can be sure not to use incorrect types or properties such as this. The tool will only allow types and properties that exist in the schemas that are included.

MaDaME was able to create mappings to all the types that were used on the Web pages. It was also able to add all the properties that the types had. In addition it was able to correct incorrect markup. These results show that the tool is able to create metadata of equal quality as that which is present on the Web now.

5.3 Browser rendering

One of the criteria for the artefact was that it should leave the visual representation of the system unchanged. The system is meant to let users add metadata to existing Web pages, and should not modify their appearance. I did a comparison of the Web pages before and after adding metadata. An example of the telegraph Web site before adding metadata can be seen in figure 5.3, and after in 5.4. The side margins and the header of the Web site has been cropped out in the images to make it easier to see the content of the sites.

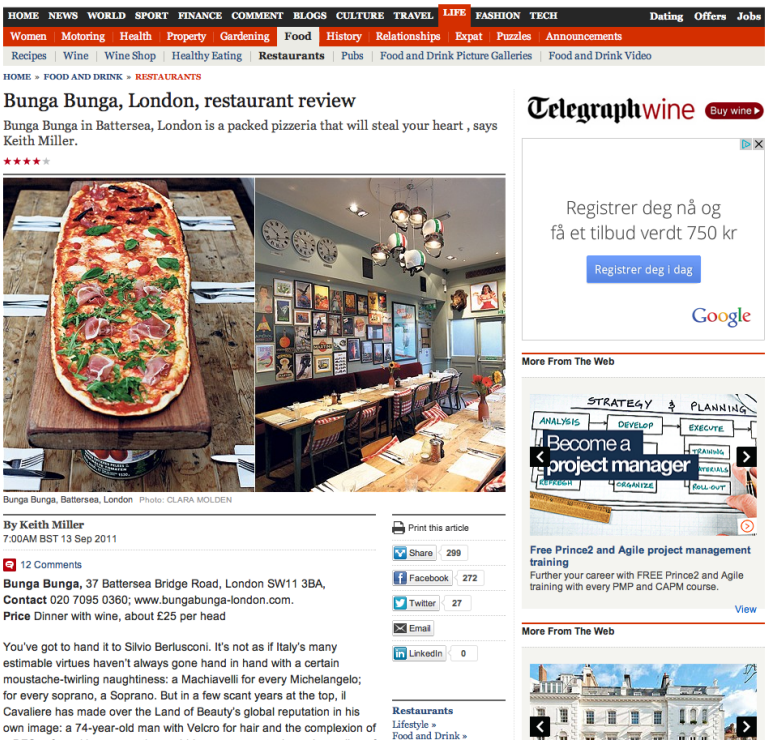

FIGURE 5.3: The Telegraph Web page before the metadata was added

Looking at the figures we can see that the overall layout of the images is unchanged. There are however some changes to the HTML.

To make the Web page appear the same when viewed on the URL the user is given when the page is exported the relative URLs tied to images and CSS have been replace with absolute URLs. To keep the code simple only the URLs that start at the root of the domain were changed. So if there was an image on the page `http://example.org` which pointed to the source `/image.png` then the source would be changed to `http://example.org/image.png`. This change is not communicated to the user, which is unfortunate, but it is possible to change if it leads to issues.

As one can see from the images there are some sections of the Web site which are not displayed, or more precisely they are displayed as loading. The switch from relative to absolute URLs have only been done for images and CSS, which means that some of the JavaScript modules that the Web site depends on will not be loaded into the Web page. The tool tries to leave the HTML as much intact as possible, and since the scripts have yet seemed crucial to display the page properly it was decided to leave them as they were.

