

Enhancing Wikipedia with Semantic Technologies

Lian Hoy Lee
The University of Auckland
38 Princes St
Auckland
New Zealand
llee058@aucklanduni.ac.nz

Christof Lutteroth
The University of Auckland
38 Princes St
Auckland
New Zealand
lutteroth@cs.auckland.ac.nz

Gerald Weber
The University of Auckland
38 Princes St
Auckland
New Zealand
g.weber@cs.auckland.ac.nz

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 survey currently proposed semantic enhancements for Wikipedia, which is one of the largest repositories of knowledge at present. Furthermore, we analyze the benefit and feasibility of the different technologies, and we identify certain gaps in current proposals. We then propose a new 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 this 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. We were also able to create a new way to leverage semantic data allowing users to perform a query based on a range of values, with the guidance of a graph to visualize the distribution of the results..

Semantic Web, Search, Semantic Wikis

1. INTRODUCTION

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 (Auer and Lehmann 2007). Users have to be familiar with the underlying ontology before they are able to formulate queries which will return results (Wang et al. 2008). 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 (Auer and Lehmann 2007; Bizer et al. 2009). 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 (Hyvönen et al.

2004). 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 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.

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 relevance to what they are looking for. This method best suits 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

¹<http://www.nzherald.co.nz>

²<http://www.ebay.com>

³<http://www.youtube.com>

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 but 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 (Auer et al. 2008). 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 uses 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 (Isbell and Butler 2007; Lei et al. 2006). 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 paper, we 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*, which are analysed in Section 2. We proceed to propose new semantic query interfaces in Section 3. In this section, we explain how semantic data from different sources can be integrated and shown in a consistent interface, enabling new features in the Wikipedia such as range queries, automatic recommendations and instant search. The paper concludes with Section 4.

2. SEMANTIC QUERY INTERFACES FOR WIKIPEDIA

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* (Hahn et al. 2010). 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 manual interaction (Nguyen et al. 2010). 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. (2006) expresses four key points that should be observed when developing interfaces to provide access to the power of semantic queries. 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. The system should be able to calculate and present the results in a short amount of time.

Völkel et al. (2006) shows support for these points as they place importance on similar points. 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 (Schaffert et al. 2005; Wang et al. 2008; Zhou et al. 2007). 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.

2.1. Graph Pattern Builders

A *graph pattern builder* is a means of querying semantic data by describing the graph pattern that we want to look for (Auer and Lehmann 2007).

The way it works is by describing the concept of querying semantic data in a way that allows the user to visualize its operation (Nguyen et al. 2010). 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 1: SPARQL Explorer - An interface to execute SPARQL queries.

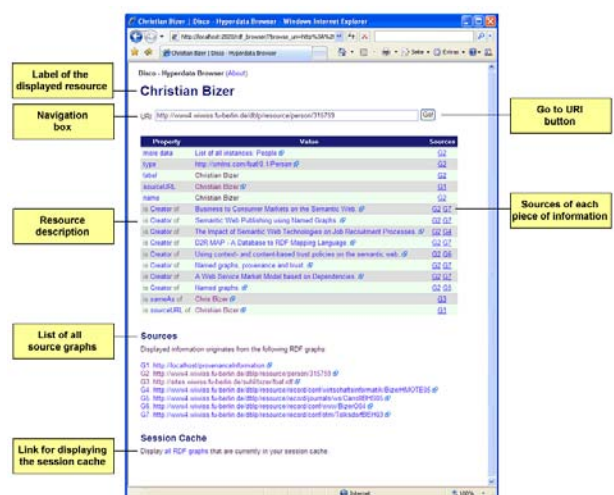


Figure 2: Browsing semantic data using Disco.

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 (Bizer et al. 2009). There is also an output area where the results are displayed, and some even include a possibility for users to choose the output format. The *SPARQL Explorer* shown in Figure 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 (Fernandez et al. 2008; Auer and Lehmann 2007). Besides syntax issues, users would often find an empty result set because their query defines non-existent relationship 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 in 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 (Auer and Lehmann 2007).

Wikipedia Query Builder

Query: Please provide triple patterns, the results should match to. Prefix variables with "?", use ">", "<", "=", "~" (Regex) for comparisons. Alternatives can be given by using "|".

