Copyright Statement

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

This thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author’s right to be identified as the author of this thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author’s permission before publishing any material from their thesis.

To request permissions please use the Feedback form on our webpage.
http://researchspace.auckland.ac.nz/feedback

General copyright and disclaimer

In addition to the above conditions, authors give their consent for the digital copy of their work to be used subject to the conditions specified on the Library Thesis Consent Form and Deposit Licence.

Note: Masters Theses

The digital copy of a masters thesis is as submitted for examination and contains no corrections. The print copy, usually available in the University Library, may contain corrections made by hand, which have been requested by the supervisor.
Enhancing Wikipedia with Semantic Technologies

Lian Hoy, Lee
February 2011

Supervisors:

Christof Lutteroth
Gerald Weber

A thesis submitted in partial fulfillment of the requirements of
Master of Science
Abstract

As the amount of content grows on the Web, there is an increasing need to provide greater search capabilities that produce relevant results. Users should be given the capability to execute complex queries in order to provide greater accuracy in their searching endeavours over the Web. The semantic Web promises to provide such a feature by making the concepts within data explicit.

We examine the benefit and feasibility of implementing semantic technologies over Wikipedia, which is one of the largest repositories of knowledge at present. In our project we begin by creating a corpus of semantic data which we extract from Wikipedia. Then, we create a platform in order to cater for semantic user interface development as separate modules. Finally, we create several semantic user interface modules to test the effectiveness of such an implementation.

The results from our project is that a platform to allow access to semantic data over existing content is not only possible, but an effective way to encourage development of semantic technologies. In our project, we were also able to create a way in which could leverage semantic data to allow users to perform a query based on a range of values with the guidance of a graph to visualize the distribution of results.
I would like to acknowledge and thank: Christof Lutteroth and Gerald Weber for their assistance and guidance throughout the project, as well as Ted Yeung and Daniel Zeng for their assistance with using PDStore.
# Contents

1 Introduction  
1.1 Semantic Web .................................................. 2  
1.2 Wikipedia ....................................................... 2  
1.3 Semantic Wikipedia ............................................... 4  
1.4 Research Aims .................................................. 4  

2 Understanding the Semantic Data Model  
2.1 Semantic Data .................................................. 7  
2.2 Ontology .......................................................... 10  
2.3 Consuming Semantic Data ......................................... 12  

3 Extraction of Semantic Data from Wikipedia  
3.1 Infobox ............................................................ 17  
3.2 Categories and Lists ................................................. 22  
3.3 Links .............................................................. 23  
3.4 Extraction Framework ............................................. 27  
3.5 Synchronizing with Wikipedia ..................................... 31  
3.6 Evaluation of the Semantic Data Extraction Process ............. 32  

4 Querying Semantic Data  
4.1 Semantic Querying compared to Keyword Search .................. 33  
4.2 Semantic Queries ................................................ 36  
4.3 Semantic Query Language ........................................ 38  

5 User Interfaces for Querying Semantic Data  
5.1 Enhancing Wikipedia’s Search Functionality ...................... 44  
5.2 Graph Pattern Builder ............................................ 45  
5.3 Faceted Browsing ................................................ 49  
5.4 Enhancing User Applications ..................................... 54  
5.5 Other Possible User Interface Enhancements ...................... 55
6 Related Work
6.1 DBPedia .......................................................... 63
6.2 WordNet .......................................................... 64
6.3 Yet Another Great Ontology (YAGO) ....................... 64
6.4 FreeBase ........................................................ 64
6.5 Semantic Wiki Applications .................................. 65
6.6 Natural Language Processing (NLP) ......................... 66

7 Conclusion
7.1 Achievements .................................................... 69
7.2 Future Directions ................................................ 70
7.3 Reflections ....................................................... 70
Introduction

From ease of access to information to keeping in touch with loved ones, the Web has brought great change to how we live our lives. As the Web grows, it is important for it to evolve as well in order to remain relevant in the years to come. When the Web started, it was made up of mainly text with hyperlinks taking users from page to page; much like reading a book and having an index to quickly find content of interest.

As the content on the Web grew, keyword search came along to make it easier to find content on the Web. To help enhance the capabilities of keyword search, tagging systems have been introduced to help group similar content together based on a keyword phrase [11].

The Web is continuously growing and evolving and as its content increases over time new problems arise such as duplication of keywords to a point that keyword search is finding it difficult to return relevant results [14]. At present, the Web is also shifting towards a medium for dissemination of other forms of media such as video and audio. This poses new challenges in providing users the ability to search over these formats of data. Due to the vast amount of information available on the Web at present, many actions carried out need some degree of automation such as authentication of a user [20, 21].
1.1 Semantic Web

As we move towards the age of the semantic Web, the amount of semantic data being made available has grown rapidly [5, 9]. With the availability of structured data comes greater possibilities in the way that we access data [9]. One of the most prominent possibilities is in the way that we search for content over the Web.

Through the use of semantic data, we are no longer limited to keyword searches and are able to carry out complex queries [3, 8]. This comes in the form of being able to query the available information in a machine processable fashion [2, 14]. At present, semantic querying is more prevalent in organizational intranets because of well defined ontology structures within an organization [8].

The main goal of the semantic Web is to produce a way in which information on the Web can be interpreted by machines, which is achieved by making concepts within data explicit through the use of semantic annotations. [20, 24, 15]. The semantic Web also aims to provide interlinked knowledge by integrating the information stored on the Web into a graph with a shared structure to enable a common way to access data but manage it in a decentralized manner [2, 14].

With the development of the semantic Web, the semantic data that is produced can be used in the development of artificial intelligence because of its machine processable trait [27]. In artificial intelligence, semantic data can serve as a body of knowledge to provide reasoning.

The semantic Web breaks the barrier of having to decorate data in non-textual format to allow for keyword search engine to be able to locate it [9]. Instead, data is annotated with properties that describes it and queries can be written to match the appropriate properties for more accurate results.

In order to enhance the capabilities of the Web, knowledge bases are increasingly used to provide smarter interaction with the users [5]. However, current knowledge base capabilities is limited and domain specific as it is very costly to build and maintain. Using a similar approach to Wikipedia where everyone can contribute would greatly reduce the cost of creating and maintaining a knowledge base for the Web. As articles in Wikipedia covers a wide range of knowledge, that can be carried over when it is extracted to form a body of semantic information [5].

1.2 Wikipedia

Wikis are a great tool for collaborative authoring and publishing of informational articles as it is easy for authors to quickly produce and share content [3]. Wikis are designed to be quick and easy to create content in order to encourage mass collaboration from
1.2 Wikipedia

contributors [6, 11, 14, 18, 20]. Something that starts of as a small article can be extended by many contributors to create a comprehensive information reference on a particular topic.

While Wikis have been deployed as general purpose collaboration tools, one of the most well known usage of Wikis is Wikipedia\(^1\), the largest encyclopedia at present which is co-authored by multiple communities all around the world [3, 6, 2]. It is available in multiple languages and the number of articles it contains is in the millions. Wikipedia growth can be attributed to its availability on the Web where users need only a Web browser to be able to access and contribute [20].

One of the features provided by Wikipedia is that it provides versioning over the changes made by contributors so that the history of an article can be tracked over time [3]. This enables authors to easily view the changes other authors have made and append their contributions to suit. When incorrect edits are made by mistake or though malicious use, versioning also enables users to quickly and easily remove them by rolling out of specific versions.

Wikipedia works by evolving and adapting to the needs of its users, therefore it has no enforced structure [6]. This works well for users as it is similar to mind maps where there is not much structure and things are linked to each other just in a much larger scale [6]. While this allows great flexibility and convenience to the user, it greatly restricts the ability to apply any sort of machine learning or data processing capabilities onto it. As the amount of content in Wikipedia grows, the lack of structure and organization makes it very difficult to navigate [11].

To maintain the usefulness of Wikipedia, users need to be able to find relevant articles quickly and easily [11, 14]. If existing articles cannot be found, contributors may end up creating similar content causing redundant copies in Wikipedia [11]. These redundant copies may also lead to confusion and misinformation if different users were to stumble upon different versions.

While Wikipedia is a Wiki focused on creating an all encompassing encyclopedia in multiple languages with the assistance of contributors worldwide, there are other Wikis such as Wikia\(^2\) which focus on a specific domain [20]. Wikia hosts multiple small Wikis, each containing information specific to a particular domain such as a Wiki for information on *Japanese Cooking Recipes* or a Wiki with information on *brewing beer*.

---

\(^1\)http://www.wikipedia.org

\(^2\)http://www.wikia.com
1.3 Semantic Wikipedia

There are many ways that Wikis can be extended, but the one that captures the interest of many researchers is that of providing semantic processing capabilities for Wikis [3, 24]. Semantic Wikis extend the capabilities of a Wiki by allowing it to be processed by machines [9, 21]. In recent years, there has been quite a lot of research which aims to introduce semantics into Wikipedia [2, 20, 22, 14, 24].

One such project is the Semantic Wikipedia project by Völkel et al., where they propose the integration of typed links and page attributes as a special syntax [3, 24]. While this provides the ability for semantic processing over Wikipedia, it goes against Ward Cunningham’s design principle as it adds to the complexity of authoring articles [3].

The Wikipedia syntax allows users to create links between articles just by specifying the name of the article to link to within the article [6]. This creates a connected graph like pattern with multiple links between nodes. Applying semantic information to Wikipedia is basically annotating the links with the appropriate relationship to form a directed graph [6, 5].

Currently, Wikipedia is limited by the capabilities of full-text search, where search results returns articles containing the keywords that is specified as the search phrase [7]. By introducing semantics into Wikipedia, it is possible to create a semantic query engine that understands the users search phrase and be able to return more relevant results [7, 9, 24]. For example, we could perform complex queries such as list all towns by the sea in New Zealand, and we would obtain a list of towns that Wikipedia has which is by the shoreline.

Semantic data can also be used to assist users in enhancing the quality of the articles on Wikipedia [7, 20]. Semantic data can be used to automatically identify missing information and prompt the user complete it. For example, Wikipedia contains information on many cars but some articles on cars do not list their manufacturer. As all cars would have a manufacturer, a rule can be specified so that all articles on cars that do not have a manufacturer defined would prompt the user for the name of the manufacturer.

1.4 Research Aims

As part of our project, we would like to achieve the following goals:

**Have an understanding of semantic data.** We would like to have an in-depth understanding of the what is semantic data, why is it important and how it is used.

**Create a way to access semantic data.** We want to develop a way in which we can apply semantic data to existing content over the Web.
Discover new ways to access data. We hope to discover new ways in which we can allow users to interact with content by leveraging the use of semantic data.

