Web Information Systems

Final Assignment:

User Interfaces for Semantic Web Repositories

Radim Čebiš
December 2007

Table of Contents

1 Introduction

Semantic web applications can be considered as other step of web development. Sometimes they are referred to as *Web 3.0* to remind the progress of the web. Semantic web can bring a lot of new and useful functions for the users. But only new functions are not enough for the success. One of the most significant aspects is user interface. If the users do not like the interface, they will not even try to find new functionality which could be valuable for them. Even very sophisticated application can be ruined by useless interface and even simple application can benefit from usable user interface which increases users' experience.

In this paper I want to focus on user interfaces supporting semantic web. In the first part I will give examples of two semantic web applications. These applications usually use *Web 2.0* aspects so I will get to *Web 2.0* also. The first part will refer to semantic web applications, which are more common and more used in the present because their limited scope and target, allowing them to be user friendly. They are indeed very close to conventional web applications. As one of the application is more like a base for the other applications to build on, I will also introduce some basic principles of working with this application via its own application interface.

In the second part I will introduce Tabulator, which is one possible form of the semantic web browser. The first part was about semantic web applications, which are more traditional ones. The second part will be about user interface for browsing whole semantic web as the giant global source of knowledge – meaning there is not limited scope or target. The browser is only interface for showing information from different interlinked data sources of data.

In the end I will try to summarize the actual state of the user interfaces supporting semantic web and semantic web applications.

2 *MultimediaN E-Culture Project*

2.1 Introduction

This project [1] demonstrates use of semantic web and novel presentation technologies in area of cultural heritage. The main difference between this and other similar projects is that they are trying to use foreign metadata from different sources as much as possible. Adding a collection involves processing original metadata and enriching them with semantic categories. For example author and material are matched to the vocabulary using a syntactic distance and best match. This is mainly done automated way, but manual annotation interface for privileged users is also supported.

As a source of data, 3 collections were included in 2006. Currently application works with five web collections with more than 106 000 items. It hosts five thesauri, namely Getty vocabularies[1] (*Art & Architecture Thesaurus*, *Union List of Artists Names*, *Thesaurus of Geographical Names*), *Stichting Volkenkundige Collectie Nederland*[2] (Dutch ethnology) and *Princeton WordNet*[3]. Getty vocabularies are systematically maintained and they link to each other. For example place of birth of the artist in the *Union List of Artists Names* refers to terms in the *Thesaurus of Geographical Names*.

As a semantic web application searching is more sophisticated. It supports indirect links,

1 http://www.getty.edu/research/conducting_research/vocabularies/
2 http://svcn.nl/
3 http://www.w3.org/TR/wordnet-rdf/

which means that the search will find images from searched style even though they are not of that style, but the painter has linked that style. Also relation search between resources is supported.

2.2 User Interface

User interface is straightforward focusing only on searching and browsing the data repositories. It is divided to different parts. The most important ones are basic search, advanced search, /facet browser and relation search.

2.2.1 Basic Search and Result Pages

In the basic search *Ajax*[4] is used to suggest query refinements which offers specification of the relation between the searched objects and searched word. Of course it suggests only queries with possible hits.

If the relation is not specified by choosing from *Ajax* offer, application groups the results by matching properties which makes the results easily understandable. I see only one issue – image titles like "culture "*Mon" / prefLabel "Mon" / (match)*" might be useful for developers or advanced users, but for usual users they are confusing. It is also against accessibility, but in this case it can be tolerated, because it would be strange to browse collections of art with turned off images. In any case I would rather replace them with more comprehensible titles.

I appreciate time line which visualizes artists' lifespans and pieces of work placed according to their creation date. But one feature is not working – there should be label displayed while the mouse is above the green circle representing the piece of work. Unfortunately this does not work even though it is implemented according to the source code. I tried it using *Mozilla Firefox* 2.0.0.8 and 2.0.0.9 versions and *Opera*. I tried to extract *SVG*[5] *XML* code and open it in *Internet Explorer* with installed *Adobe SVG Viewer 3.03*[6] plug-in but I ended with completely same result.

After selection of the work the detail page is shown. The page is well structured and usually it includes a lot of information. Unfortunately the information is sometimes in Dutch even though the language is set to English. I suppose this might be very confusing for people who do not know any Dutch. I also consider the labels of properties not so well understandable. I would appreciate some shift to natural language or at least some effort to keep the style consistent, because sometimes the first letter is capital sometimes not. But of course it should be kept in mind that the web is trying to show as much data as possible and it is using many big thesauruses. Still I believe that some kind of translation between properties and shown labels would be appreciated.