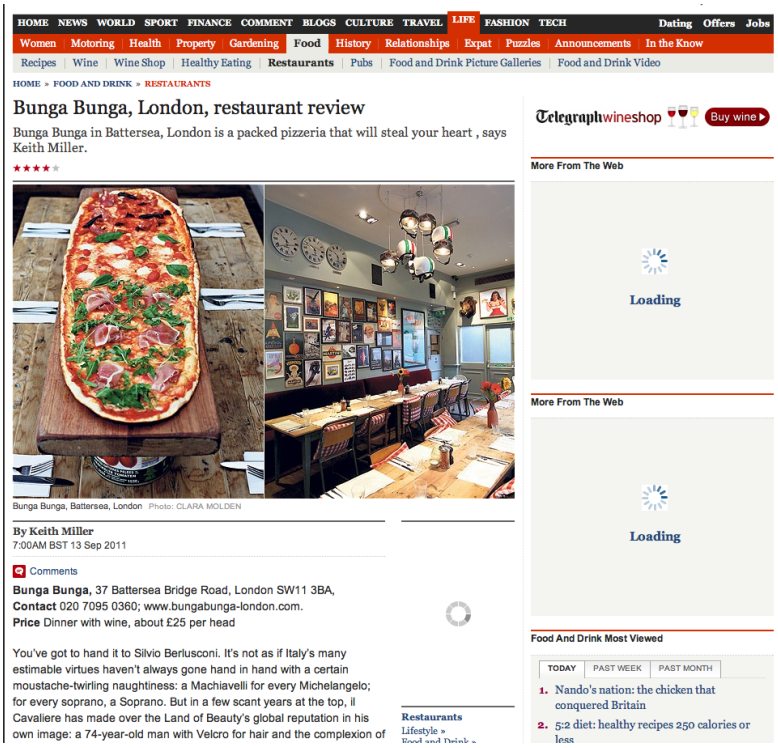

FIGURE 5.4: The Telegraph Web page with metadata added

It should be noted that this means that the page will sometimes miss some content, as visible in the figures.

One issue that might arise is if some of the content on the site, either CSS or JavaScript, depend on the order of the elements or their direct placement in the hierarchy. Since MaDaME is intended to add metadata to user selected sections of the page the system had to to create its own HTML elements to attach the metadata to, since that is the only reliable way to talk about just that section. There are CSS selectors that select either direct children or the n-th children of an element. Since the system needs to add elements it might be that the element that was added cause another element to either not be the direct child of the former parent, or that it changes the elements number amongst the children. In a similar way, DOM manipulation in JavaScript sometimes uses the children of an element or the index of an element in a list of children. I have not found any way to avoid this issue, and can only hope that most developers depend on id and class attributes instead of the specific location in the DOM.

5.4 Analysis of usage data

In addition to the testing described earlier in this chapter the Web application has also been released on the Internet. This has allowed us to see how users who do not know the system would interact with it, and if they could use it to add metadata.

The data that has been gathered at this point is inconclusive, but can hopefully give some indication of how the tool will be used. The usage data that has been collected contains the IP of the user. This information was used to exclude users from the University of Bergen IP-range, as this data would mostly consist of usage tied to testing or debugging the system in the development phase. The usage data consists of the data from the 8^{th} of April to the 12^{th} of May.

As mentioned, the tool is mainly targeted towards adding metadata to single words, with adding data to sections being a secondary means of input. This is consistent with how the users have used MaDaME. From the usage data collected we can see that of the 298 of the text selections requests the server has received, only 12, or about 4%, have been for selections so big that they have been categorized as sections of text. The other 96% of the selections were of single words or of single entities or concepts like "Alexander Graham Bell" or "semantically classified lexical databases".

I have not found any cases of the users have used the resulting HTML on their Web pages, so it is not possible to say anything about their satisfaction with the page at this point.

Chapter 6

Summary and Conclusion

Before starting the work with this thesis I saw that there was a need for a system that would lower the barrier of entry to added semantic metadata to Web pages. I believe that having a Web of linked semantic data will become increasing important to find relevant information as the amount of information on the Internet continues to grow.