1.4.1 Structure of this Thesis

Our thesis is broken down into 4 main chapters: Understanding the Semantic Data Model, Extraction of Semantic Data from Wikipedia, Querying Semantic Data and User Interfaces for Querying Semantic Data. It is followed by a chapter of related work and the conclusion. Chapter 2 describes the concept of semantic data while Chapter 3 covers the extraction of semantic data from Wikipedia. Chapter 4 examines the use of semantic data to provide the ability to execute complex queries on Wikipedia. In Chapter 5, we will be discussing different ways in which we can make the user experience better when using Wikipedia, through the use of semantic technologies. Chapter 6 covers some of the work done by other researchers on the topic of semantic data. Chapter 7 concludes the thesis, points out future directions and adds some final reflections and remarks.
Before we delve into the depths of semantic technology, it is important to understand the different components of the semantic data model. In this chapter, we examine the fundamental components of the semantic data model and how they relate to each other. We will describe how they are created, the schema that it adheres to and ways in which they can be consumed.

The first section introduces what is semantic data and how it is derived. In the next section, we describe ontologies and how it is used as a schema for access to semantic data. In the last section of this chapter, we present the ways in which semantic data is queried and consumed. To assist in describing the concept behind the semantic data model, we will refer to the recommendations by the World Wide Web Consortium (W3C) where available.

2.1 Semantic Data

Semantic data is basically similar to metadata, which is data that describes data. In the semantic data model, the relationship between entities is made explicit in order to make it accessible for machine interpretation [14, 21].

As seen in Figure 2.1, the entity containing the data dog and cat is annotated with
similar relationships that link them to the entity *animal* as both dog and cat is a type of animal. The relationship between entities has to be directed from one entity to the other. In the case where the relationship is the same in both directions, the relationships has to be defined twice; one for each direction.

The reason behind having directed relationships is that it maintains the explicit nature of the relationship [14, 21]. Let’s re-examine Figure 2.1, we can see that a *dog* is a type of *animal* but we can say that *animal* is a type of *dog*. Therefore we need to explicitly define another relationship which goes the other way round.

There is no limit to the number of unique relationships between entities as there is always more than one way to describe the relationship between two entities [14]. With a greater number of unique relationships, machines have a larger pool in which they can match against and interpret knowledge [10, 15].

### 2.1.1 Resource Description Framework (RDF)

The recommended implementation standard of semantic data by W3C is the RDF [1]. RDF defines the base vocabulary, serialization format and the resource identification that semantic data should implement in order to allow interoperability between different implementations.

In RDF, semantic data is stored in the form of *subject-predicate-object* known as triples. It provides a way in which the directed graph of semantic data can be represented in a serialized manner [1]. This is important as persistent storage of data as well as transfer of data over the network requires that data be serialized. An example of a directed graph

![Figure 2.1: Entities annotated with semantic relationships.](image)
2.1 Semantic Data

Figure 2.2: Triples - A flat representation of a directed graph.

After conversion to the triple format can be seen in Figure 2.2 where once converted, the data is serialized in the conventional text format. Storage of semantic data is flexible as RDF graphs can be easily generated at runtime [19]. This allows the use of readily available technologies and current data to be used to form RDF data sources.

Once semantic data in transformed into a serializable form, it can then be persisted in the form of a text file. It is also possible to take advantage of existing query and indexing strategies from relational databases by storing the triples in them [10, 2]. To provide better support than relational databases for querying and retrieval of triples, specialized databases known as TripleStores has also been created with performance optimizations targeted specifically for data stored in the form of triples.

2.1.2 Linked Data

On the Web, multiple semantic data sources can be interlinked to create a mesh of semantic knowledge [2, 10, 19, 5]. In order to allow for interlinking between several data sources, RDF specifies that each entity and relationship must have a unique resource identifier (URI) [5, 6, 10]. This is to avoid possible conflicts between different data sources that share similar entities such as a repository of information on people and another with information on countries having the property name.

Auer et. al. recommends having a common serialization format across the different
data sources even though it is not required [2]. The common serialization format will mean that application that consume the data need not be developed with the added complexity of working with different formats. In RDF, two serialization formats are specified, the RDF+XML format and the Notation3 format.

An example of using interlinked data sources is linking articles on cities from Wikipedia with Geonames data set to enrich the data [2]. Each city article could then be displayed with geographic information such as the location of the city on an interactive map build for the Geonames repository.

### 2.2 Ontology

The representation of semantic data has no schema because of its design that allows great flexibility in the way data is maintained [12]. Without a schema, there is no guidance or direction onto which queries can be formulated [12, 13]. In contrast, a relational database uses a schema that partitions data into tables and each table has multiple rows of data. Each row is split into several predefined fields of a specific data-type specified in the schema. A query is formulated based on the schema and looks through the defined fields.
for values specified in the query. The main challenge of developing a schema for semantic
data is that it has to continuously adapt to an ever changing landscape [2]. The schema
also has to support the multitude of ways to describe and classify entities [13].

An ontology is the means of describing an entity and how the entities are related
to each other. Ontologies play their part in semantic data by providing a well defined
vocabulary to allow for users to execute queries [12, 13]. In a way, ontologies can be seen
as a schema for semantic data as they both share many traits and can be classified as a
form of metadata [3, 17, 32].

The key difference between a schema and an ontology is that an ontology is applied over
existing semantic data rather than data being made to conform to a schema [12, 13, 17].
Because ontologies are applied onto semantic data, it is possible and common for a single
corpus of semantic data to have multiple ontologies applied [14, 21].

As ontologies are represented in the same form as semantic data, it shares the same
principles as semantic data [14]. Therefore, it is possible to apply an ontology over an
ontology to increase the number of connections within the directed graph. This in turn
increases the number of dimensions in which data can be queried [14]. As an example,
lets consider a domain specific ontology on a persons biography and a domain specific
ontology used for an address book. They both link to a person entity from which we can
use to find out the biography of the person using the biography ontology and the contact
details of the person using the address book ontology. If we combined both the ontologies
together, it is then possible to directly retrieve a persons biography by querying for the
person based on one of the contact details such as mobile number.

Ontologies used must be consistent across a plethora of communities on the Web to
allow linking between different sources of semantic data [2, 25]. The reuse of ontologies
over different data set can help ensure consistency in the way semantic data is represented.
At present, it is difficult to determine the inconsistencies of currently available semantic
data because of the lack of tools that provide a common visibility point [2].

The ontology that is currently endorsed by W3C is the Web Ontology Language (OWL)
[9, 13, 25]. OWL contains a high number of definitions for entities spanning a wide range
of topics and is partitioned into several namespaces to make it easier to work with. OWL
is made available in several versions with different depths in which entities are described.
This is to enable OWL to be deployed for different requirements trading expressiveness
for computational complexity.

Some examples of other domain specific ontologies are:

**Friend-of-a-Friend (FOAF)** FOAF is a vocabulary for describing people.

**Dublin Core (DC)** DC defines general metadata attributes.
**Semantically-Interlinked Online Communities (SIOC)** SIOC is a vocabulary for representing online communities.

**Simple Knowledge Organization System (SKOS)** SKOS provides terms that represent taxonomies and loosely structure knowledge.

**Creative Commons (CC)** CC is a vocabulary for describing license terms.

According to Schaffert et. al., using smaller domain specific ontology sets is preferable over larger general purpose ones [21]. Smaller domain specific ontologies are easier to maintain as they have a smaller field of focus. Furthermore, it is more efficient to apply only the required ontologies based on the domain rather than having redundant data from applying large general purpose ontologies.

### 2.3 Consuming Semantic Data

In order to support a consistent way in which semantic data can be access, the **SPARQL Protocol and RDF Query Language (SPARQL)** has been created [2]. SPARQL provides a formal way in which semantic data based on RDF is queried. SPARQL will be covered in greater detail in Chapter 4.

Because of the way that RDF works, it is possible to build semantic data as separate components to provide semantic features over existing content platforms such as Wikis [11]. To achieve this, semantic data will need to first be extracted from the content platform and parsed into a form of semantic data such as RDF. Then, a maintenance procedure is needed to keep the semantic data in sync with the originating content platform.

Once the corpus of semantic data has been created, an **Application Programming Interface (API)** is built on top of the data to provide access. An API would serve as an access point that allows developers to easily integrate semantic data into their applications and provide users with a seamless user experience [2]. One example of an API implementation for semantic data is the SPARQL endpoint; where SPARQL queries are used to retrieve entities in the semantic data.

Because semantic data maps closely to the way humans interpret knowledge, it can handle more complex queries [10, 15]. For example, if we wanted to find a town by the shore that is closest to Auckland in New Zealand. This can be achieved in a semantic query by specifying the shortest distance to Auckland as one of the criteria and having a shoreline as the other criteria. It is also possible to execute statistical queries without having an aggregation list [10]. For example, we can get a list of cities with a population greater than two million without manually building a list of cities with their population.

One of the major advantages of semantic data is that it is not limited to textual content [14]. Semantic data can be used to annotate all types of media such as videos.
2.3 Consuming Semantic Data

and audio by just treating the media content as an entity and specifying relationships to it as seen in Figure 2.4. This allows semantic technologies such as SPARQL queries to be used to retrieve the media file.

With the ability to link data over multiple data sources in RDF, relevant information from multiple sources can be consumed by a single application to provide a more complete set of information to the user [14]. Take a retail website for instance, it can retrieve price and availability information from its own repository while leveraging the description details for its products directly from the manufacturers repository.
3

Extraction of Semantic Data from Wikipedia

Wikipedia is a rich source of information which can be greatly enhanced by applying semantic technologies on to its content [4, 5]. According to Ruiz-Casado et. al., manually extracting semantic knowledge over existing data is a very costly and laborious exercise [20]. Therefore, they have recommended the use of an automated extraction method to extract semantic knowledge from Wikipedia.

In order to spur the development of the semantic technologies, a large corpus of data is needed to assist in the development and evaluation of different techniques [2]. By extracting the semantic data from Wikipedia, a large amount of semantic knowledge is made available because of the vast amounts of information stored in Wikipedia’s articles. When users are able to reap the benefits of semantic knowledge, they will be more committed to including semantic information when authoring articles. This would complete the cycle and ensure the healthy growth of semantic technologies.

MediaWiki, which is the Wiki software that Wikipedia runs on has introduced some extensions that provide some form of structured information [3]. Through the use of these extensions, we could extract this readily available structured information and transform them into semantic data. By creating a reusable process in which the structured information is extracted and converted into semantic data, it is possible to preserve the simplicity of authoring articles in Wikipedia while extending its capabilities separately.
The structure of Wikipedia is extremely flexible in order to support the various domains of information that it holds [7, 13]. It is therefore up to the contributors to mould and shape the structure in a way that is appropriate to them for a particular domain. The current state of Wikipedia shows categories and lists to be the two main forms of structuring for articles within Wikipedia [7, 13]. Using categories and lists, it is possible to extract semantic information which we can use to build our corpus of semantic data [7, 23].

WikiText which is the markup syntax used by MediaWiki can also be used to acquire semantic information within each article [10, 13]. The two markups that we will be examining are the Infobox markup which provides a quick summary of important points for an article and the Link markup which allows different articles to be linked together by keywords.

As Wikipedia is designed to cater to human interpretation; it allows for a very flexible structure of information [3, 4]. This leads to inconsistencies within the data such as the use of different measurement units or different wording of similar concepts. Therefore, it is difficult to apply automated information extraction and mining techniques to extract semantic data from Wikipedia.