All the properties link to the page with their description. This function might be sometimes useful as sometimes user wants to see some description and which values the property holds, allowing him to better understand the property. Also for every value we can see the works having this value. Unfortunately this function cannot be used for inspecting the works according to some property, because it does not support any type of listing. This completely disallows meaningful usage while there is so many works and images that it takes too long to load the page with all the predicates and works. I think this function could be useful, but I suppose that it is not supported while the project has its own facet browser which offers better browsing capabilities

4 http://en.wikipedia.org/wiki/Ajax_(programming)
5 http://www.w3.org/TR/SVG/
6 http://www.adobe.com/svg/viewer/install/

in the sense of setting properties' constraints. I will focus on the facet browser later. Also there is compartment for resources which use this property as value for their description, but this view will be hardly used by an ordinary user.

Property values which has some other description also link to the page with some more information about the value. The pages are same as pages with property description with only small differences (value does not have predicates). This time the compartment with resources which use the value to describe themselves is the most important. The resources are grouped according to property which holds the selected value. While property page includes all works which have the property with any value. The page with property value includes only works which have the selected value in any property. This makes the pages with the exact value description more useful while the number of shown works is much lower.

All the pages have unified style and probably they are only changed according to the data received from knowledge base after the query. Same style makes every page familiar to user. Unfortunately it also brings all the negatives which I mentioned above – inconsistent labels, inconsistent language, incomprehensible or missing image titles, and lack of listing.

2.2.2 Advanced Search

The *Ajax* functionality is missing in Advanced Search. Instead of that the user is offered to specialize the query. Via advanced search the user cannot only search the works but also other object types as persons. But as the most useful function for users I consider the time based search. This type of queries unleash the power of semantic web. They are not just simple queries – meaning that user can only input the year, but very sophisticated, because user can enter various types of queries as shown in Table 1.

I	II	III	IV	V	Example
all in after before	-	his (optional) her the {name}	last old late young youth early	period of his(/her) life period life (X) year(s) of his life year age	"in his early age" "in late period of his life " "in the last year of his life" "in the last 5 years of his life"
all in after before	the (optional)	ca.(/c.) (optional)	{century number e.g. 16th or sixteenth}	century	"before the 17th century" "after ca. sixth century"
all in after before	-	ca.(/c.) (optional)	{year} {start year} - {end year}	-	"before ca. 1510" "in 1600-1670"
all in after before	-	{name}	was born is born birth birthdate ulan:birthDate was dead is dead died/die/death death day/death date ulan:deathDate	-	"after monet was born" "after 'van gogh' died"
all in after before	the (optional)	{name}	- -	period (optional)	"in the cubist period" "after fauvism"

Table 1. Query specification options[7]

7 Source: http://e-culture.multimedian.nl/demo/www/faq.html#time

Of course to use the power of these queries user have to learn what is possible. But then she can ask very special questions. The results of advanced search are provided the same way as in basic search.

2.2.3 Relation Search

This type of search offers the user to find relations between arbitrary objects. It finds relations between two *URI*s. This function is very useful, because user does not need to compare two objects. Also this function can go deeper and find relations which are not obvious but transitive.

To use this function user must use "My Art shopping basket". While browsing he can put there different *URI*s by right clicking and selecting "Add to my collection". Then in the Relation Search, *URI*s can be transferred to the fields. This is very useful, because user does not need to put all the *URI*s aside to his text file or somewhere else, but application can hold them for him. Also it is much more simple for inexperienced users who might not know how to use the clipboard. We should also take into account that this application could be used in computer kiosk in museums, where there might not be option to save the *URI* to the text file or clipboard while kiosk software usually prevents users from accessing system functions.

2.2.4 /facet browser

E-culture implemented their own faceted search [2]. As a standard facet browser it allows the user to explore the knowledge by filtering the information in arbitrary order. It provides easy and convenient way to navigate through data collections. One facet represents one dimension of the underlying data (for example "Date creation"). By combining constraints specified by multiple facets, a user can specify complex queries through intuitive interface.

The /facet is a nice example of rich internet application. It uses *Ajax* for all communication, so the page does not need to fully reload. Only when we select a founded object, its description is shown on its dedicated page as in other types of searching. Of course this approach has also some limitations. User cannot go back in the browser and he cannot send the *URL* with selected facets and so on to his friends. But I believe that this is not so important as having fast response from the application.