Subject	Predicate	Object
?city	leader_name	?leader
?city	subdivision_name	United_States
?city	e	>1000
	established_date (261)	
	established_date2 (83)	
	elevation_ft (62)	
	elevation (12)	
	established_date3 (9)	
	established_date1 (1)	

?city	?leader	>1000
Cadillac, Michigan	Ronald Blanchard	1328
Denver, Colorado	John Hickenlooper (D)	1609
Roswell, New Mexico	Sam LaGrone	1089
Santa Fe, New Mexico	David Coss	2231
Las Cruces, New Mexico	William Michael Mattiace	4000 ft - 1219
Artesia, New Mexico	Manuel Madrid	1030
Carlsbad, New Mexico	Robert Forrest	1004

[Permalink](#)

Browse Wikipedia

You can browse a part of the Wikipedia extraction with OntoWiki at <http://wikipedia.3ba.se/filem>

Download

NTriples dump of all extracted RDF statements from Wikipedia: wikipedia.nt.bz2

Sourcecode is available from: <http://prowl.sf.net>

Save current query

Label Save

Previously saved queries

- Soccer player with tncost number 11 from club with stadium with >40000 seats born in a country with more than 10M inhabitants
- Films with music from John Williams
- Films with Quentin Tarantino as actor, producer, or director

Figure 3: Query Builder - A triple by triple method of specifying semantic data queries.

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 (Bizer et al. 2009). To simplify the process of creating a SPARQL query, a step by step approach such as query builders are used (Auer and Lehmann 2007; Bizer et al. 2009). As seen in Figure 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.

The creation of an application like the *relationship finder* as seen in Figure 4 is more of a tool to assist users in understanding how the entities in a corpus of semantic data are linked together. As it displays a graph allowing the user to visualize how the entities specified are linked, it provides a way

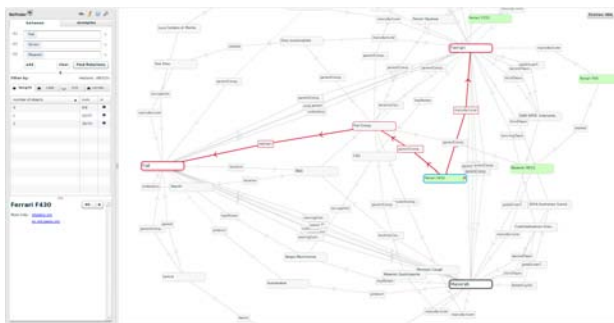


Figure 4: Relationship finder used to find how Fiat, Ferrari and Maserati are linked together.

to find possible properties and relationships to use when constructing a semantic query (Bizer et al. 2009).

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.

2.2. 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 (Haase et al. 2009; Hahn et al. 2010). Faceted browsing makes it easier to construct queries as semantic queries are easily represented as a set of 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 (Auer and Lehmann 2007; Isbell and Butler 2007; Wang et al. 2008).

In faceted browsing, taxonomic structures help guide the user by restricting the scope of their query and placing a context onto their search pattern (Auer and Lehmann 2007; Hyvönen et al. 2004). 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 (Auer and Lehmann 2007). 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 (Auer and Lehmann 2007). 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 (Auer and Lehmann 2007). If we consider faceted browsing as a way to navigate Wikipedia, users would start with a generic list of classifications, such as albums, automobiles or biography. Users would then be shown 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 (Auer et al. 2008; Isbell and Butler 2007). 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. (2009) in their project "Ask the Wiki".

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 (Auer and Lehmann 2007). 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.

An example of an early domain specific faceted browser is the *Ontogator* application seen in Figure 5 by Hyvönen et al. (2004). *Ontogator* is designed



Figure 5: Ontogator which is used for browsing semantic information on political figures (Hyvönen et al. 2004).

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. 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 are provided through a series of tabs, where the user can easily switch between them to make comparisons between the different politicians.

To help speed up the search process and reduce the complexity of semantic searches, a form of keyword search could be combined with faceted browsing (Haase et al. 2009; Hahn et al. 2010; Wang et al. 2008). 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 (Haase et al. 2009). The first, which is the easier method to implement, is to analyze the results and display the facets with the highest amount of matches (Haase et al. 2009; Hahn et al. 2010). 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 (Haase et al. 2009; Hahn et al. 2010). 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 to the query (Hahn et al. 2010).



Figure 6: The multiple domain faceted browser by OpenLink Software.