Depending on the usage of the extracted semantic data, it is possible to tune the tolerance of inconsistencies of extracted data to yield better results [7]. The method of extraction of semantic data from Wikipedia that we use is fairly lenient as users can correct errors in the extracted data later on as part of the authoring process [20]. Our
reason for this is because it is more important for us to have a large working base of semantic data. This would result in semantic queries executed on our corpus of extracted semantic data having a higher possibility of finding a match. Also, correction of errors in the extracted data is less laborious than the recreation of content with semantic data provided the extraction process does not have a high error rate.

In the next few sections, we will talk about the Infobox structure, Category and List structure and the Link structure of Wikipedia. We will examine how the information they provide is relevant and how to extract the semantic information that they present. Then, we will present our extraction process built using an extraction framework followed by a short section on maintaining the consistency between the source; Wikipedia and our corpus of semantic data.

### 3.1 Infobox

Wikipedia makes use of templates to assist authors in producing a well formatted article by providing a base that guides the author when producing content [3, 2]. The most useful template type for generating semantic knowledge used in Wikipedia is the Infobox template [3, 2, 4]. It is a collection of attributes and values associated with an article which summarizes the important points of the article [23, 24, 29, 2, 10, 4].

In Figure 3.2, we can see the key points about Auckland city from the Infobox extracted from the Wikipedia article on Auckland. Information such as where Auckland is located in New Zealand, the country in which it belongs to and its population can be easily found within the Infobox.

The attribute/value pairs found in Infoboxes can be easily translated into semantic data such as a RDF graph where the subject is the article, the predicate is the attribute and the object is the value [10, 13, 3, 4]. Because similar articles contain common attributes, it is possible to abstract a level of information that is common across multiple articles [10, 13, 17]. The abstracted level of information is ideal because it provides a way that the generated semantic data can be queried and compared. Using this abstracted level of information, we can produce an ontology that is specific to the information available in Wikipedia’s Infoboxes.

While extracting Infoboxes from Wikipedia articles may sound ideal, there are some factors which make it difficult to produce high quality semantic data [3, 4]. These factors are mainly to do with the inconsistency of information provided within Infoboxes [29, 5, 10]. This is because there are very little restrictions on how the Infobox templates are used.

As contributors to Wikipedia place importance on different points within an article the attributes presented within the Infoboxes vary from article to article [2]. It is also
possible that the contributors make use of different terms to represent the same concept for an attribute. As can be seen in Figure 3.3, the Infobox on the ship: “TSS Earnslaw” and the automobile: “Subaru Impreza” place importance on a common concept of who made them. However, the Infobox on the ship uses the attribute “Builder” while the Infobox for the automobile uses the attribute “Manufacturer”.

Both the Infoboxes in Figure 3.3 disseminate information with very different emphasis. The Infobox for the ship includes dimension information while the article on the “Subaru Impreza” places dimension information in a separate Infobox seen in Figure 3.4.

Aside from the different attributes used across different articles; there is also the problem of inconsistent value format used to present the information [13, 10]. For instance,
someone from the United States might use the “mm/dd/yyyy” date format display a date while someone from the United Kingdom might use the “dd/mm/yyyy” date format. Someone reading the article would not be able to tell which date format has been used and neither will a regular expression used to extract the date.

Authors may sometimes choose to use tables instead of Infoboxes to present a summary of their content [3]. This hinders the discovery of semantic knowledge because of the different syntax used; which hides it from parsers. Anomalies in the values such as the use of the metric scale in some articles and the use of the imperial scale in other articles for the same attribute causes problems as they cannot provide an accurate comparisons [3]. While it is possible to extract the measurement units together with the value to reconcile this, the problem lies with the measurement units not having a standard placement as well as being stated in different formats. Figure 3.4 demonstrates an example where similar attributes have different representation of the values.

The flexibility of Infoboxes which allow multiple values to be grouped together in a
single attribute also limits the ability of extracting useful semantic information [3]. This is because parsers have no way of identifying logical break points within the values in order to separate them. As seen in Figure 3.5, the URL attribute is assigned two Web addresses and a generic “Official mirrors list” which is a link to a page with a full list of official mirror sites.

To gain some consistency between similar Infoboxes across different articles, typed Infoboxes are used [29, 13, 7, 23]. An example would be the automobile typed Infobox template which is shown in Snippet 3.1. Using the type name of these typed Infobox templates, we placed a classification for each article [22, 23].

There are however, articles that contain multiple Infoboxes with different type names.
### 3.1 Infobox

#### Snippet 3.1:
Sample WikiText of an automobile typed Infobox.

```wikitext
{{Infobox Automobile
|image = [[File:2007 Subaru Impreza Front Quarter.jpg
   |250px|2007 Subaru Impreza RS hatchback (Australia)]]
|name = Subaru Impreza
|manufacturer = [[Subaru]]
|parent_company = [[Fuji Heavy Industries]]
|production = 1993–present
|assembly = [[ta, Gunma|ta]], [[Gunma]],
   [[Japan]] (1993–present) <br/>
   [[Lafayette, Indiana]], [[USA]] (2003–present)
|class = [[Compact car|Compact]]
|predecessor = [[Subaru Leone|Subaru Leone/Loyale]]
}}
```

Figure 3.5: Infobox where multiple values are assigned to an attribute.
In the construction of our semantic data, we could potentially take the first Infobox from each article and assign its type name as the classification for the article. Although Wikipedia has tidied up their list of typed Infobox templates, there are some legacy ones still in use which have random names such as “Automobile1234”. Here, we have two choices: preserve them in the extracted semantic data or filter them out based on the correct list of typed Infobox templates.

Then, there is also the issue of defined parameters within Infobox templates being optional. Missing information within the Infoboxes would cause users to miss some articles if they were relying on navigation and search capabilities based on the generated semantic data. This then falls back to the users to locate the article through more conventional methods such as keyword search engines or manual site crawling based on the existing Wikipedia navigation. Once users are able to locate the article, they could perform incremental edits on the article to provide further information in the Infobox; which will then be carried through the next time the extract is performed.

Using the classification of articles, we could view a collection of articles as specialized sections within Wikipedia. This provides a topic based segmentation much like the Wikis on Wikia.com. While it may seem counter-intuitive to split the articles in Wikipedia into different sections and have users pick a section in which to view; the users interaction with Wikipedia has not actually changed. They are still able to access Wikipedia as a whole, but they now have the added possibility of providing context to what they are looking for making their search more refined.

3.2 Categories and Lists

Categories are a special construct within the WikiText syntax that allows the editor to group several articles which share a common topic. The WikiMedia software uses these constructs to automatically generate category pages for users to quickly identify articles related to a topic. Categories are used to assist in exploratory browsing of Wikipedia.

An example of categories used in Wikipedia is that the article on Mitsubishi Motors and the article on Subaru are marked with the category Car Manufacturers. A page named Car Manufacturers is automatically generated with links to both of the articles. As more articles about other car manufacturers are added to the category, the page is automatically updated to include those articles. Users who are looking for information on car manufacturers in general can visit the Car Manufacturers page to view a list of relevant articles.

As categories may have sub-categories to further classify articles, it presents a hierarchical structure to the way articles are organized in Wikipedia. Snippet 3.2,
3.3 Links

Links in Wikipedia are basically hyperlinks that take users from article to article or to external resources. They are an important aspect of Wikipedia as they allow users to quickly reach more information on which they pertain to. Take for example, the snippet

```
[[Category:Auckland]]
[[Category:Populated places established in 1840]]
[[Category:Port cities in New Zealand]]
[[Category:Populated coastal places in New Zealand]]
[[Category:Host cities of the Commonwealth Games]]
[[Category:Isthmuses]]
[[Category:Former national capitals of New Zealand]]
```

Snippet 3.2: WikiText used to define categories within an article.

which is *WikiText* taken directly from the article on Auckland shows that an article may belong to multiple categories as well. With articles belonging to multiple categories, the author of the article can group their article with different topics. This allows users to easily navigate between articles based on the topic that they want to know about.

The flexibility of the category system enables the author to be free to create any hierarchical structure which they deem appropriate. While this may seem like a powerful feature, it is actually difficult for the author to identify existing categories that match their article. Because of that, there are duplicate topics that are termed differently. Typographical errors made when specifying categories would also mean that some articles are hidden from the user if they choose to use categories for their navigation.

As seen in Figure 3.6, categories arrange the articles into the form of a hierarchical tree. If the link between Category 1:A and Category 2:B were to be broken, users would no longer be able to navigate from Article C to either Article D or Article E using the category system.

The list system is similar to the category system in terms of being able to provide navigational functionality based on a topic [7]. One of the differences between the category system and the list system is that lists can be embedded into the article itself whereas categories are always shown at the bottom of the article.

Another difference is that the structure of the list system is flat where a list always links to an article and not another list. It is, however possible to have a list that links to an article that only contains another list. The third difference is that lists are maintained manually by the author of the article, which gives them the flexibility of including more information such as a quick summary for items in the list.

3.3 Links
Leisure

Auckland is popularly known as the "City of Sails" because the harbour is often dotted with hundreds of yachts and has more per capita than any other city in the world, with around 135,000 yachts and launches. Around 60,500 of the country's 149,900 registered yachtsmen come from the Auckland Region.[35] About one in three Auckland households owns a boat.[36]

on Auckland’s leisure aspect in Figure 3.7; a user may be reading the article and comes across the word *yacht*. Lets assume the user is unfamiliar with what a *yacht* is. All the user has to do is click on the word *yacht* which will take the user to the article that gives an in-depth explanation of what a *yacht* is.

**Snippet 3.3:** Current method of specifying a link using WikiText syntax.

While the link itself does not prove to be useful semantic information; adding type information for the link as suggested by Völkel et. al. makes the link a semantic relationship [24, 7]. Völkel et. al. proposed enhancing the current WikiText syntax to allow
3.3 Links

London is the capital city of [[is capital of::United Kingdom]]

Snippet 3.4: Method of specifying a typed link using an extension to the WikiText syntax.

for type information to be added [24]. Snippet 3.3 shows a snippet of WikiText with “United Kingdom” specified as a link to the article for United Kingdom while Snippet 3.4 demonstrates the new method of specifying a link with type information.

![Figure 3.8: Typed links as described by Vökel et al.[24].](image)

Typed links is a good idea for articles moving forward, but then there is the problem of the author needing to know the list of possible types for each link. This takes away the essence of what a Wiki is; which is to be quick and simple. There is also the problem for existing articles; that it would not provide the same degree of semantic information until the authors revisit each article and append the appropriate types for each link.

There are other possible ways of extracting semantic information from links as well without the use of typed links. We considered the possibility of automatically generating type information for the links based on the content of the article. The problem is that the relationships that is extracted needs to be classified whether they represent a strong semantic relationship or are just navigational links [7, 16].

Milne describes a method of calculating semantic relatedness by inspecting the terms used to represent the links in the articles using what he calls the Wikipedia Link Vector Model [16]. Using the model, terms are placed into context by the way articles are linked.
to each other. Take the ambiguous term *blade* for instance, which can mean a sharp instrument for cutting or a comic series.