While the dataset is huge, it is not possible to show all the facets, since the amount of information would be overwhelming for the user and it would take too much space on the screen. As the facet browser was built as a universal interface for any type of dataset. It is able to derive which facets should be offered to a user from analysis of the dataset. For example the *rdfs:label* property is used to specify the name of the facet and the *RDF*[8] properties are used as facets, after filtering "schema level" properties of *RDF*, *RDFS*[9] and so on. Of course this function might not always give desired results. Sometimes it can occur that the result will be associated with too many facets and the interface will become hard to use. For that reason manual refinement of the generated facet configuration is supported, which is very important, since it allows developers to tune the interface for specific end users, who might have exact needs. Facets can be easily selected from top menu, which also group the facets to categories.

While an individual facet can sometimes contain a lot of values which would not lead to any result. These values are dynamically removed, preventing the user from running into dead ends. Also when there is no item in the result set having a particular property, the facet

8 http://www.w3.org/RDF/
9 http://www.w3.org/TR/rdf-schema/

associated with the property is removed form the interface.

When an individual facet is representing constraint which is hierarchically structured (for example art styles and periods), the hierarchy is shown in the user interface, enabling the user to orient better in this particular dimension of data. User can also switch to standard flat list representation. The tree of the hierarchy can be very deep leading to the biggest disadvantage of navigating complex tree structures: it can be hard to navigate to the particular value while it can be very deep in the structure. Thus the search interface in addition to the navigation interface is provided. Every facet has its own keyword search field with a dynamic suggestion facility. This function can make the interaction much faster while navigating through deep tree but also when navigating in long flat lists (for example facet with creators' names).

Browser does not only support flat and hierarchical lists, but it has a mechanism for visualization and interaction plug-ins. An example of this is the time line, which is used to visualize time-related facets. Unfortunately it does not provide any interaction to allow the user to constraint some property. Another example could be the map showing related geographical information, but it looks like it is not working right now.

Other very useful function is cross browsing. It allows the user to switch the type of searched resource while keeping his constraints where appropriate. This makes the investigation of particular resources connected with the search much faster.

I really appreciate the constraints display and the ease of control. Clicking the value highlights it and puts it into constraints. Another click removes it from the constraints. Also the constraints display allows us to remove the constraints.

As opposed to basic/advanced search, the user can list the results. Grouping by arbitrary property is also supported.

I see one big problem – the /facet browser is not reliable enough. It often happens that the message “There was a problem retrieving the XML data” occurs, which causes the application to halt and the user has to start inputting everything again, because only solution is usually to reload the page. This can be really bothersome. Also the speed is sometimes not acceptable for comfortable browsing.

The user can sometimes run into troubles because there are very close names for different categories in the knowledge base. Also probably there are some problems with character sets. For example let say that we are looking for “Česká Republika” (Czech Republic) in Places. We select Place Type facet and there we must select “nation”, while the Czech republic is not considered as “country” or “state” in the semantic data hierarchy. This can be confusing for a lot of people who consider word “nation” more related to inhabitants than places. Of course we can also use the Name facet, but there we run into troubles. The search text box does not work with character “Č”, so it does not offer any results when we are inputting letters nor when we wrote whole word and pressed enter and while there is not “Czech republic” (written in English) in knowledge base, we cannot find “Česká Republika” with help of Name facet. As you can see, bad manipulation with character sets can make it very difficult to find a resource in the repository and the user can, for good reasons, believe that the resource is not inside the repository.

2.3 Resume

I think the interface for this application is more than satisfactory. It gives the user a lot of freedom with fine-grained search. Also I like the design which helps the user to understand the interface very fast – the unification makes it much easier.

But of course there are some problems which I can reproach. Sometimes the speed can be annoying, mostly in /facet browser or relation search. Also the reliability of the /facet browser should be better. Maybe it could be helpful to think about the *URL*s in basic/advanced search, because the current ones are really long, but I suppose it is not so important, but maybe sometimes user wants to send a message to his friend with a link with his search results and then it does not look nice.

According to *E-Culture Multimedian Demonstrator* paper, *RDF* database maintains also metaphone (sounds-like) index to words. Unfortunately it does not always help the user when he makes error in spelling. For example search for "picaso" or "pikaso" does return 0 results. In this case I would like to see some kind of dynamic advice based on statistics or word similarity as for example in *Google*[10] search. I feel this could be helpful while this is also search engine.

3 *Freebase.com*

3.1 Introduction

Freebase aims to be an open database of all the world's information. It has a lot of aspects of *Web 2.0* but on second sight it is very close to semantic web technologies. That is why it was labelled as *Web 2.7* application at a presentation in *2007 Java One Conference* [3].

Freebase is collaborative project. It encourages the users to add the content. It is also obtaining the data from large open data sets like *Wikipedia*[11]. It is easy to find a lot of topics with description copied from *Wikipedia*. *Frebase* is interested in all topics, even the ones which would be too obscure for encyclopaedia as opposed to *Wikipedia*.