The advent of schema.org has made it easier to embed metadata on Web pages. The scope of the ontology however is largely limited to commercial and search engine specific concepts. In addition to this the microdata format pushed by the authors of schema.org does not allow for mixing vocabularies. I wanted to improve the situation by making it possible to add metadata about arbitrary content. My goal when starting work with this thesis was to create a prototype of a system that would allow users to add metadata to Web pages by using natural language, without requiring the to know the formal underpinnings of ontologies.

A research question was formalized that stated:

"Is it possible to create a tool which allows naïve users to easily add metadata to their Web sites using natural language?"

To answer this question there a number of sub-questions that has to be answered. How should users pick the parts of a Web page they want to add metadata to, and find the concepts it describes using natural language? Is WordNet suitable for representing disambiguated concepts from natural language in a way that will allow us to map these concepts to formal ontologies? How should an algorithm be implement to finds mappings

from the natural language concept to types in formal ontologies in a way that preserves the semantic content of the concept? Is it possible to add metadata to Web pages in such a way that it does not change the way the page is rendered by browsers?

In this thesis I attempted to use WordNet as a method of representing natural language. WordNet was developed to have word boundaries corresponding to how humans mentally represent concepts. Using WordNet also made it possible to build on earlier work on creating mappings between WordNet and formal ontologies for the Semantic Web. WordNet also contains the concept of the hypernym, a relation that could be utilized to find mappings to higher level concepts if the synset itself did not contain a direct mapping to an ontology.

I examined the two algorithms for finding the best mapping from a given synset into the ontologies the system was mapping to. These were later evaluated to find which of them gave the best mappings.

A Web application was created to lets users add metadata to Web pages by selecting content on the page, and disambiguating the selection by clicking on suggested interpretations of the selection. MaDaME also allows users to import and export Web pages into the Web application for mark up.

6.1 Findings

I will now summarize the results of the analysis that was done in chapter 5.

The results found to a large degree supports the feasibility of using WordNet as a way to represent natural language when mapping to formal ontologies. Some cases where the integrity of the hypernym relation was violated were found. These errors have been reported to the maintainers of WordNet, and they will be fixed in the next public release (C. Fellbaum, personal communication, May 1, 2013 and May 13, 2013). I found that WordNet was unable to capture the grammatical number of natural language. This means that the tool will not have any way to distinguish between ontological types that differ in this respect. There were only a few instances in the ontologies that were utilized in the thesis of this flaw hindering the completeness of the mapping.

It was found that using hypernyms as a basis for mapping gave correct mappings. The incorrect mappings that were discovered during analysis were found to be caused by errors outside of the system. One should continue to look for further discrepancies in the future to help the development of these tools as well. The hypernyms first approach does however result in high-level mappings, and could benefit from further refinement.

The metadata that was created using MaDaME is comparable to that present at current Web sites which use schema.org to enrich their content. The testing also showed that the tool could help users avoid using incorrect types and properties, since it limits the properties allowed to add to those that are defined as belonging to the type.

As described in section 5.3 the testing of how the Web pages were rendered after metadata was added by the tool showed that the documents were displayed in the same manner before and after metadata was added. My analysis did uncover cases in which adding metadata could potentially change how the page was displayed. The tests did however find that the system was able to add metadata to the Web pages without changing the way they were displayed in the browser.

6.2 Further work

The goal of this thesis has been to create a functioning prototype of an artefact that allows users to add metadata to Web pages by using natural language. The system has been able to fulfill the most basic requirements, and shown that the concept is feasible. There are now several new interesting ways the tool could be developed further to increase its value to users, and to researchers.

It would be interesting have mappings to more ontologies, and one could offer the user a chance to say what the topic of the page was. In this way one could offer mappings to the Friend Of A Friend ontology if it was a Web page dealing with social interaction, the Good Relations ontology if it was a commerce page and so on. To do this the mapping module should get further development to complete the process of uncoupling the ontologies from the code.