By examining the links between the articles for both, the article for *blade* meaning sharp instrument has a high number of links to other cutting instruments such as scissors. The other article for *blade* meaning a comic series would have a high number of links to other comics. From there, we only need to classify a sample set of articles and the *Wikipedia Link Vector Model* can be used to apply that classification over the rest of the articles.

Strube and Ponzetto have proposed a computable approach to generating semantic relations between articles by blending category information as well as the disambiguation redirects in order to generate semantic relations between the articles. [22]. The disambiguation redirects are the set of pages that provide a list of articles that share a common name but represents different things in different context. Take *Mercury* for instance, it can represent the chemical element, a planet or a god from Greek mythology.

By traversing these links and applying several functions to the contents of the articles, Strube and Ponzetto are able to generate type information for the links [22]. For example, consider the pieces in the *Chess* board game. Because of the contents and links between the articles of different pieces like *King*, *Queen*, *Bishop* and *Rook* with the article on *Chess*, they are able to generate the relation *has piece* from *Chess* to each of the pieces.

In order to classify the extracted typed links based on how strong a semantic relationship it represents, there are two measures proposed by Chernov et. al. [7]. The first is to count the number of links that exist between two articles for each extracted typed link, where the greater link count represents a stronger relationship. The second method is that for a particular typed link, the ratio of links between the two articles compared to the total number of links in the two articles represents the strength of the semantic relationship.

We consider the second method a better measure of semantic relationship strength as it is unaffected by the difference in the number of links between articles. To answer the question of why quantifying semantic relationship strength is useful; the measurement allows us to identify navigational links from semantic link as well as provide ranking capability in semantic searches. This works much like Google’s PageRank where search results are ordered based on the number of links to each result. In semantic queries, the results can be sorted based on the relationship strength.

There are also redirects in Wikipedia which handle the multitude of ways in which an article can be identified. Isbell and Butler states that having such redirects is problematic because multiple identifiers are used for a single resource while semantic data principles require entities to be unique [13]. They suggest the use of the relationship *owl:sameAs* in order to link the different terms giving the same meaning to make use of these redirects.
as semantic information. We think that this approach is appropriate for handling the redirects as when semantic data is used in querying, it presents a larger vocabulary to match against, which in turn provides more relevant results.

### 3.4 Extraction Framework

![Extraction Framework](image)

To assist in the extraction process of semantic data from Wikipedia, Bizer et al. from the DBPedia project describes an extraction framework as seen in Figure 3.9, which they use [5]. The extraction framework consists of three stages in which data from Wikipedia is converted into semantic data. The three stages are the extraction stage, the parsing stage and the ontology matching stage. We take a similar approach as we think that this is a good process for dealing with the task of extracting semantic data from Wikipedia.

PDStore is a *TripleStore* type database used for storing semantic data in the form of a connected graph; where instances are tied together by relationships. As we are part of the working group for PDStore, we make use of this opportunity to test the capabilities of PDStore with a large set of data. For that purpose, we make use of *PDStore* to store the large corpus of semantic data that we extract from Wikipedia.
**PDStore** provides some interesting features that is useful for the purpose of storing semantic data extracted from Wikipedia. One of the features of **PDStore** is that it has the ability to provide versioning over its data. Wikipedia has versioning system built into its authoring process and we want to be able to carry the same functionality across from Wikipedia to the extracted semantic data. Another interesting feature is **PDStore’s** ability to synchronize its data between multiple instances. As Wikipedia is accessed by high amounts of traffic from around the world, it will need the ability scale out to multiple instances in order to avoid bottlenecks; much like content deliver networks (CDN).

### 3.4.1 Extraction

Extraction is the stage where raw data from Wikipedia in the form of *WikiText* is scanned through and the appropriate sections of the data such as *Infoboxes*, categories and links are identified and extracted [5]. To acquire the raw data from Wikipedia, we had three options: scrape the content over the Web, make use of the MediaWiki API or download a dump of the raw data.

Scraping the content over the Web was not considered plausible as information is lost when *WikiText* is converted into HTML [13]. We considered using the MediaWiki API to
retrieve the latest copy of each article but the requirements to be met to carry out such a procedure meant that we needed to hit the Wikipedia site with substantial amount of traffic in order to crawl through and download all the articles. Also, Wikipedia’s policy prohibits any form of automated crawling because it consumes a high amount of their resources.

That leaves us with the raw data dump that Wikipedia makes available whenever it successfully completes the dumping process. The raw data is in the Extensible Markup Language (XML) format; a format in which we can use an XML parser to extract each individual article. For our extraction, we used the English language Wikipedia data dump with no history information, dated 30 January 2010. Its size is 5.6 gigabytes when compressed and expands to 24.9 gigabytes.

Because of the size of the raw data file, we needed an XML parser that could read the contents of the file from disk rather than load the XML tree into memory for processing. As we do not need to navigate back and forth in the XML tree, we used a XML stream reader which reads the tree in the order it is stored in the file.

```latex
{{Infobox template
|attribute1 = value1
|attribute2 = {{value2}}
|attribute3 = value3
}}
```

Snippet 3.5: Snippet of a clean Infobox extract.

```latex
{{Infobox template
|attribute1 = value1
|attribute2 = {{value2}}
}}
```

Snippet 3.6: Snippet of an extracted Infobox which terminated too early.

The WikiText syntax makes heavy use of brackets to denote the sections of the data that we want to extract. Because it allows recursive use of brackets, we could not use regular expressions to extract the data that we want as it has no knowledge of the recursion level and would terminate too early for some of the sections. An example of this can be seen by comparing Snippet 3.5 and Snippet 3.6.

As stated by Isbell and Butler, an abstract syntax tree (AST) type parser would best suit the extraction of the different sections specified by the WikiText syntax [13]. For our purposes, we only require certain sections such as the Infoboxes and categories from each article, therefore we consider building an AST for that purpose to be excessive.
Instead, we devised a simple bracket balancing mechanism to effectively extract the different sections that we want from the rest of the article content. Our mechanism works by keeping a tally on the number of brackets found. Each time the pattern of an opening bracket is found, we increase the tally by one and each time a matching closing bracket pattern is found, we reduce the tally by one. We identify a section where the beginning is when the tally is increased from zero and the end is when the tally is zero again.

3.4.2 Parsing

The sections extracted from the Wikipedia data dump is in the form of plain text. Parsers are then used to apply manipulation rules on the extracted data in order to make it consistent and meaningful [5, 10]. For example, a parser is used to convert the datatype of numerical data so that arithmetical operations can be performed on it.

Most WikiText parsers operate by converting raw data directly into HTML for rendering, therefore, we created a special purpose parser for our purposes [13]. When building our parser, we focused on the Infobox structure because it contains the greatest amount of semantic information that we can extract. We used regular expressions in our parser to separate the attributes and values for each Infobox. Because PDStore stores data in a strongly typed manner, we incorporated a function in our parser to test and convert each of the extracted values to the appropriate data type.

Using the category information that we obtained from the WikiText, we make use of the owl:class relationship to link them together with the relevant article entities in the extracted semantic data [9]. As for the links in the WikiText, we did not include them in our project as it required substantial development of complex parsing as well as NLP algorithms, which is outside the scope of our project.

3.4.3 Ontology Matching

After the semantic data is extracted from the articles, it needs to be mapped onto one or many ontologies; which is the most laborious part of extracting semantic information from Wikipedia. [5, 10]. This is because of the need to manually create rules that will match the entities in the extracted semantic data to appropriate entities in the ontology. The set of rules that match the semantic data to the ontology has to be continuously maintained over time if this process were to be reused.

Mapping the extracted semantic data to ontologies is needed in order to make the semantic data accessible and useful [4, 25]. Bhole et. al. makes use of a method where a sample set is created by manually analyzing several random articles and applying heuristic categorization to link entities with an ontology [4, 23]. When applying heuristic categorization, they also wrote rules on how the entity is matched to the ontology. This gives
the most accurate match between the semantic data and assuming the extracted semantic data is consistent across multiple articles, it will produce the highest number of matches as well. From there, it is possible to make use of machine learning techniques to automatically match the remaining entities with the ontology [27, 28].

We made use of the same technique but on a much smaller scale just as a proof that similar techniques can be applied to our extracted semantic data. The articles which we used to define our rules were the “Subaru Impreza” article, “Mitsubishi Lancer” article and the article on “Auckland”. The rules successfully linked entities in the other articles to our sample set. They are the articles on “Berlin”, “Tokyo”, “New York”, “London”, “Lamborghini Gallardo”, “Enzo Ferrari (Automobile)” and “Ford Mustang”.

There have also been attempts to create a computable approach by determining semantic relatedness in order to automatically match ontologies onto data [16]. According to Milne, there are two main approaches in calculating the semantic relationship between entities [16].

The first approach is to perform a statistical analysis on a large corpus of data to build an experience context. The experience context is then used to determine which entities should be linked as well as which relationship to apply. Using this method generally does not provide very accurate results. The second approach is to make use of lexical structures from references such as WordNet to determine the relationship each entity has with each other. This approach has a limited scope and scale to which it can be applied to.

3.5 Synchronizing with Wikipedia

Once we have our corpus of semantic data, it is important to keep the contents continuously synchronized with Wikipedia. This ensures that the corpus of semantic data does not go out of date and stays relevant for the semantic technologies that consume it.

One possible solution to keep the data in sync with the content from Wikipedia which was proposed by Isbell and Butler is to monitor changes to Wikipedia over a Really Simple Syndication (RSS) feed [13]. Everytime a change is detected in Wikipedia, the changes are fed through to the extracted corpus of semantic data as well. This would provide the latest information in the corpus of semantic data, but it would lose track of changes between the time the data dump is generated from Wikipedia and the time the RSS is connected with the extracted semantic data.
3.6 Evaluation of the Semantic Data Extraction Process

We used a 2.13 GHz Intel Quad Core processor with 2 GB of RAM in the evaluation of our extraction program. Our program was written in Java running on version 1.6.0 of the runtime environment and we used the Simple API for XML (SAX) library to parse the XML data. In total, there are 9,541,307 articles spread across 24.9 GB of data. The first run took 38 hours to complete but we noticed that only one of the cores was used and there was a high memory usage causing plenty of page faults.

A modification to the code was made so that each article would be processed in separate thread and the SAX library was swapped for the Streaming API for XML (StAX). With those two changes, we saw massive improvement where the total time taken for the extract was reduced to 4 hours.

The file size of the extracted semantic data was 2.6 GB, which is relatively small when compared to the original data from Wikipedia. Upon querying the extracted semantic data, we found that PDStore could not handle that amount of data. The system would run out of memory and the application would crash after 4 hours of running the query.

We are not alone on this as most semantic data repositories currently available only work well with small data sets and do not scale very well with larger ones [8, 10]. An indexing strategy is needed in order to speed up the lookup process as scanning through the whole data set for each query takes too long and also draw heavily on available resources [8]. Similar properties in different ontology sets should also be mapped together to form a consolidated ontology to coerce faster results.