As *Web 2.0* aspects I can list collaborative approach and rich user interface. It has also aspects of semantic web. Data is structured and represented as a big graph with a set of nodes and relations between them. The data can be queried via open *API* (application interface). Its type system is very close to *RDFS*. The core *RDFS* constructs map almost exactly into it: *rdfs:resource* > */type/object*, *rdfs:class* > */type/type*, *rdfs:property* > */type/property*, *rdfs:label* > */type/object/name* etc. But the *subClassOf* and *subPropertyOf* are missing. Sub typing can be expressed, but the query engine itself does not infer types. Unfortunately *Freebase* has no explicit support for *RDF*, but the *RDF* interoperability is one of the planned features.

A lot of the efforts related to semantic web concentrate on ontologies. Ontologies can be very useful for experienced users, but sometimes for most of the users, they may seem too academic and unfriendly. The *Freebase* type system is solving this, because it is basically a flexible and editable ontology which is edited by community keeping it understandable for majority of users. One topic can have many types (for example Person, Film actor, Director).

As the *Freebase's* web site is only one part of the project. I will also try to mention the *Freebase's* "user interface for programmers", which is used to query data from *Freebase* and use them in other applications.

3.2 Type system

As data of the *Freebase* is structured, every topic can have different types. Each type contains properties which hold the structured information. Properties are like fields in the database. For example type "Film" contains "Directed by", "Performances", "Music by" and

10 http://www.google.com/
11 http://en.wikipedia.org

other properties. Every type has also default properties, which are for example “name” and “image”. Each property has its type, which can be of basic type (“Text”, “Date/Time”, etc.) or a link to any other type in the system. The value can also be a set of topics of specified type. Without the properties the type is more like a simple tag. Every type can have its description and include other types. That means that the topic will be of the other included types and will have the properties of the included types. For example type “Film actor” includes “Person” type. Also it can be specified where we want our property to appear on the page about the topic of that type.

Of course because the data modelling is not a small and easy science discipline, the types can be edited only by domain administrator, but the user can create his own types and suggest them to the administrator. The interface for creation or editing of the types is very modern and intuitive. The user does not need to know low level database mechanisms like indexing or specification of the keys. Everything is handled automatically by the system.

3.3 User Interface

3.3.1 Basic search

There is more ways to search the *Freebase*. I will start with search via the search input box which is shown on every page.

While inputting letters into the text box, the *Ajax* suggest is used to offer the possible results containing the entered letters in the name. It shows the name of the item (on *Freebase* called topics) and its types, which is very useful while many things can have same name but different type. For example when we enter “Company”, we can select, if we want a resource of “Film” or “Play” type, which of course refers to different results. We can also move the mouse pointer above the list and then a small window with shortened description is shown, providing the user easy way to confirm that this is the thing he is looking for. If one item from the offer list is selected, the page with the result is immediately shown. Freebase currently supports subset of the *Lucene*[12] search syntax.

If we do not select the item from the offer list, we get to the page with search results. This page offers us to narrow the results by type (lets assume that we did not select topic of the type “Type”). Narrowing looks pretty similar to facet browsing. Unfortunately it is not. We can only select type of the topics, we want to list. For example we have searched “Ford”. Then we input to narrow search the type of “Person”. Now we see only topics with type of “Person”. User interface allows us to input another type, but if we add the type of “Film actor”, we will still see the same results. Also if we add type of “Building” we will see the same results. So it only looks like a facet browser, but it is narrowing results only for the first type and it looks like the other specified types are not narrowing the search nor working as union of topics with specified types. This I consider not so predictable behaviour. My expectations were that it will work like facet browser.

The search results page also offers listing. It lists 30 topics on one page. Unfortunately this value cannot be edited by user. When the user inputs the type to narrow the results, he is forced to use the *Ajax* suggest list, while the application want to specify the exact name of the type (this is indicated by the icon next to the input box). This can be sometimes very annoying while the user has to wait for the list to appear even though the names of the types are unique, so it would be possible to narrow the search even with simple written specification.

I would also appreciate the restraining of offered types. Like it is done in *E-Culture*'s /facet

12 http://lucene.apache.org/java/docs/queryparsersyntax.html

browser. Because it is useless when we select a type and we do not get any results.

3.3.2 Browsing via types

Freebase supports interface for browsing via types. Types are grouped to categories (sometimes called domains). The user can go to page with an overview of types grouped into the category. There is also the list of the administrators of the domain and the featured topic. It is easily understandable. When we select a type we will get to the page where we can narrow the results. We can get to the same page, when we type the name of the type to the search box and select from the list the topic with type "Type Profile".