There should be developed a way to allow for multiples of a single property on the Web page. The idea of adding properties came quite late in the project, and is as a

consequence not as feature rich as it should be. One should also research into finding some way of allowing for properties from other ontologies. One difficulty here would be finding a way of presenting these without overwhelming the user.

Adding multiples might also mitigate the issue that synsets are not regarded as distinct because of their grammatical number. For schema.org the issue of aggregated terms is limited as it only has two types that are the aggregation of multiples of a type. By allowing multiples of properties, the system could handle adding the aggregation of these behind to the document automatically. This would be a good solution for the issue with schema.org, but it might not scale well if the system is expanded to include other ontologies that separate concepts by way of grammatical number.

When the Web site has experienced more usage it would be exciting to examine the usage logs to see which types of text gets tagged. Actual usage data would be an interesting source to discover concepts that users frequently want to map. Examination of these logs could therefore be a useful starting point to find out which ontologies to create mappings for, and which parts of these ontologies which would create the most value for the users. The usage data collected will make it possible to extrapolate text that the system did not find good disambiguations for by assuming that this is text that was selected but where the user did not click any of the suggested senses. It is also possible to count the frequencies at which different synsets or DBPedia terms were chosen as the concept the user wanted to describe, and use this to target the work of mapping.

6.3 Conclusion

My goal in this thesis was to answer the question if one could create a tool that let users add metadata to a Web page using natural language. I have now described the process of developing the prototype of MaDaME and the testing and analysis of the results. The findings have produced positive results for the main research questions. I have found a representational language that can capture the central semantics of natural language, and a means of mapping this language to ontologies with the help of mapping files. I have also managed to add the metadata to the Web pages without changing the rendering of the Web pages.

I acknowledge that there is still a lot of work that needs to be done before the system is finished. The system has traded expressiveness for simplicity, both to make it easier to use and to make the scope of the project manageable. The prototype has however been capable of answering my research question, and I feel confident that it has demonstrated the feasibility of creating a system that creates semantic metadata by utilizing natural language.

Bibliography

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). DBpedia: A Nucleus for a Web of Open Data. In Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., and Cudré-Mauroux, P., editors, *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735, Berlin, Heidelberg. Springer Berlin Heidelberg.

Auer, S. and Lehmann, J. (2007). What have Innsbruck and Leipzig in common? Extracting Semantics from Wiki Content. In *In ESWC*, pages 503—-517.

Bang, B. H. K., Dané, E., and Grandbastien, M. (2008). Merging semantic and participative approaches for organising teachers' documents. *Proc Conf. Educational Multimedia, Hypermedia & Telecommunications, pages*, pages 4959–4966.

Benzmüller, C. and Pease, A. (2012). Higher-order aspects and context in SUMO. *Web Semantics: Science, Services and Agents on the World Wide Web*, 12-13:104–117.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web - A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 284(5):34–+.

Brooks, C. H. and Montanez, N. (2006). Improved annotation of the blogopshere via autotagging and hierarchical clustering. *Proceedings of the 15th international conference on World Wide Web (S. 625-632). Edinburgh, Scotland: ACM.*

Chen, M., Fuhrt, B., and Purdin, T. D. F. (1990). Systems Development in Information Systems Research. *Journal of Management Information Systems*, 7:89–106.

Collins, A. M. and Quillian, M. R. (1969). Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior*, 8(2):240–247.

Fillenbaum, S. and Jones, L. V. (1965). Grammatical contingencies in word association. *Journal of Verbal Learning and Verbal Behavior*, 4(3):248–255.

Gantz, B. J. and Reinsel, D. (2011). Extracting Value from Chaos State of the Universe : An Executive Summary. Technical Report June, IDC.

Golder, S. and Huberman, B. A. (2005). The Structure of Collaborative Tagging Systems. *Audio, Transactions of the IRE Professional Group on*.