In order to support semantic querying on a larger scale such as the extracted semantic data from Wikipedia, a distributed effort could be used to spread the task into smaller parts and execute them separately to generate the results in an acceptable time frame. Pre-emptively loading the appropriate portions of the ontology for each query like what has been done by Fernandez et. al. in PowerMap would also help reduce the time needed to execute the query [8].

Another problem that arose during the course of our project was the development of a SPARQL interface for PDStore which we were going to make use of in our project, was delayed. In the interest of completing the other parts of our project within the timeframe available, we replaced our extracted semantic data with DBPedia’s API instead of reducing the semantic data we had generated to a more manageable size. We would return to correct these problems and re-evaluate the extraction process in the future.
4

Querying Semantic Data

The main benefit that semantic data provides to its users is that it allows greater search capability by providing the means of executing complex queries [11]. Take synonyms in the English language for example; if we were to consider the word “dash” and “sprint” in the context of running where they have similar meaning. Some users might use the word “dash” and some users might use the word “sprint” in their search phrase which would return very different results using keyword search. This is because the exact word is used to match the results and if the word does not exist in its index, the keyword search engine has no other means to locate it.

Using semantic queries, it is possible to determine that “dash” and “sprint” may have the same meaning by examining the relationship between the two words [11]. In this chapter, we will examine the differences between semantic queries compared to keyword search, how semantic queries work and the way in which semantic queries are expressed.

4.1 Semantic Querying compared to Keyword Search

The way that keyword search works is by matching the words in the search phrase specified against textual content within Web pages to find exact matches. When a match is found on the page, the Web page is added to the list of results which is then displayed to the user when the search completes. Keyword search makes no attempt to interpret the search phrase that the user specifies [7, 10, 31].
The reason that keyword search is such an indispensable tool is that in the early days of the Web, the content on the Web is mostly made out of text. Even now, most of the content on the Web is textual in nature, although there is an increasing number of other media formats such as video and audio.

The way that semantic queries work is that it queries against the underlying concepts based on what is specified in the query [12, 16]. This allows users to ask complex queries such as “name all the cities with a population greater than 5 million” or “list the number of skyscrapers built in the last 2 years”. Instead of just specifying keywords to be matched against, semantic queries are used to describe the data that is to be retrieved.

Keyword searches have the advantage of being easier to formulate as the user can just enter any word they wish to match. In most cases, users would specify a search phrase that contains some commonly found words. This would cause the results produced to match many pages, most of which are not relevant to what the user is looking for. In some occasions, it is also possible that the keyword being searched for is not present on the Web page that the user is looking for because a different word is used within the content [16].

On the other hand, semantic queries requires users to analyze their search phrase and break them down into simple parts so that it can be expressed in a way that can be
interpreted by the semantic query engine [2]. It does however yield more accurate results when compared to keyword search, thus reducing the amount of work that the user has to do to manually scan through the results and filter out the irrelevant ones.

As a comparison, let’s consider searching for *countries with tropical climate*. Assuming that the keyword search engine has a list of common words to exclude, the terms “countries”, “tropical” and “climate” would be used for matching while the word “with” is excluded from the search. Many modern keyword search engines such as Google, Yahoo and Bing make use of a similar word index to provide some form of basic semantic capability. This would then include words like “country”, “temperature”, “condition” and “humid” to its list of words to match.

Depending on the ordering mechanism employed, most search engines would place the results where the keywords are matched together as a phrase at the top to the least number of keywords matched at the end of the results list. Some examples of irrelevant Web pages that would be included are pages on climate change, tropical diseases, countries around the world and financial climate.

Using semantic queries, a variable is specified as the entity to search for. The variable entity is then described as being of the class *country* and having a *tropical climate*. Figure 4.2 demonstrates how the results are matched. *Bahamas* and *Madagascar* are
matched because they both are classified as countries and match the condition of having a tropical climate. Although Hawaii and Kuala Lumpur has a tropical climate, they are not countries, therefore they are not matched.

Another advantage of semantic queries is that it can be applied to other formats other than text. By annotating other formats such as video data and audio data with semantic information that describes them, semantic queries are able to match against the semantic information and retrieve the data [9, 13]. A similar strategy can be achieved with keyword search by tagging the data with relevant keywords to match against. However, semantic queries are able to perform comparative matches such as finding video data which is greater than 30MB while the same cannot be done with keyword searches.

The Google keyword search engine uses its proprietary weighting system, PageRank. It provides the most relevant results to the user first where the results are order based on the number of links that link to a particular result from highest number of links to lowest [8].

Semantic searches could also benefit from a weighting system to help order the results retrieved [8, 17]. One way of providing such a feature is to use a ratio on how closely related a property is to an entity in the semantic data. However, experiments by Fernandez et. al. shows that current semantic technologies gain only a slight improvement using the weighted results because the ratios were approximately the same for similar matches [8].

The use of an intermediary layer which applies NLP techniques to translate keyword phrases into a semantic query has been proposed by Lei et. al. as a way to eliminate the steep learning curve associated with formal query languages used for semantic queries [15]. This would maintain the current user experience of searching on the Web with the exception of attaining more relevant search results [26, 32]. More advanced search engines such as Google and Bing have started to use this technique in conjunction with their weighting algorithms to provide users with the most accurate and relevant results [26, 32].

4.2 Semantic Queries

In order to make semantic data useful to its users, there needs to be an interface in which the user can interact with the semantic data [5, 26]. At present, the official recommendation from W3C is the formal query language named SPARQL. With the recommendation on how semantic data should be queried, interface applications can be created based on a consistent access method [8].

In this section, we will have a look at how semantic queries work with reference to semantic data that has been extracted from Wikipedia. One of the biggest challenges with creating a body of semantic knowledge is that in order for it to be useful, it requires
a sizable amount of semantic data [10]. The greater amount of semantic data that we have, the greater the chances of finding matches to answer our queries. Since Wikipedia contains a huge amount of knowledge in its articles, it makes a good candidate as a large corpus of semantic data.

To be able to access the large amount of semantic data from Wikipedia, there needs to be a schema that describes the data [3]. The schema will provide a consistent access point from which users can build their semantic queries as seen in Figure 4.3. Because Wikipedia’s structure changes over time to suit the needs of its contributors, the schema must be able to adapt to these changes. In order to accommodate the flexibility of Wikipedia, an ontology is used to describe the semantic data that has been extracted from it [2, 10].

As there are many ways to represent the same concept or idea in natural language, a flexible method of storing semantic relationships is needed [12]. In RDF, creating multiple relationships representing the same link between two entities is possible by repeating the subject and the object while replacing the predicate with different relationships. This allows for easy expansion of the underlying semantic vocabulary, thus increasing the ability to interpret a user’s query.

Linking two entities using multiple relationships is not the recommended approach
for specifying the different relationships that can be used. Instead, the recommended method is to link the different relationships together using the relationships `sameAs` in both directions which is represented by a dotted-line in Figure 4.4. Using this method allows the possibility of creating performance enhancements on the underlying storage by reducing the number of indexes to scan through. It also eliminates redundant links and streamlines the mapping process as future entities using the relationship need only specify one of the relationships from the group.

Using semantic technologies, it is also possible to provide the ability to query statistical information on-the-fly in any dimension without the need to reshape the data using methods such as table pivoting or an aggregate table. [2]. An example of a statistical query that users might use on semantic data from Wikipedia is determining the number of games available for the Amiga console.

### 4.3 Semantic Query Language

Users tend to distance themselves from formal query languages due to their steep learning curve [15]. However, the use of formal query languages such as SPARQL, Sesame RDF Query Language (SeRQL) and Notation 3 Query Language (N3QL) is needed in order to use the full capabilities of semantic queries [9]. Before learning how to compose queries using the formal query languages, users need to be aware of the principles behind semantic data such as how ontologies are used to provide structure in semantic data. Then, they
need to learn the syntax of the formal query that they want to use in order to query the semantic data effectively.

From our talks with colleagues and friends, we found that most of them are not comfortable with writing semantic queries as they think it is complicated and takes too long to learn. They would rather make use of easier methods of searching such as keyword search at the expense of not being able to perform complex queries. Using a graphical user interface (GUI) to hide the complexities of constructing the query would simplify the process [15].