The page offers narrowing and listing the results and link to the schema of the type. This time the page is very close to the facet browsing, because it offers the user to define values of at most nine properties (for example "Name", "Directed by" etc.). It shows first nine properties excluding some of them such as images, the summary, dates and numerical properties. When the expected type of the property is not just a text, but for example "Film actor", the *Ajax* suggest is used to help the user to enter the name of the "Person", but this time it is not required to use the list.

The constraints can be easily removed. Again there is no functionality to remove the items from the suggested list, which would lead to 0 results. But I must admit that it is not a big issue and the application is very fast as opposed to E-Culture.

3.3.3 View of the topic

Each topic has its page with its description. The page contains all structured data with a lot of links. It offers all functionality needed to edit the topic, because it is still collaborative site. For example we can flag the topic as offensive or edit the description.

The important function is page history, which is similar to the *Wikipedia's* one. It makes it easy to revert the changes, which is very important, while not all the users always behave correctly.

In overall the page is easy to understand and the user can obtain a lot of structured information very fast.

3.4 API for applications using Freebase

Freebase offers the open *API* and free data (under *Creative Commons*[13] licence) to encourage the developers to build new applications incorporating knowledge from *Freebase*. For querying the knowledge base the *MQL* (*Metaweb Query Language*) is used. *Metaweb* is the name of the company working on the *Freebase*. This language is using *JSON* (*JavaScript Object Notation*)[14] which defines the query. Every query is a *JSON* object and every result from the server is also a *JSON* object. That makes it easy to create queries in *JavaScript* and the developer does not need any other kind of technology layer to use the *Freebase* data. The server service is implemented on top of the *HTTP*, meaning that to query the server we only need to send the *HTTP* request with appropriate *URL* and the result is sent as a response. Cookies are used for user authentication.

The *MQL* grammar is published on the web site of *Freebase*. As the most interesting attribute I would call the ease of use. There is even query editor on the website where the developer can construct his queries and look at the results. This way he can became familiar with

13 http://creativecommons.org/
14 http://json.org/

the language without the need to know *XMLHttpRequest* facilities.

MQL is a subset of *JSON* so the developer is not forced to use some other technology than popular *JavaScript*. Constructing the read query is very easy. The developer only specifies what he knows, which is usually some kind of user input for example name of the film. Then he specifies what he wants to know by assigning the null or empty array value to that property.

For example to find the actors starring in the "Requiem for a dream", we can construct a query like this:

```
{
  "query":[{
    "actor":[],
    "film":"Requiem for a dream",
    "type":"/film/performance"
  }]
}
```

[] specifies that we want to return the array of values of actors. The other lines specify filtering of the data. In this case we are only interested in objects of */film/performance* type, which have the property "film" set to "Requiem for a dream". The returned result looks like this:

```
{
  "code":"/api/status/ok",
  "result":[{
    "actor":["Ellen Burstyn"],
    "film":"Requiem for a Dream",
    "type":"/film/performance"
  },{
    "actor":["Jared Leto"],
    "film":"Requiem for a Dream",
    "type":"/film/performance"
  },{
    "actor":["Jennifer Connelly"],
    "film":"Requiem for a Dream",
    "type":"/film/performance"
  },{
    "actor":["Marlon Wayans"],
    "film":"Requiem for a Dream",
    "type":"/film/performance"
  }]
}
```

As you can see, the result is symmetrical to the query. Response objects have the same properties, but the requested values were filled in. This example uses good knowledge of schema of the objects. That all the performances are saved as a type */film/performance* with properties containing film and actor. This information is easy to obtain, while there is nice interface for browsing types and examining their properties.

In the second example, we are asking the same question, but this time we will be interested only in objects of */film/film* type and then we use nested query to obtain names of the actors from its property of */film/performance* type:

```
{
  "query":[{
    "name":"Requiem for a dream",
    "starring":[{
      "actor":null
    }],
    "type":"/film/film"
  }]
}
```

The result then look like this:

```
{
  "code":"/api/status/ok",
  "result":[{
    "name":"Requiem for a Dream",
    "starring":[{
      "actor":"Ellen Burstyn"
    },{
      "actor":"Jared Leto"
    },{
      "actor":"Jennifer Connelly"
    },{
      "actor":"Marlon Wayans"
    }],
    "type":"/film/film"
  }]
}
```

As you can see, the result is again symmetrical to our query. Also notice how we asked for the array and that it was returned the same way only filled with values and the properties, which we had asked for.