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

In Figure 6, the term “ferrari” was used as the search phrase. The gray 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 in the right panel are rather cryptic. The performance of the

⁴<http://www.google.com>

⁵<http://www.yahoo.com>

⁶<http://www.bing.com>

OpenLink faceted browser was promising, however, returning results within less than a second.

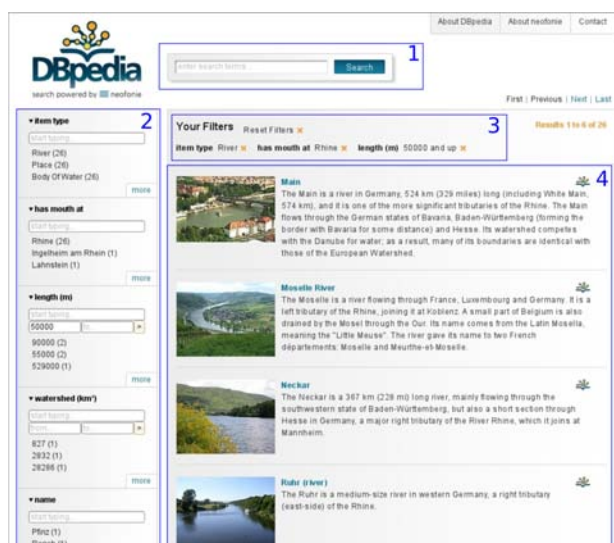


Figure 7: *Neofonie* - A simplified approach to faceted browsing (Hahn et al. 2010).

Another faceted browser to implement a combination of keyword search and faceted browsing is the *Neofonie*⁷ faceted browser by the people involved in the DBPedia project. In Figure 7, 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 users 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 (Hahn et al. 2010). 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 such as faceted browsing. However, implementing

⁷<http://dbpedia.neofonie.de/browse/>

faceted browsing as a general search method over the Web 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 in their study group found faceted browsing to be an unnecessary complication (Haase et al. 2009). According to Hahn et al. (2010), using the methods proposed such as in the *Neofonie* faceted browser is the best way of introducing semantic querying capability to the user. Through the years, general users have already adapted to keyword search and found ways to work around its limitations. 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.

2.3. 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 about missing information in their articles.

Enhancements to Wikipedia may also include the automatic creation of a list of recommended articles based on the article that the user is currently viewing (Hyvönen et al. 2004; Zhong et al. 2002). 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.

A novel use of semantic data to enhance an application is the *DBPedia Mobile* software produced by Bizer et al. (2009). 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



Figure 8: Mobile geolocation software enhanced with semantic information Bizer et al. (2009).

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 more about it.

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 (Bizer et al. 2009). It is also possible to use semantic technologies commercially by using the structured information from the relational tables to automatically derive classification and aid product matching (Zhong et al. 2002).

If a users search behaviour is profiled, it is also possible to provide a list of content of interest to the user (Hyvönen et al. 2004). 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.

3. INTEGRATING NEW USER INTERFACES WITH WIKIPEDIA

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 (2007) have implemented their interface using the MediaWiki software to serve Wikipedia content while using *Jena*⁸, to serve out

⁸<http://jena.sourceforge.net>

the semantic information as a separate access point. They then merged the two pieces, the content and semantic information, together in the *Firefox* Web browser by using a plug-in called *GreaseMonkey*. *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 Butler (2007) had to use workarounds such as dynamic script tag generation or hosting their own copy of Wikipedia on the same domain.

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 (Hoffart et al. 2009; Isbell and Butler 2007). This eliminates the requirement for client-side scripts to be installed via plug-ins such as *GreaseMonkey*, and overcomes the security limitation of Web browsers that only allow *AJAX* requests to be performed on the originating domain of the document.

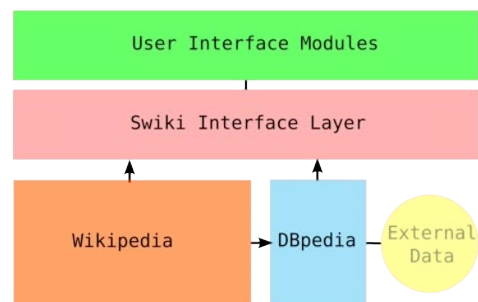


Figure 9: An overview of the architecture used.

The architecture of our system can be seen in Figure 9, where Wikipedia and DBpedia are treated as the sources 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 functionality 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, and were surprised how

poorly they performed. Finally, we found the *Tag Soup* parser, which managed to parse and output all the content correctly.