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?manufacturer ?name ?car
WHERE {
  ?car <http://purl.org/dc/terms/subject>
  ?car foaf:name ?name .
  ?man foaf:name ?manufacturer
}
```

Snippet 4.1: Example SPARQL query for looking up information on luxury cars.

In our project, we made use of SPARQL to tap into the semantic knowledge provided by DBPedia. Snippet 4.1 shows an example of a SPARQL query used to retrieve a list of luxury cars, displaying the name of the manufacturer, the name of the car and a link to the resource for the car in the results. Let us break the query down and examine the different parts.

The top most part of the query where the lines begin with “PREFIX” is the section where the different namespaces used are declared. This is the list of ontologies that we would like to query against in order to retrieve data. The next section is the line beginning with “SELECT” which is the line that specifies what information we would like to retrieve. They are basically variables where information is placed into them and outputted.
The “WHERE” clause is the next section which is the heart of the query. It specifies what is needed to be matched in order to assign the variables. The first line in Snippet 4.2 states that the variable “car” should be matched to the “subject” predicate which is linked to the “Luxury_vehicles” category. The next line states that the “car” object should follow the “name” predicate to populate the “name” variable. The line after that states that from the “car” object, it should follow the “manufacturer” predicate in order to populate the variable “man”. Because “man” variable is a manufacturer object rather than a name, another line is included to populate the last variable “manufacturer” with the name of the manufacturer.

SPARQL queries which span over multiple repositories operate by extracting the relevant data based on the URIs specified in the query and merge them together in a temporal space before performing the actual query [19, 26]. According to Quilitz and Leser, this could pose a potential problem with copyright issues as well as not being feasible to trans-
fer substantial amounts of data over the network [19]. They propose that executing queries over multiple repositories be decomposed into multiple smaller queries and executed on the individual repositories where the data is stored.

To achieve this, they introduced an intermediary querying engine which operates as a layer in between the user and the repositories. The original query is analyzed and decomposed based on the service descriptions that it maintains. Each smaller query is then executed separately on each repository and the results are then merged back together in the intermediary query engine before returning the results to the user.
User Interfaces for Querying Semantic Data

Using a large ontology in semantic queries can be daunting to a user as it is very difficult to identify appropriate properties and relationships to use in the query [3]. Users have to be familiar with the underlying ontology before they are able to formulate queries which will return results [26]. By incorrectly specifying the properties or the relationship, the user often ends up with a zero result set, which is not ideal.

Therefore, a mechanism is needed in order to assist users in making use of the semantic data and hide the complexity involved in accessing it [3, 5]. This can be achieved in a similar fashion to how it is done with relational databases. With relational databases, applications are used as access points for end users to view and manipulate the data stored in relational databases.

When surfing the Web, users generally take two main approaches to locate content that they are looking for, browsing and keyword searching [12]. Browsing is basically navigating the Web by following through on a series of links. Users generally start at the main page of a Website that covers the areas that they are interested with. An example of such a sites that could probably be generalized would be news sites such as The Herald\(^1\), sites selling items such as eBay\(^2\) and video sites like YouTube\(^3\).

\(^1\)http://www.nzherald.co.nz  
\(^2\)http://www.ebay.com  
\(^3\)http://www.youtube.com
From the main page, the user would click on links such as the category they are interested in and browse deeper into the site until they reach some content of interest. This method places users into a well-defined context where the content that they browse through would have some degree of relevancy to what they are looking for. This method best suit the scenario where a user is looking for something but is not too sure what it is.

Keyword search on the other hand is a very different approach to locating content over the Web. To use keyword search, the user has to have a word or some words in mind which the content they are looking for would contain. For example, if the user was looking for an article on *how potatoes are grown*, they need only specify that as their search phrase and a list of results pertaining to growing potatoes would be shown. However, there will be some results which do not relate to the phrase which are mixed into the results as well. Some examples are articles on *cooking potatoes* or articles on *growing up*.

One of the reasons why keyword searching is so popular among Web users is because of its simplicity and effectiveness [2]. Keyword search engines are able to retrieve relevant results for what the user is looking for without much effort from the user. There are exceptions to this where the search results are skewed toward more popular use of a term. Consider the word *transformer*, if the user were looking for the electrical component called a transformer, they would have to scan through to find relevant results as most of the results would pertain to the “Transformers” movie instead. This is because keyword engines such as Google rank their results to the most popular use of the word, which aids the majority of users in finding relevant results.

End users are often discouraged from using the powerful searching capability provided by semantic queries because of the complexity of writing the query [13, 15]. In order to encourage users to embrace the use of semantic queries with open arms, the process of constructing a semantic query needs to be simplified to a point where it is as easy to use as keyword search. In this chapter, we will examine the different user interfaces that have been developed over time with the intention of making semantic querying easier to use. The user interfaces are split into two distinct approaches, *graph pattern building* and *faceted browsing*.

### 5.1 Enhancing Wikipedia’s Search Functionality

Using semantic data, we can perform complex queries against Wikipedia’s content, such as finding the *models of vehicles manufactured between 1950 and 1980* or *animals which are mammals* [10]. We will be focusing on ways in which we can provide these kinds of searching capability to users based on the semantic data that is extracted from Wikipedia.

With the availability of semantic data for Wikipedia, queries that extract and compile information that spans over multiple articles can also be performed without laborious
One such example is searching for the names of the co-stars from the “James Bond” movie series. If done manually, a user would have to go to the article on the “James Bond” movie series to get a list of each episode in the series. The user would then have to browse through all the articles in the list and compile the names of all the co-stars.

Using a semantic query, the co-stars are just specified as a variable. The relationships which link the co-star to the movie and then to the series is then specified and the name property of the series is specified as “James Bond”. The semantic query engine would then do the rest of the work to determine if the actor is a co-star in the “James Bond” series and display a list for the user.

Lei et. al. expresses four key points that should be observed when developing interfaces to provide access to the power of semantic queries [15]. The four points are:

**Gradual learning curve.** The user interface needs to be easy to learn as not to scare users away.

**Expressiveness.** The interface should not restrict the user to a limited set of functionality where possible.

**Intuitive presentation of results.** Users should be able to identify and understand the results without the need for external help.

**Low response time.** Be able to calculate and present the results in a short amount of time.

Vökel et. al. shows support for these points as they place importance on similar points [24]. To reduce the learning curve for the end user, the technical details of semantic data such as ontologies and RDF triples could be hidden from the user [21, 26, 32]. While this seems counter intuitive, hiding the technical details actually reduces the complexity of the interface creating a more welcoming feel to the user experience.

### 5.2 Graph Pattern Builder

The graph pattern builder is a means of querying semantic data by describing the graph pattern that we want to look for [3]. The way it works is by describing the concept of querying semantic data in a way that allows the user to visualize its operation [17]. This is easily achieved because semantic data is conceptually represented as directed graph and querying is done by matching a pattern on the graph. From a technical perspective, there is no difference in specifying a semantic query using SPARQL from using the graph pattern builder.
Figure 5.1: SPARQL Explorer - An interface to execute SPARQL queries.

Figure 5.2: Browsing semantic data using Disco.
The first iteration of user interface application that help users construct semantic queries were basically just text boxes where users wrote SPARQL queries with some namespace mapping defined and shown to the user [5]. There is also an output area where the results are display and some even include a possibility for users to choose the output format. The SPARQLExplorer shown in Figure 5.1 is one of the applications that provide such features.

The downside of such an interface is that the end users would still have a hard time coming to grips with writing syntactically correct queries to be executed [8, 3]. Besides syntax issues, users would often find an empty result set because their query defines non-existent relationships properties to look for. At this point, the usefulness of semantic data is still limited to users who have a good working knowledge of semantic technologies.

There was another kind of application for semantic data that was created around the same time, which is Disco. Disco is a semantic data browser which does not provide any form of searching functionality but rather, it provides a way in which users can browse over the semantic data stored on a repository to discover and learn about its contents. This can be used to assist users in determining the different relationships and properties stored for an entity and help in the construction of semantic queries [3].
In order to bring the advantages of semantic queries to the end users, there needs to be a way in which well-formed queries can be generated with minimal effort from the user [5]. To simplify the process of creating a SPARQL query, a step-by-step approach such as query builders are used [3, 5]. As seen in Figure 5.3, users specify their query by defining the pattern of triples that they are looking for. An identifier starting with a “?” is used to indicate a variable that the user is looking for and users are able to select the relationship from a drop-down list. A count of possible matching values is displayed together in the drop-down list for users to get a feel of the number of possible matches and avoid the problem of specifying queries that return no results. The result is displayed based on the variable fields with links to the resource.

![Figure 5.3: Relationship finder used to find how Fiat, Ferrari and Maserati are linked together.](image)

The creation of an application like the relationship finder as seen in Figure 5.4 is more of a tool to assist users in understanding how the entities in a corpus of semantic data is linked together. As it displays a graph allowing the user to visualize how the entities specified are linked, it provides a way to find possible properties and relationships to use when constructing a semantic query [5].

There is another query builder interface named iSPARQL, which is similar to the relationship finder. Semantic queries in iSPARQL are specified by drawing a graph of the pattern that you want to match. This is done by loading ontologies into a toolbox and then dragging and dropping the appropriate items from the toolbox onto the canvas. iSPARQL will then query the underlying semantic data to find the pattern and the results is displayed in a separate tab on the page.
5.3 Faceted Browsing

The evolution of query builders lead to the development of faceted browsers. The idea behind faceted browsing is that users specify filters to reduce the amount of results until they find what they are looking for [9, 10]. Faceted browsing makes it easier to construct queries as semantic queries is easily represented as a set of multiple filters. For example, if we are searching for planes build by Boeing, we would create a filter for entities of the class plane, a filter for the manufacturer Boeing and a filter for the relationship built by.

Although the user still has to decompose their query into different filters, it is much easier than working with earlier query building applications where users had to manually find the appropriate properties and relationships to use. Faceted browsing works on the principles of focusing on a fixed taxonomy structure, which can be referred to as templates of semantic queries [3, 13, 26].

In faceted browsing, taxonomic structures help guide the user by restricting the scope of their query and placing a context onto their search pattern [3, 12]. From the corpus of semantic data extracted from Wikipedia, it is possible to map the articles onto a taxonomic structure based on the category from which the article was extracted [3]. By selecting a category, the user’s ability to access information is restricted based on the list of available properties for that particular category in the taxonomic structure.

Using the available properties for the category that the user has selected, the user is able to query the data by applying filters based on the properties that they have selected to progressively reduce the result set [3]. To further assist the user, the result set can be analyzed and only properties that provide the means to reduce the result set further are presented to the user. Also, the properties are checked to make sure that they will not return empty result sets to the user.

By progressively reducing the result set, the user can track back and forth in their query to fine tune the results that they are presented with [3]. If we consider faceted browsing as a way to navigate Wikipedia, users would start with a generic list of classification such as albums, automobiles or biography. Users would then be provide with a list of relevant properties which the user can select to create filters that would reduce the number of results. Users can reiterate through the list of adjectives and create as many filters as it would require to find the article that they are looking for.

The disadvantage of using faceted browsers is that the taxonomy structure has to be predefined and mapped onto the different ontologies that it will use [2, 13]. This means that there is a need to maintain a taxonomy tree as part of the application and users are bound to the way in which it is structured when querying. It is possible for some of the content to be placed in the wrong area of the taxonomy structure, effectively hiding it from the end user. An example is the taxonomic structure that has been generated for
each category in Wikipedia done by Haase et. al. in their project “Ask the Wiki” [9].

Another disadvantage in faceted browsing is that it consumes a high amount of resources to compute the results for the user at each iteration of specifying a filter [3]. Each time a user applies a filter, the application requires a computational analysis to be performed on the data to retrieve appropriate results as well as to provide a refined list of properties for the user to create additional filters.

Figure 5.5: Ontogator which is used for browsing semantic information on political figures [12].