We can also ask for whole objects. We only need to use empty set {} and the service will return single value expanded as an object. For example we want to know everything about person with name “Jared Leto”:

```
{
  "query":[{
    "*":{},
    "name":"Jared Leto",
    "type":"/people/person"
  }]
}
```

The “*” is a wildcard character which specifies that we want to return all the properties. {} specifies that we want to return the expanded object as a result, not just a value. The shortened result than looks like this:

```
{
  "code":"/api/status/ok",
```

```
  "result":[{
    "date_of_birth":{
      "type":"/type/datetime",
      "value":"1971-12-26"
    },
    "gender":{
      "id":"/topic/en/male",
      "name":"Male",
      "type":["/common/topic","/people/gender"]
    },
    "height_meters":{
      "type":"/type/float",
      "value":1.75
    },
    "parents":[],
    "place_of_birth":{
      "id":"/guid/9202a8c04000641f80000000000cc33c",
      "name":"Bossier City, Louisiana",
      "type":["/common/topic","/location/citytown"]
    },
    "type":"/people/person",
    "weight_kg":null
  }]
}
```


MQL support variety of wildcards, sorting, limiting the number of results and intersection set operation.

Writes with *MQL* are very similar. Only difference is that writes have two other directives (create, connect) and some directives are missing (e.g. sort, limit, while these are useless when writing).

The *Metaweb* also offers *Mjt* (*Metaweb JavaScript Templating*) [4] [5], which is open source *XHTML* templating engine. This engine can be used to generate *XHTML* output (interface) from the results received from *Freebase*. It works pretty similar to any other templating engine. The only difference is immediate support of *Freebase*. This way it is possible to create complex web applications based only on *HTML* and *JavaScript*, while all the data are obtained from the *Freebase*.

3.5 Resume

I think *Freebase* is very interesting project. The user interface is easy to follow and the type system is used as it should be. The interface is fast and it is well designed – even the *URL*s are nice looking. Of course there are some minor problems, but nothing really important. Also the knowledge base is pretty big, while the data are copied from other sites with help of data mining. So there is not such a big problem with chicken-and-egg problem, which is pretty common in most projects depending on user collaboration.

API for developers is very easy to understand, but I still lack the *RDF* interoperability support. *MQL* is not standardized in any way and it is just technology of one company. This makes it difficult to combine resources from different data sources, which is characteristic for semantic web. Also it uses only its own ontology and it completely lacks support for any other

ontologies. It does not reuse any of the well known vocabularies and it does not share its own ontology, which makes it impossible to link the data with something else. In this way it is very closed application which is not using semantic web technologies to become a part of the big shared knowledge. It is still just a closed knowledge, even though it can be edited/queried by the users and populated through data mining. I feel that this is its biggest problem, while it does not help semantic web to become a big knowledge source.

4 *Tabulator*

4.1 Introduction

Tabulator [6] is generic browser for linked semantic data. It is built to browse an open web of *RDF* resources. It navigates between relationships between data. It will never offer the best intuitive user friendly interface as an application specifically tailored for its purpose, but the *Tabulator* is more an experimental application, of which purpose is more to explore the possibilities of semantic web and how to bring the power of domain-specific user interfaces to a generic browser. It does not have expectations of providing such as intuitive interface as domain-specific applications, so there were not any usability tests. It runs always on the client's computer, so it does not have centralized design, thus can be easily scaled up. Also it forces the application to perform linked data protocols, which are part of the research motives, because there is still a lot of semantic web data which is isolated from other data and the developers want to encourage creation of the links. *URI* for the browser represents an invitation to look it up.

There are two versions of the *Tabulator*. One is *Firefox* extension, which is in alpha version. The other one is web application and I will be writing about that one, in stable version 0.8. The web version is more easily accessible for new users, because it does not require any installation. Whole browser is written in *JavaScript* and runs inside a web browser. Unfortunately currently it runs only under the *Mozilla Firefox* web browser. Some adjustments in web browser's configuration are needed, because *Tabulator* connects to other servers, which is implicitly considered dangerous, while some pages could get to sensitive data with help of cross-site scripting.

The browser performs limited inference. For example it merges nodes which are according to *owl:sameAs*[15] same. To present as much data as possible it mixes different data sources and needs to dereference the links contained in *RDF* descriptions. Of course this cannot be done the simplest way of downloading the data whenever it comes across a new *URI*, because it could continue without limit in unbounded web. The browser has to anticipate what data could be useful for the user.

The *Tabulator* is using *SPARQL*[16] to perform queries, which can be specified by the user. It does not support whole *SPARQL* specification, but the most used basics are implemented. As a dynamic store of *RDF* data is constructed by accessing external sources while browsing, the *SPARQL* engine might not give all the results, if the results exist somewhere else than in *URI*s contained in loaded resources.