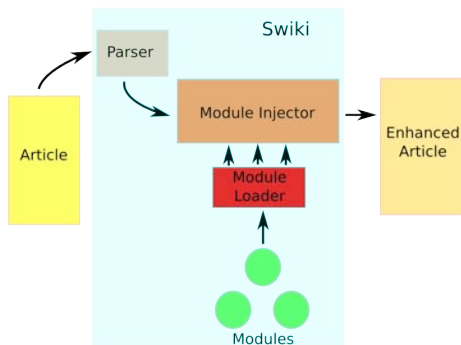


Figure 10: Injecting user interface modules as articles pass through.

To facilitate easy development and testing of different user interface enhancements, we created a module loader which injects different modules into the articles after they are parsed. Figure 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.

3.1. Enabling Range Queries

Once we had created the infrastructure that we needed 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 for range selections very well.

We came across the interface of the *PriceSpy*⁹ website which lists 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 was not available if users did a search instead.

This is an interesting feature that we felt is possible to provide with semantic data; therefore, we drafted an interface to provide such functionality for general semantic queries. As seen in Figure 11, the user would begin with a keyword search as with the *Neofonie* faceted browser. From there, we display a

⁹<http://www.pricespy.co.nz>

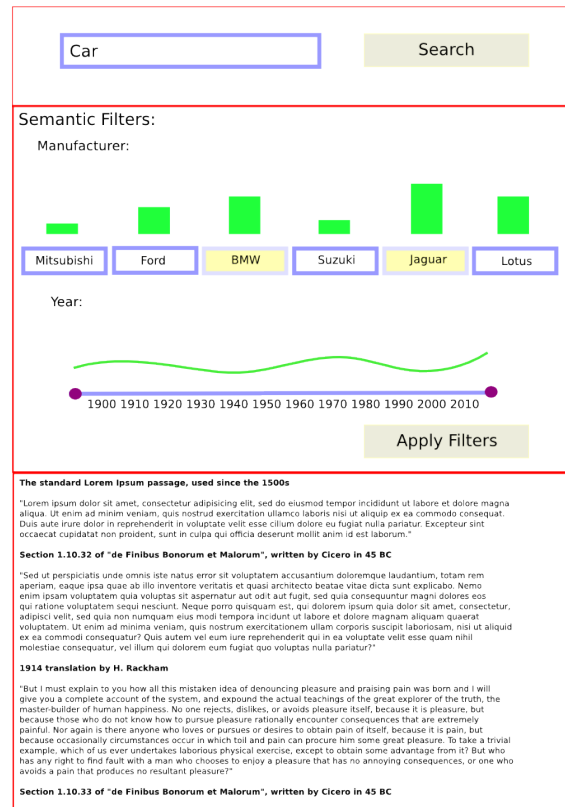


Figure 11: Search interface with better support for ranges.

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.

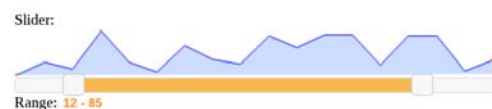


Figure 12: The slider module allowing users to select a range of values.

We then moved on to produce a prototype module of the slider as seen in Figure 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. Contemporary triplestores are not 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.

3.2. Recommending Related Articles Automatically

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.

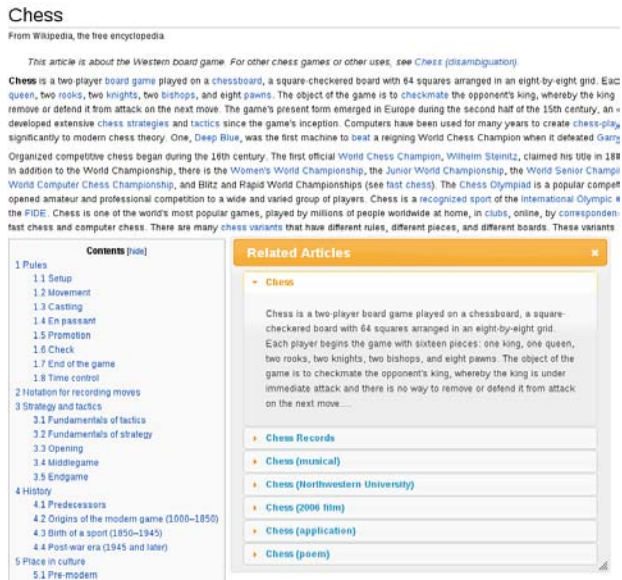


Figure 13: Related articles shown to the user as a floating window.