An example of an early domain specific faceted browser is the Ontogator application seen in Figure 5.5 by Hyvönen, Saarela and Viljanen [12]. Ontogator is designed specifically for browsing semantic information on political figures. It makes use of a small but well-defined ontology which describes information about political figures. The Ontogator application provides its faceted browsing capability as a fixed set of predefined graph patterns for users to filter out politicians based on the users interest.

The filters are visible in the area called “Vilinnat” on the upper left corner in Figure 5.5. They look like labels applied to the page and are colour coded so that the user can easily identify where the filters have been applied in their articles. Different ways of representing the information is provided through a series of tabs where the user can easily
switch between them to make comparisons on the different politicians.

To help speed the search process and reduce the complexity of semantic searches, a form of keyword search could be combined with faceted browsing [9, 10, 26]. In such a scenario, the user begins with a keyword search and a list of results is shown to the user. The user is also provided with several facets based on the query results for the user to further refine their result if they are not able to locate what they are looking for. When the user applies a filter based on a facet, only results in the original list that match the filter are displayed to the user.

The way that the appropriate facets are displayed to the user can be determined using one of two methods [9]. The first which is the easier method to implement; is to analyze the results and display the facets with the highest amount of matches [9, 10]. This has the possibility of misleading the user into thinking that what they are looking for does not exist if the result item falls into a facet with very little results in a relatively large result set.

The second method is to provide a form of computed heuristic analysis on the user’s query to determine the most appropriate facets to display [9, 10]. For instance, if the users keyword is Ford, evaluating that term on a category index would place automobile as the closest matching classification. From the classification, a list of appropriate facets is shown to the user based on the taxonomy tree structure. It is also possible to combine the two methods in order to allow for cases where there is still a high number of available facets after applying heuristic analysis on the query [10].

![The multiple domain faceted browser by OpenLink Software.](image)

Figure 5.6: The multiple domain faceted browser by OpenLink Software.

An example of a application that combines keyword search and faceted browsing is
the OpenLink faceted browser seen in Figure 5.6. Upon reaching the OpenLink faceted browser, users are presented with a simple text box, much like other search engines like Google\textsuperscript{4}, Yahoo\textsuperscript{5} and Bing\textsuperscript{6}. There, they proceed by specifying the search phrase that they want to use.

In Figure 5.6, the term “ferrari” was used as the search phrase. The grey area at the top shows the current filters that have been applied to the search. Users then specify the relationship, or property that they want to filter by with the panel on the right. Once they have chosen a property or relationship, the user is then presented with a list of possible values to select from. Each time a filter has been specified, the results are displayed to the user in the main area.

We found the OpenLink faceted browser to be confusing and difficult to operate as there is no guidance and it does not have an intuitive layout. After we specified a keyword and began our search, we were lost on how to further specify more filters to reduce the result set as the controls on the right panel is rather cryptic. The performance of the OpenLink faceted browser was promising however; returning results within less than a second.

Another faceted browser to implement a combination of keyword search and faceted browsing is the Neofonie\textsuperscript{7} faceted browser by the people involved in the DBPedia project. In Figure 5.8, the area marked as 1 allows users to specify a keyword search phrase. Area 2 is a list of possible facets for the results where users can create filters to reduce the set of results and provide a context to their search. Each facet also provides a count of how many articles would match that particular filter as a guide on choosing which filters to apply. Once a new filter has been defined, it is shown in Area 3 where users are given the opportunity to review and remove filters. The area marked as 4 shows excerpts from each article in the result set making it easier for the user to identify if it is indeed what they are looking for before they proceed.

The Neofonie faceted browser is a great improvement over the OpenLink faceted browser as its use is intuitive enough that user are able to make use of it without the need for lengthy manuals. It can be said that it bridges the gap between the power of faceted browsing with the simplicity of current day keyword search. At the moment, Neofonie is only available to be used for the searching of articles within Wikipedia.

There are large commercial websites that make use of similar faceted browsing capabilities such as eBay and Amazon \cite{10}. This is because those Web stores have well structured product information based on schemas of relational tables. It is therefore possible for them to implement indexes over the data to provide multi-dimensional querying

\textsuperscript{4}http://www.google.com  
\textsuperscript{5}http://www.yahoo.com  
\textsuperscript{6}http://www.bing.com  
\textsuperscript{7}http://dbpedia.neofonie.de/browse/
such as faceted browsing. Implementing faceted browsing as a general search method over the Web however is a large scale exercise as semantic data needs to be generated for all content before faceted browsing techniques can be used.

In an evaluation done by Haase et. al., they found that users tend to be content with simpler interfaces such as keyword searching and many of their study group found that faceted based browsing to be unnecessary complications [9]. Using the methods proposed such as the Neofonie faceted browser according to Hahn et. al. is the best way of introducing semantic querying capability to the user [10]. Through the years, general users have already adapted to keyword search and found ways to work around its limitation. It will take awhile for users to understand the power that faceted browsing provides and get accustomed to using it. Areas that would help speed the adoption of semantic queries are areas where keyword search capability is very limited, such as the ability to search over media formats like audio and video content.
5.4 Enhancing User Applications

Aside from providing enhanced searching capability, semantic technologies can be used as a means to develop application enhancements that are based on semantic data. Using semantic data, it is possible to create an enhancement to the MediaWiki software that would increase the quality of information presented in each article. For example, semantic data can be used to detect missing or broken links between articles in Wikipedia and suggest to contributors on how they could be fixed. The consistency of information between articles can also be increased by placing similar articles into a consistent layout and informing authors on missing information in their articles.

Enhancements to Wikipedia may also include the automatic creation of recommended articles based on the article that the user is currently viewing [12, 31]. This reduces the amount of work an author has to do when producing articles as they no longer need to think of related articles and manually creating links to them. As an example, if a user is viewing the article on a well known political figure, the user would be shown a list of similar well know political figures or articles pertaining to information on the geographical area that the political figure has influence over.

![Figure 5.8: Mobile geolocation software enhanced with semantic information [5].](image)

A novel use of semantic data to enhance an application is the DBPedia Mobile software produced by Bizer et. al. [5]. The DBPedia Mobile software is an enhancement to a global positioning system (GPS) map application by providing interactive content on nearby locations based on information from semantic data. By tapping on a point of interest, the user is taken to the Wikipedia article on the location so that the user can read up more
on that particular location.

Because of the interlinking capability of RDF over multiple sources of data, it is also possible to overlay the history of a person onto a map and trace their migration over time as well [5]. It is also possible to make use of semantic technologies in commercial use as well by using the structured information from the relational tables to automatically derive classification and aid product matching [31].

If a user’s search behaviour is profiled, it is also possible to provide a list of content of interest to the user [12]. This is the method used by major search engines to provide relevant advertising content together with the search results. If a user were to use a keyword *holiday* in their search, and then define *Australia* in their country facet; the system can deduce that the user is searching for a *holiday destination* in *Australia*. The system can therefore recommend to the user several holiday destinations in Australia that it knows of in its repository of semantic information.

### 5.5 Other Possible User Interface Enhancements

In our project, we wanted to see if there are any other ways that we could extend user interface enhancements to make it even easier for end users. Since we were creating interface enhancements for Wikipedia and did not have the ability to make direct changes to its implementation, we needed a way that allows us to introduce user interface enhancements without directly modifying the MediaWiki code-base.

Isbell and Butler have implemented their interface using the MediaWiki software to serve Wikipedia content while using *Jena*[^8], to serve out the semantic information as a separate access point [13]. They then merge the two pieces together using the *Firefox* Web browser by using a plug-in called *GreaseMonkey*, in order to merge both content and semantic information together. *Jena* is an open-source semantic Web framework built for *Java* to make it easier to work with semantic data while *GreaseMonkey* is a *Firefox* Web browser plug-in that allows users to install or write *Javascript* code on the *Firefox* Web browser.

A security limitation of Web browsers requiring *Asynchronous Javascript and XML (AJAX)* requests to only work with the source domain means that Isbell and Bulter had to use workarounds such as dynamic script tag generation or hosting their own copy of Wikipedia on the same domain [13].

Since semantic data extracted from Wikipedia can exist as a separate entity from Wikipedia, it is possible for us to create an external interface layer that would inject content onto the existing Wikipedia user interface [11, 13]. This eliminates the requirement for client-side scripts to be install via plug-ins such as GreaseMonkey and overcomes the

[^8]: http://jena.sourceforge.net
security limitation of Web browsers that only allow AJAX requests to be performed on
the originating domain of the document.

![User Interface Modules](image)

**Figure 5.9: An overview of the architecture used.**

The architecture of our system can be seen in Figure 5.9, where Wikipedia and DB-
pedia is treated as the source of our data. Since DBpedia is linked across other semantic
data repositories, we have access to their data as well. In our system named *Swiki*, we
split our functionality into two virtual folders which are “wiki” and “data”. The func-
tionality provided through the “wiki” folder deals with content coming from Wikipedia
while the functionality exposed through the “data” folder deals with the semantic data
from DBpedia.

In order to inject content into Wikipedia articles as it passes through *Swiki*, we needed
to parse the HTML content in our system to manipulate it. For that, we tried several
HTML parsers: *HTML Cleaner*, *JTidy*, *Tag Soup*, *Xerces* and *Neko*. We were surprised
at how appalling the quality of the HTML parsers were, hence the need to try so many
different ones.

The first that we tried was the *Xerces* parser which was built in as part of the enterprise
version of Java (J2EE). After parsing an article from Wikipedia and outputting it to the
*Firefox* Web browser, no content was rendered. We checked the source that the browser
had received and the content was there but somehow, the content was not rendered. We
had the same issue with the *Neko* parser as well.

The *JTidy* parser did a little better by rendering the main heading and the sidebar but
the main content was missing. The output from *HTML Cleaner* was rendered properly
with the exception of HTML encoded special characters such as apostrophe which was rendered as ‘apos; instead. Finally, we found Tag Soup which managed to parse and output all the content correctly.

To facilitate easy development and testing of different user interface enhancements, we create a module loader which injects different modules in the articles after they are parsed. Figure 5.10 describes the flow of our module injector as it passes through the Swiki system. Different user interface enhancements were then built as a series of modules which we can load or remove as we choose. Each module injects some Javascript code as well as some HTML content to provide the functionality of that module.

Once we had created the infrastructure that we need to trial different kinds of user interface enhancements we proceeded to examine what would be useful. Based on the current Neofonie faceted browser, we noticed that it was not able to handle “OR” conditions and also it did not cater to range selections very well.

We came across the PriceSpy\textsuperscript{9} website which list many electronic products from different retailers for users to search. Because the products are well defined by their product information, they made use of charts to provide distribution information of the products based on different properties. As users browse the site, they could filter the listing of products based on ranges in the chart, however the feature is not available if users did a search instead.

This was an interesting feature that we felt is possible to provide with semantic data,

\textsuperscript{9}http://www.pricespy.co.nz
Figure 5.11: Search interface with better support for ranges.
therefore, we drafted an interface to provide such functionality. As seen in Figure 5.11, the user would begin with a keyword search as with the Neofonie faceted browser. From there, we display a histogram chart to users showing the distribution of articles with a certain property. User are then able to remove results by deselecting properties that they do not want. As for range values such as years, we display a line graph based on the number of articles that correspond with that particular year. Users can then select a range by sliding the slider.

![Slider](image)

**Figure 5.12: The slider module allowing users to select a range of values.**

We then moved on to produce a prototype module of the slider as seen in Figure 5.12. The prototype showed promising results although creating the line graph took a long time to load because values for each interval had to be calculated and retrieved from DBpedia. To increase the speed of creating the line charts and histograms, the values would need to be precalculated for the different intervals. Because the amount of semantic data from DBpedia is so large, it is difficult to determine which values to precalculate.

Another module that we created has the ability to automatically create a list of related articles based on the article the user is currently viewing. Once the list of related articles is determined, it is presented in a floating window so that the user is free to move it to a place that is suitable for them.