15 http://www.w3.org/TR/owl-ref/
16 http://www.w3.org/TR/rdf-sparql-query/

4.2 User Interface

4.2.1 Exploration mode

In this mode we can navigate through *RDF* graph in tree view in flexible table structure. Even though we are browsing heterogeneously typed graph tree view feels naturally. While traversing to other nodes the *Tabulator* is downloading data from links, which could contain data about the nodes.

The mode occupies the upper part of the browser. It begins with several initial *RDF* nodes and the new nodes can be added by *URI*. Node can be expanded and its relationships are shown. If the property value is an *URI* the collapsed node is shown. Next to the every node is an icon, which indicates status of the retrieval for that *URI*, which is pretty useful, while it gives the user information about what data are loaded into the browser. The nested nodes can get quickly very deep, but the user can place the selected node as a starting node by shift-clicking the node's expand/collapse icon or he can create other starting point with the node as a root by double-clicking, so it is partially solved. When the user wants to see the *URI* of the node, he can click on the node cell and the *URI* (when possible) will be shown in the top *URI* bar.

As the browser mixes data from different sources, it is essential that the user can verify the source of the data. This is solved by the overview of the sources, which is shown almost in the bottom with legend and log. When the user clicks on any data cell, the source is highlighted in the sources' pane. Then he can double-click on the source and the meta data of the source can be explored in the exploration mode.

We can also build simple queries through exploration mode by specifying patterns. The user does not need to about *SPARQL*, while the queries are specified by clicking. User can highlight property fields or objects using alt-click. Highlighting property field means, that we are looking for resources, which contain this property and it has the same relationship with “upper” node as the node we have selected the property in. For example if there is a list of people, we can expand the node about someone and click on the property field with his email. Then the query, which finds all the people from the list with an email, is constructed. There is also check box which can be used to make the property optional – so the nodes which do not have this property will not be refused. If we highlight object, we tell the *Tabulator* that we are looking for resources with the same value in the particular property. This way we can for example filter all women in the group by specifying sex property to be female. The *Tabulator* performs the query after the click on “Find All” button and the results are shown in analysis area. An icon is shown next to the properties, which could be shown in Map or Calendar view in the analysis part.

4.2.2 Analysis mode

Tree view is not well applicable for data with well-defined structure, where the user wants to sort them, compare them etc. That is why the Tabulator have analysis mode. This part occupies middle part of the browser. It can be used after performing a query – it uses the results from the queries. It is composed from several views, which can be switched. The views are written in *JavaScript* and extensible. Experienced user can write and upload his own view, which only needs to provide a way how to draw query results' information.

Basic views are: table, map, calendar, time line and *SPARQL* query editor.

The most often used view is the table, because it is universal way to display a lot of objects with same properties, which can be easily compared and sorted. By clicking an object in table, it is opened in exploration mode. This way the analysis and exploration are nicely interleaved and

it is very useful. Query results can be exported to the *HTML*, which is important, because otherwise there would not be any easy way to transfer and use the results somewhere else. Every time the query is composed, it is saved independently. We can switch between various queries' results in the view.

The map view can be used to visualize the geographical data on the *Google Maps*[17]. It can show the results from more queries – so the list of queries has check-boxes instead of radio buttons in table view. To show the data, it is necessary that the result contains geographical information (longitude and latitude). This can be easily examined from the exploration mode, where these properties are marked by an icon. By clicking the icon representing a resource in the map view, we can obtain some more information or by double-click we can put the node to the exploration view.

The calendar view is used to show the date/time data in common way. Again it can show data from more queries. By double-clicking the node can be put to the exploration view. Unfortunately clicking the node provides additional information in the bottom part of the viewer, where it is not so well visible. The time line view works almost the same, only the additional information is shown on better place. It was developed in Simile[18] project.

The other tab is the *SPARQL* editor. It is not a view, because it does not show results of the queries, but the queries itself. It is very useful feature, as the user can control and edit the query constructed in exploration mode. Expert user can also create his own more complex queries than the ones constructed in exploration mode, but of course it might not be useful for most of the users.

4.3 Resume

Tabulator shows that it is possible to make a generic browser, which can navigate through current semantic data on the web instantly. I like the way it uses multiple sources of data and mixes them together to provide user with as much information as possible. It also hides very well the underlying techniques as RDF and OWL, making it better understandable for inexperienced users. The effort in showing a lot of data in limited space can be very well noticed as it does not use usual circles with arcs for graph visualization, which takes too much space.