Figure 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.

3.3. Instant Search



Figure 14: A text box that provides instant searching on Wikipedia.

The third module that we developed is similar to the “Instant” feature in the Google search engine. Figure 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.

3.4. Evaluation

To gain an understanding of the effectiveness of the user interface enhancements, we presented 10 people with the different enhancements, asking them to use it repeatedly over the duration of several days. We interviewed them on how useful each enhancement is and which of the features provided the most benefit to them on a day-to-day basis. The results were that 9 out of 10 of them found the slider module useful and would like to see it further developed. 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.

4. CONCLUSION

We have analysed existing 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.

In this project, we created a platform for semantic interface enhancements for the Wikipedia. The platform is able to dispense Wikipedia’s content enhanced with semantic technologies, allowing developers to design user interface modules that can be injected easily. Furthermore, we created modules for range queries, recommendations and instant search. Particularly useful is the notion of using histograms and graphs to provide visual feedback to the user, for queries based on value ranges. Range queries are hardly supported in existing query interfaces. In addition to making range queries more intuitive, the user interface we propose gives users an idea of how many results to expect in such a query. However, the proposed range query interface is limited by the current performance of triplestores, the databases used to store semantic data.

Once the semantic technologies developed using the semantic data from Wikipedia reach a mature stage, they could be applied to other content on the Web. This is a long-term goal that would give users the ability to perform complex queries that are much more powerful than the current keyword search. However, the extraction of semantic data from the prevalent semi-structured and unstructured web pages is not an easy task, and it will take time until semantic data for many current web pages will be publicly available.

5. REFERENCES

- S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. *The Semantic Web*, page 722-735, 2008.
- S. Auer and J. Lehmann. What have innsbruck and leipzig in common? extracting semantics from wiki content. *The Semantic Web: Research and Applications*, page 503-517, 2007.
- C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia-a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2009. ISSN 1570-8268.
- M. Fernandez, V. Lopez, M. Sabou, V. Uren, D. Vallet, E. Motta, and P. Castells. Semantic search meets the web. In *The IEEE International Conference on Semantic Computing*, page 253-260, 2008.
- P. Haase, D. Herzig, M. Musen, and T. Tran. Semantic wiki search. *The Semantic Web: Research and Applications*, page 445-460, 2009.
- R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. Faceted wikipedia search. In *Business Information Systems*, page 1-11, 2010.
- J. Hoffart, T. Zesch, and I. Gurevych. An architecture to support intelligent user interfaces for wikis by means of natural language processing. In *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, page 1-10, 2009.
- E. Hyvönen, S. Saarela, and K. Viljanen. Application of ontology techniques to view-based semantic search and browsing. *The Semantic Web: Research and Applications*, page 92-106, 2004.
- J. Isbell and M. H Butler. Extracting and re-using structured data from wikis. Technical report, Technical Report HPL-2007-182, Hewlett-Packard, 2007.
- Y. Lei, V. Uren, and E. Motta. Semsearch: A search engine for the semantic web. *Managing Knowledge in a World of Networks*, page 238-245, 2006.
- H. Nguyen, T. Nguyen, H. Nguyen, and J. Freire. Querying wikipedia documents and relationships. In *Proceedings of the 13th International Workshop on the Web and Databases*, page 1-6, 2010.
- S. Schaffert, A. Gruber, and R. Westenthaler. *A semantic wiki for collaborative knowledge formation*. Citeseer, 2005.
- M. Völkel, M. Krötzsch, D. Vrandečić, H. Haller, and R. Studer. Semantic wikipedia. In *Proceedings of the 15th international conference on World Wide Web*, page 594, 2006.
- H. Wang, K. Zhang, Q. Liu, T. Tran, and Y. Yu. Q2semantic: A lightweight keyword interface to semantic search. *The Semantic Web: Research and Applications*, page 584-598, 2008.
- J. Zhong, H. Zhu, J. Li, and Y. Yu. Conceptual graph matching for semantic search. *Conceptual Structures: Integration and Interfaces*, page 92-106, 2002.
- Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. Spark: Adapting keyword query to semantic search. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, page 694-707, 2007. ISBN 3540762973.