Figure 5.13 shows an example where a user is viewing the article on the boardgame *Chess*. In the floating windows is a list of other similar articles which the user can expand to see a short excerpt of that article. If the related article is something that they are interested in, they can click on the excerpt to view the article itself. In case the user is not interested in the related articles, the user can also close the floating window to get it out of the way.

The third module that we developed is similar to the “Instant” feature on the Google search engine. Figure 5.14 shows a demonstration of the feature where as we type each character, the list of results is updated to show most relevant results to what is displayed in the text box. The search results are based on the same algorithm that the related articles use, therefore, if we were to type “chess”, we will be presented with a list of articles related to the article on *Chess*. Its usefulness comes when longer phrases are used so that users can get some form of feedback as to whether they are using appropriate terms in their search phrase.
To gain an understanding of the effectiveness of the user interface enhancements, we presented 10 people with the different enhancements. We had casual discussions with them on how useful each enhancement is on a day-to-day basis and which of the features provided the most benefit to them. The results were that 9 out of 10 of them found the slider module useful and would like to see further development on them. They said they would like it not only for Wikipedia articles but across other content on the Web as well as it is currently impossible to search using ranges.
5.5 Other Possible User Interface Enhancements

Figure 5.14: A text box that provides instant searching on Wikipedia.
6 Related Work

6.1 DBPedia

DBPedia started out as a research project that tests the feasibility of incorporating semantic technologies over a diverse and constantly evolving data corpus [2]. It leverages the enormous amount of textual data from the articles in Wikipedia to create a large corpus of semantic data [5, 10]. Wikipedia was chosen as the content source for DBPedia because it was under constant revision and would benefit the most from semantic query capabilities which is in line with its project goals [2].

Since then, the main objective of DBPedia has shifted to be an ongoing effort to maintain a repository of semantic knowledge that is consistent with Wikipedia [2]. The semantic data in DBPedia is made available over the Web for applications with semantic technology capabilities to consume.

DBPedia makes use of its own ontology derived from extracting commonly used attributes from infobox templates found in Wikipedia [5, 10]. It also applies other ontology sets such as OWL, YAGO and FOAF on top of its own [2]. DBPedia makes use of a community effort to write rules on which entities extracted from Wikipedia is matched to the different ontologies.

The semantic data that is held by DBPedia is also interlinked with other semantic data repositories to provide a comprehensive body of semantic knowledge [5]. Some examples of repositories that it links to is the New York Times where news headings are linked to
DBPedia concepts and Geonames where geographic information is linked with articles on locations.

DBPedia exposes a SPARQL endpoint to the public in order to enable its users to query the DBPedia data set [2]. This provides the ability to create sophisticated querying interfaces such as the faceted browsers and visual queries for Wikipedia [10]. Using semantic data, user interface enhancements for Wikipedia such as suggested articles and related content are also possible.

6.2 WordNet

WordNet is a lexical database for the English language that groups together words that represent a common concept [15]. Using WordNet as a reference, it is possible to create a more comprehensive semantic graph for the English language by relating similar words together. The results are that users are able to be more flexible in expressing their semantic query and will still achieve the appropriate results.

6.3 Yet Another Great Ontology (YAGO)

The main goal of YAGO is to provide a consistent and accurate ontology that users can refer to [23]. YAGO is derived through the combination of vast knowledge available in Wikipedia with the structuring and cross references provided by WordNet. In creating YAGO, Wikipedia’s disambiguation redirects and leaf categories from the category system were mapped onto its own hierarchy system. [5, 10, 23].

6.4 FreeBase

FreeBase has the same operating principles as Wikipedia where it relies on the contribution from users to extend its data [5, 10]. When FreeBase started, it extracted content from Wikipedia and parsed it as semantic data for its initial base [5, 10]. From there, it has had a healthy number of active contributors maintaining and extending the information available [13].

The semantic data that FreeBase holds is made freely available to the public through its user interface. FreeBase also provides its data in the form of RDF with an API for developers to write applications with semantic knowledge capabilities. Linking of FreeBase with other repositories is also possible such as what DBPedia has done where similar entities are tied together with a RDF link [10]. This allows users to benefit from a source that is continuously updated from Wikipedia and a source that is actively maintained by its contributors for their semantic data needs.
6.5 Semantic Wiki Applications

Wikis are a convenient way to manage knowledge in the digital world as it is quick and simple to use because there is no overhead cost in having to create a structure and adhere to it. The downside to that is that Wikis are not able to provide intuitive navigational features such as complex querying, related content and concept based browsing. This is one of the reasons why Wikis would benefit most from implementing semantic technologies as it provides a way in which structure can be placed on to Wikis automatically. In this section, we look at Platypus Wiki, SemperWiki and Semantic MediaWiki which builds semantic knowledge capabilities into itself.

6.5.1 Platypus Wiki

As a workaround to allow for some form of structure, some Wikis use taxonomies and hierarchical structures to organize its data. Although taxonomies and hierarchical structures provide some navigational and organizational enhancements, it limits the number of paths the article can be accessed [6, 18].

The Platypus Wiki attempts to close this gap by introducing RDF as the organization of the data stored in its Wiki [6, 18]. In the Platypus Wiki, data is stored in a directed graph and multiple links can be associated between each other, increasing the number of available paths to each entity in the graph. The Platypus Wiki makes use of the OWL ontology to represent the properties that link the resources together.

Users of Platypus Wiki are responsible for annotating the articles with metadata to provide the semantic capabilities and feedback is provided in the form of navigational components for the article [6]. An interesting feature included in Platypus Wiki is that it allows the automatic creation of links based on matches to a keyword lookup [6]. If a particular keyword is found in the article, it is linked to the relevant article automatically, reducing the amount of work that author has to do. However, to prevent all content from becoming links; keywords have to be maintained manually with user interaction so that only well-known keywords end up being automatically generated as links.

6.5.2 SemperWiki

SemperWiki is a personal Wiki where it works more as a personal information management system [18]. Personal Wikis are basically like note-taking application such as Tomboy on Linux and OneNote in the Microsoft Office Suite. SemperWiki provides functionality to allow users to annotate their data with semantic information as well as tools for users to make use of the annotations.

One of the shortcomings of Platypus Wiki which is addressed in SemperWiki is the lack
of enticement to provide semantic annotations when authoring content [18]. According to Oren, there is a need for immediate feedback during the authoring process in order for authors to see the benefit and be motivated to include semantic annotations as they create or update content [18]. SemperWiki provides immediate feedback through automatically generated navigational links to related content and grouping of similar content into a hierarchical taxonomy [18].

The use of semantic search in a personal context is somewhat questionable as it is only suited for people who create large collections of personal information and lack organizational capability. In a collaborative space, each individual has a different mindset of organization and that is where semantic queries bridge the gap.

### 6.5.3 Semantic MediaWiki (SMW)

The SMW project is an effort to extend the existing MediaWiki software with semantic processing capabilities [14]. In SMW, semantic relationships are introduced via an extension to the markup syntax [14]. This means that content has to be created from scratch with the extra overhead of adding semantic relationships in mind.

To encourage the use of SMW with its extra overhead of specifying semantic relationships during the authoring process; it needs to exhibit the benefits of semantic data very strongly [14, 18]. One of the interface features implemented in SMW is the factbox which provides authors with immediate feedback on available semantic information in the article that they are authoring [14]. The factbox provides viewers a quick summary of the article to scan through and determine if the article is indeed what they are looking for. It also provides a list of links to related content which assist viewers in navigation.

SMW also extends the capability of MediaWiki by providing a way in which dynamic content can be introduced into articles [14]. The way that this is implemented in SMW is that its syntax allows for semantic queries to be inserted into the contents of the article. When the article is viewed, the query is run to generate the output for the dynamic content section of the article. This is useful when there is a need to include volatile data that varies over time. An example demonstrated by Krötzsch et. al. is the ability to have a list of upcoming conferences within an article [14].

### 6.6 Natural Language Processing (NLP)

NLP is synonymous to the Semantic Web as both tries to achieve some form of machine intelligence over data [11]. Natural language processing techniques lends to semantic technologies the ability to trawl through data and annotate them with semantic information in an automated manner [20]. What it takes from the semantic technologies is the repos-
6.6 Natural Language Processing (NLP)

itory of semantic knowledge that it can use to apply reasoning algorithms on in order to provide intelligence.

Some effort has been made by Yahoo! Research Barcelona to annotate the contents of Wikipedia with semantic information through the use of NLP techniques; however, it is still in its early stages [5, 10]. Their goal is to provide the Yahoo! search engine with more relevant search results and better result ranking.

According to Hoffart et. al., current natural language processing capabilities have not reached a point where the rate of accuracy is of an acceptable level to users [11]. What they propose is to have natural language processing techniques implemented as a suggestive interface rather than an enforcing process. In their study, instead of processing content in a Wiki to apply a structure to the Wiki automatically; they built an interface that works together with the Wiki to provide suggestions to the user on creating a better structure when authoring.

Another approach is the Powerset Search Engine which applies NLP techniques on the content within Wikipedia to create a reference set of words [5, 10]. The same technique is then used on the query from the user to form another reference set which is cross-referenced with the one from Wikipedia in order to produce the result of the search. Powerset is able to provide a relatively accurate list of results with the exception of a few corner cases.

One major downside of NLP techniques is the associated compute cost that it incurs to translate natural language into a form that the machine can reason with [15]. NLP techniques always require that data is processed and stored in an intermediary form in order to be responsive to the user. Semantic data is one of the intermediary forms in which NLP can make use of; therefore, a blend of the two methods would be an ideal solution to provide intelligence over data.

PowerAqua is a project by Fernandez et. al. which attempts to bridge NLP type queries with semantic queries such as SPARQL [8]. The purpose for this is to reduce the complexity of creating semantic queries by translating natural language queries using NLP techniques into a formal semantic query. Because natural languages queries are not restricted in context, the generated semantic query has to be able to span multiple domains [8].
We have covered the process of creating user interface enhancements for Wikipedia based on semantic data. Semantic data allows users to perform complex queries to quickly and easily find relevant information that they are looking for. Besides querying, semantic data provides the means for machine processing capability to allow applications to make use of the knowledge which is embedded into Wikipedia.

Once the semantic technologies developed using the semantic data from Wikipedia reaches a mature stage, it can be applied to all content on the Web. When that happens, users would have the ability to perform complex queries with ease to obtain any information on the Web. The Web would then become a definitive source of knowledge where everyone who seeks information can turn to and obtain the information they require quickly and with ease.

7.1 Achievements

In this project, we have gone through the process of making user interface enhancements for Wikipedia based on semantic data. Starting by understanding what is semantic data, followed by the extraction of semantic data from Wikipedia, we then examined how the semantic data can be accessed in order to make it useful. Finally, we created several user interface enhancements as loadable modules in a platform that combines the content from Wikipedia with semantic data.
The following list highlights the main achievements of this project:

- **Extraction of semantic data from Wikipedia.** We have successfully extracted the semantic data from Wikipedia and applied ontologies on to it in order to provide a consistent way to access the semantic data.

- **Creating a platform for interface enhancements.** A platform was created to be able to dispense Wikipedia’s content enhanced with semantic technologies. The platform allowed us to quickly and easily develop user interface modules that can be injected onto content from Wikipedia.

- **Discovering range value querying for users.** Using a graph to provide visual feedback to the user incorporating slider and toggle controls to allow the user to make queries based on range values rather than fixed ones.

### 7.2 Future Directions

Giving users visual feedback and the ability to query based on a range rather than a specific value was a concept that we had found to be very useful. Therefore, we would like to continue our development of the graph interface as well as to build prototypes of other new ways to query Wikipedia.

One of the things that we would like to have a look at is improving the speed of the graph interface to be able to load in less than a second. We would also like to analyze the behaviour of a large number of users when they search by profiling them using logs to discover their search patterns in order to further develop better user interfaces. Another item that we would like to get done is to study the feasibility of applying our methods across all content on the Web.

### 7.3 Reflections

In this project, we covered a wide scope of work which could be divided into 3 distinct areas: extraction of semantic data, creating an architecture for development of semantic user interfaces and the development of semantic user interfaces. For future projects, it would be better to focus on a single aspect so that we can have more depth to our research.

The time spent on the different parts of the project was also erratic and we were also sidelined by components that we were waiting for. There needs to be small fixed milestones set for future projects to ensure that time is well managed and objectives are clear.