Of course *Tabulator* cannot beat the domain-specific applications in their very user-friendly interfaces, but I suppose that this will be always the prize for universal solution. The interface could be better, if there were sufficient user interface tips in ontologies, which could be used by such a generic application to provide more effective and useful user interface for data domains previously unknown. As for now *Tabulator* offers several views on domain-specific data, which prove that it can be extended for other types of data.

Tabulator offers very easy way to create instant mashup[19] of the data without any application specific programming. But of course this is easy only for very experienced users.

Sometimes it takes a lot of time to process the query, but this is the price for using interpreted *JavaScript* and different data sources. The number of *URI*s which have to be dereferenced is shown in the right upper corner, so it gives rough information about remaining time to process the query.

I did not like that it takes a lot of time to learn the user interface. I suppose that it is because this kind of applications is not often used and that semantic web gives so many possibilities to work

17 http://code.google.com/apis/maps/index.html
18 http://simile.mit.edu/timeline/
19 http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)

with the data, which makes the interface very hard to design. I also think that the *Tabulator* is more something like proof of concept and not final application. In this way I really like the way it composes a lot of information from different sources, but sometimes it can make user interface chaotic. So the usability depends on the user's needs. Also I consider the connection between exploration, analysis and source views to be very useful, as we can always switch very fast between them. Also user can become familiar with specification of queries by clicking, very fast and after some time it feels natural. In any way I do not like to criticise the interface, because it is experimental application. But in my opinion the major problem of the interface is that it is too different from anything else, which makes it somewhat harder to learn. Still I appreciate huge amount of functions and possibilities given to the user. Unfortunately this can be appreciated only by experienced user with patience.

5 Conclusion

I tried to describe three applications, which use semantic web technologies in some extent. Two domain-specific applications have very user-friendly and modern interface. They show how semantic web technologies can be used to offer more information and more functions. The *Tabulator* is example of what could semantic web bring in general as it is not targeted on specific domain. It is considerably less user friendly, but it works with any data and shows highly heterogeneous domains. Most likely it will never become a widely used application, but it well demonstrates how the semantic web architecture should be used.

All projects use *Ajax* and some other *Web 2.0* aspects, which shows that the *Web 2.0* is nowadays standard at least in user interfaces.

I consider the difference between two domain-specific applications and the *Tabulator* as the most interesting. *Tabulator* demonstrates how the future web could look like, but I do not think that this way web would be friendly for the most of the users. I believe that the semantic web technologies will and have found [7] practical adoption between specialized communities, where the domain and scope are more limited. But we should become aware of the most users, who do not always want as much information as possible, but who want just some kind of “tabloid” web with shiny, entertaining pages. Also sometimes we just need only some basic information and do not want to be overwhelmed by information for experts.

The technology is here but it is important to use it the right way. *Tabulator* shows the way how the semantic web should be built. For example there is a lot of accent given to linking between resources and to usage of appropriate vocabularies. I think in this sense *Tabulator* can be very useful for the community.

References

[1] Schreiber G, et al. MultimediaN E-Culture Demonstrator. In: International Semantic Web Conference, 2006.
(http://www.cs.vu.nl/~guus/papers/Schreiber06a.pdf)

[2] Hildebreand M, Ossenbruggen J, Hardman L. /facet: A browser for heterogeneous semantic web repositories. In: ISWC ’06: Proceedings of the 5th international conference on the Semantic Web, 2006.
(http://www.cwi.nl/~media/publications/iswc06.pdf)

[3] Spivak N, et al. Developing Web 3.0. In: JavaOne Conference, 2007.
(http://bblfish.net/work/presentations/2007/BOF-6747.pdf)

[4] Fitzgerald M. Metaweb JavaScript Templating. In: O'Reilly Short Cuts.
(http://www.metaweb.com/mjt-shortcut-final3.pdf)

[5] Matzan J. Inside Freebase Mjt Web Applications. In: O'Reilly Short Cuts.
(http://www.metaweb.com/finalMetawebApps2.pdf)

[6] Berners-Lee et. al. Tabulator: Exploring and Analyzing linked data on the Semantic Web. In: The 3rd International Semantic Web User Interaction Workshop, Athens, Georgia, 2006.
(http://swui.semanticweb.org/swui06/papers/Berners-Lee/Berners-Lee.pdf)

[7] Herman I. State of the Semantic Web. In: Norwegian Semantic Days, 2007.
(http://www.w3.org/2007/Talks/0424-Stavanger-IH/Slides.pdf)

Other sources used:

E-culture project at: http://e-culture.multimedian.nl/

Freebase at: http://freebase.com/

Tabulator at: http://dig.csail.mit.edu/2005/ajar/ajaw/