Gruber, T. (2007). Ontology of Folksonomy: A Mash-up of Apples and Oranges. *International Journal on Semantic Web & Information Systems*, 3(2):1 – 11.

Gruber, T. R. (1993). Toward principles for the design of ontologies used for knowledge sharing. In *International Journal of Human-Computer Studies*, volume 43, pages 907–928.

Guarino, N. (1998). Formal Ontology and Information Systems. In *FOIS '98: Proceedings of the international conference on Formal Ontology in Information Systems*. IOS Press.

Guha, R. (2011). Introducing schema.org: Search engines come together for a richer web. http://googleblog.blogspot.no/2011/06/introducing-schemaorg-search-engines.html.

Hebeler, J., Fisher, M., Blace, R., Perez-Lopez, A., and Dean, M. (2009). Modeling Information. In *Semantic Web Programming*, pages 63–92. Wiley, 1 edition.

Hevner, A. R., March, S. T., and Park Jinsoo, R. S. (2004). Design Science in Information Systems Research. *MIS quarterly*, 28(1):75–105.

Kim, H. L., Decker, S., Scerri, S., Breslin, J. G., and Kim, H. G. (2008). The state of the art in tag ontologies: a semantic model for tagging and folksonomies. *Proceedings of the 2008 International Conference on Dublin Core and Metadata Applications*, pages 128–137.

Mayer, M. and Menzel, J. (2009). More Search Options and other updates from our Searchology event. http://googleblog.blogspot.no/2009/05/more-search-options-and-other-updates.html.

Mika, P. (2005). Ontologies are us: A unified model of social networks and semantics. *The Semantic Web–ISWC 2005*, pages 522–536.

Miller, G. A. (1990). Nouns in WordNet: A Lexical Inheritance System. *International Journal of Lexicography*, 3(4):245–264.

Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. J. (1990). Introduction to wordnet: An on-line lexical database. *International journal of lexicography*, 3(4):235–244.

Niles, I. and Pease, A. (2001). Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems - FOIS '01*, volume 2001, pages 2–9, New York, New York, USA. ACM Press.

Nunamaker Jr, J. F. and Chen, M. (1990). Systems development in information systems research. *System Sciences, 1990., Proceedings of the Twenty-Third Annual Hawaii International Conference on*, 3:631–640 vol. 3.

Passant, A. and Laublet, P. (2008). Meaning Of A Tag: A collaborative approach to bridge the gap between tagging and Linked Data. *Proceedings of the WWW 2008 Workshop Linked Data on the Web (LDOW2008), Beijing, China.*

Pemberton, S., Adida, B., McCarron, S., and Birbeck, M. (2008). {RDFa} in {XHTML}: Syntax and Processing. {W3C} recommendation, W3C.

Pretorius, A. J. (2004). Ontologies-Introduction and Overview. *Semantic technology and applications research laboratory.*

Ronallo, J. (2012). HTML5 Microdata and Schema.org. *The Code4Lib Journal*, (16).

Shirky, C. (2007). Shirky: Ontology is Overrated – Categories, Links, and Tags. http://www.shirky.com/writings/ontology_overrated.html.

Tang, J., Leung, H., Luo, Q., Chen, D., and Gong, J. (2009). Towards ontology learning from folksonomies. *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 2089–2094.

Tennison, J. (2011). Lessons for Microdata from schema.org. http://www.jenitennison.com/blog/node/156.

Tonkin, E. and Guy, M. (2006). Folksonomies: Tidying up tags. *D-Lib*, 12(1).

Veres, C. (2011). LexiTags: An Interlingua for the Social Semantic Web. In *Proceedings of the 11th Interational Semantic Web Conference ISWC2011*, pages 1–12.

Weinberger, K. Q., Slaney, M., and Van Zwol, R. (2008). Resolving tag ambiguity. In *MM '08: Proceeding of the 16th ACM international conference on Multimedia*.